

Skriptum

Modellbildung und Simulation Mechatronischer Systeme



Fachhochschule Vorarlberg
Masterstudiengang Mechatronik

Markus Andres, Thomas Schmitt

Dornbirn, 28. September 2011

Vorwort

Dieses Skriptum wurde von den Autoren während ihrer Tätigkeit als Wissenschaftliche Mitarbeiter im Department of Engineering (DoE) an der *Fachhochschule Vorarlberg* im Zeitraum Oktober 2009 bis September 2011 verfasst.

Inhalt des Skriptums

Der erste Teil „*Grundlegendes*“ des Skriptums soll den Leser für das Thema Modellbildung und Simulation motivieren. Dazu müssen zuerst einige Begriffe definiert werden. Anschließend werden verschiedene Wege vorgestellt, um mathematische Modelle physikalischer Systeme zu erstellen. Danach werden wichtige Modelleigenschaften, sowie der Detaillierungsgrad und dessen Einfluss auf Modelle und deren Ausführungszeit vorgestellt. Ein paar Worte zur Simulation von Modellen beenden dieses Kapitel.

Im zweiten Teil „*Einführung in die Zustandsraumdarstellung*“ wird ein einfaches elektrisches System mittels Differentialgleichungen beschrieben. Genauer gesagt werden physikalische Systeme mittels gewöhnlicher linearer Differentialgleichungen 1. Ordnung mit konstanten Koeffizienten beschrieben. Sodann werden diese Systeme in den Zustandsraum übergeführt. Abschließend werden die Gleichungen der Zustandsraumdarstellung (engl. state-space representation) etwas genauer interpretiert und gezeigt, wie diese - zumindest für den skalaren Fall - analytisch gelöst werden können.

Im dritten Teil „*Klassische Modellbildung: Beschreibung physikalischer Systeme mit DAEs*“ wird die Erstellung von mathematischen Modellen physikalischer Systeme mittels Gleichungen und Differentialgleichungen (auch differentialalgebraische Gleichungen, kurz DAEs) beschrieben. Die Schritte zur Erstellung des Gleichungssystems werden anhand eines konkreten Beispiels erläutert. Nachdem das mathematische Modell erstellt wurde, wird gezeigt wie man auf verschiedenen Wegen zu einem Simulationsergebnis gelangt. Dazu muss das mathematische Modell in eine entsprechende Form gebracht werden. Der einfachste Weg ist die Erstellung eines Blockschaltbilds, welches direkt aus den DAEs abgeleitet wird. Des Weiteren wird gezeigt wie man aus den DAEs sehr einfach zur Zustandsraumdarstellung gelangt. Basierend auf der Zustandsraumdarstellung wird abschließend eine Übertragungsfunktion (engl. transfer function) des physikalischen Systems erstellt und mit Hilfe von MATLAB simuliert.

Im vierten Teil „*Grundlegende Zusammenhänge physikalischer Systeme*“ werden elek-

trische, mechanische und hydraulische Systeme behandelt, wobei die mechanischen Systeme zusätzlich in translatorische und rotatorische Systeme unterteilt werden. Dieser Abschnitt soll die Ähnlichkeiten dieser Domänen verdeutlichen und dient als eine Vorbereitung für die im nächsten Kapitel vorgestellten Bondgraphen.

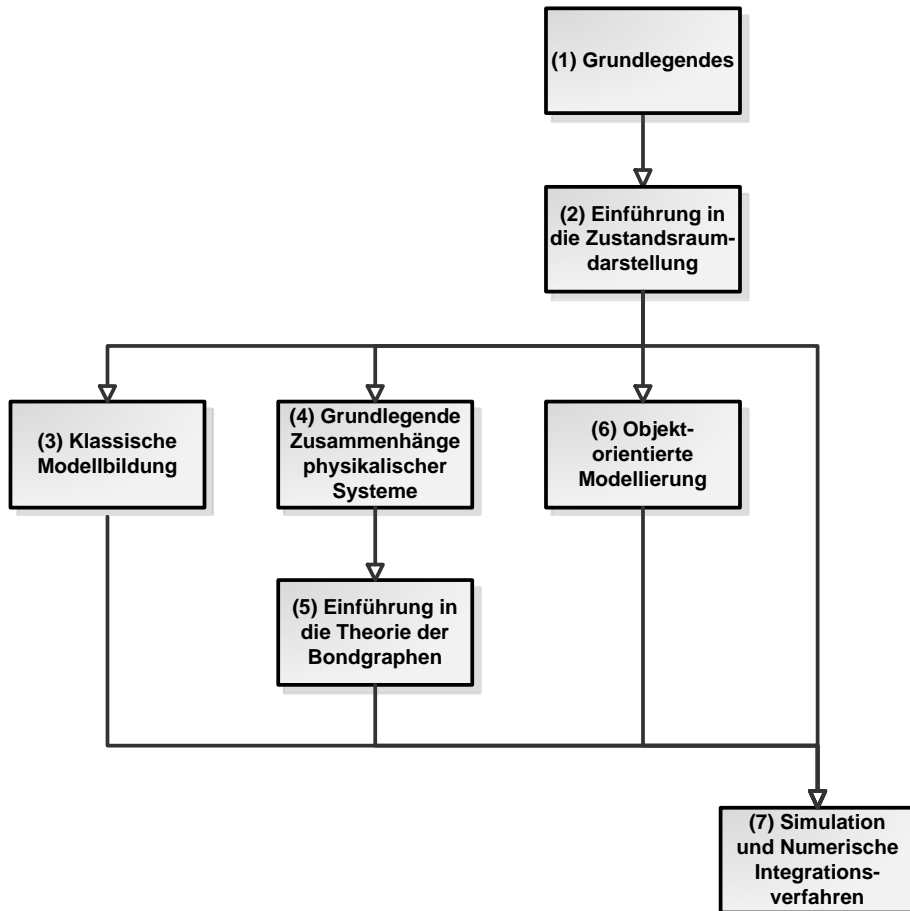
Im fünften Teil „*Einführung in die Theorie der Bondgraphen*“ wird die Theorie der Bondgraphen (engl. bond graphs) vorgestellt. Diese alternative Modellierungstechnik beruht auf der Energieerhaltung geschlossener physikalischer Systeme und modelliert dessen Energieflüsse graphisch. Um das Beispiel des ersten Abschnitts via Bondgraphen modellieren zu können, werden die wichtigsten Bondgraph-Elemente vorgestellt. Anschließend wird gezeigt, wie man mit Hilfe der Bondgraphen zur Zustandsraumdarstellung bzw. zum Blockschaltbild gelangt. Dazu müssen die Bondgraphen kausalisiert werden. Anschließend werden weitere wichtige Elemente zur irreversiblen bzw. zur reversiblen Energieumwandlung vorgestellt. Im nächsten Schritt werden sogenannte kausale Pfade vorgestellt. Beim Kausalisieren kommt es sehr oft zu sogenannten Konflikten, wie z.B. algebraischen Schleifen. Es wird gezeigt wie Konflikte in Bondgraphen erkannt und behoben werden können. Eine sehr gute Einführung in die Theorie der Bondgraphen ist in den Büchern [KMR06] [LG94] und [Cel91] zu finden.

Im sechsten Teil „*Objektorientierte Modellierung*“ wird ein mächtiges Konzept vorgestellt, welches der Modellbildung völlig neue Türen öffnet. Zu Beginn wird etwas auf die Entstehung der objektorientierten Modellbildung eingegangen. Nach einer Motivation für dieses Thema werden die Voraussetzungen für diese Art der Modellierung erläutert. Anschließend wird die objektorientierte open-source Modellierungssprache Modelica etwas detaillierter vorgestellt ((siehe [Ott09], [Ass10] und [Fri04])). Eine Notwendigkeit, welche sich aus der objektorientierten Modellierung ergibt ist die sogenannten symbolischen Vorverarbeitung (engl. symbolic pre-processing). Die Schritte und die damit verbundenen Probleme welche bei einer symbolischen Vorverarbeitung nötig sind, werden hier im Detail diskutiert und anhand zahlreicher Beispiele veranschaulicht.

Der siebte und letzte Teil „*Simulation und Numerische Integrationsverfahren*“ befasst sich mit der Simulation physikalischer Systeme [CK06]. Dazu wird anhand einfacher Solver wie z.B. dem sogenannten Eulerverfahren erklärt, wie physikalische Systeme auf digitalen Rechnern simuliert werden. Basierend auf diesen Grundlagen wird über die numerische Stabilität einfacher Solver diskutiert. Abschließend wird eines der bekanntesten Verfahren, das sogenannte Runge-Kutta Verfahren vorgestellt. Ferner sollen die unterschiedlichen Eigenschaften physikalischer Systeme (z.B. steife Systeme; engl. stiff systems) klar verstanden werden. Schlussendlich sollen die Studenten in der Lage sein, für viele Problemstellungen den richtigen Solver wählen zu können.

Wie ist dieses Skriptum zu lesen?

Die folgende Abbildung dient als Hilfestellung und Orientierung für ein Selbststudium dieses Skriptums. Die Kapitel „Grundlegendes“ und „Einführung in die Zustandsraum-



darstellung“ vermitteln das nötige Grundwissen. Nach diesen Kapitel werden drei Wege der mathematischen Modellierung vorgestellt, die je nach Interesse studiert werden können. Zum Schluss wird auf das Thema „Simulation und Numerische Integrationsverfahren“ eingegangen und gezeigt wie analytische Lösungen numerisch approximiert werden können. Natürlich können auch alle Kapitel in der vorgegebenen Reihenfolge gelesen werden, was viele Synergien zwischen den beschriebenen Techniken erkennbar werden lässt.

Danksagung

Unser Interesse an dem Thema „Mathematische Modellierung und Simulation physikalischer Systeme“ wurde im Jahr 2006 während eines Auslandssemesters an der Universität in Linköping/Schweden (LiU) geweckt. Hier erlernten wir in der Vorlesung „Modeling and Simulation“ - welche von Måns Östring gehalten wurde und auf dem Buch [LG94] basiert - die Grundlagen der Modellbildung im Speziellen mittels Bondgraphen und Systemidentifikation. Nach Abschluss des Bachelorstudiums der Mechatronik an der Fachhochschule Vorarlberg im Jahr 2007 folgte das Masterstudium mit weiteren geplanten Vertiefungen im Bereich Modellbildung und Simulation (M&S). Im dritten Semester besuchten wir die Vorlesung „Mathematical Modeling of Physical Systems“ von Prof. Dr. François E. Cellier vom Department of Computer Science an der ETH-Zürich. Anschließend verfassten wir unsere Masterarbeiten ([And09], [Sch09a]) in Zusammenarbeit mit Prof. Dr. François E. Cellier und Dr. Dirk Zimmer (der damals bei Prof. Cellier seine Dissertation verfasste und mittlerweile beim Deutschen Zentrum für Luft- und Raumfahrt, kurz DLR tätig ist). Während unserer Tätigkeit als Wissenschaftliche Mitarbeiter besuchten wir dann die Vorlesung „Numerical Simulation of Dynamic Systems“ von Prof. Dr. François E. Cellier an der ETH-Zürich.

Basierend auf diesem Wissen entstand das vorliegende Skriptum sowie die dazugehörige Vorlesung „Modeling and Simulation of Mechatronic Systems“ welche im 1. Semester des Masterstudiums Mechatronik and der Fachhochschule Vorarlberg unterrichtet wird. Weiters basiert das Modulfach „Virtual Physics“ - welches sich über die ersten drei Semester des Masterstudiums für Mechatronik sowie für Informatik erstreckt - ebenfalls auf dem vorliegenden Skriptum.

Dabei möchten wir uns in erster Linie bei Prof. Dr. François E. Cellier und dem Studiengangsleiter des Studiengangs Mechatronik Dipl.-Ing. Dr. Johannes Steinschaden bedanken, die es uns ermöglicht haben, sowohl die Vorlesungen an der ETH-Zürich zu besuchen, als auch unsere Masterarbeiten durchzuführen. Ein zusätzliches Dankeschön an Prof. Dr. François E. Cellier dafür, dass er uns nach wie vor als Gastdozent besucht, um unsere Studenten mit seinem Wissen zu bereichern.

Weiters möchten wir uns bei Dr. Dirk Zimmer für dessen tatkräftige Unterstützung während unseren Masterarbeiten und den hilfreichen Diskussionen während unserer Tätigkeit als Wissenschaftliche Mitarbeiter bedanken. Auch Ihm gilt ein zusätzlicher Dank dafür, dass wir unser Modulfach genauso nennen dürfen wie das von Ihm an der TU-München unterrichtete Fach „Virtual Physics“.

Weiters bedanken wir uns bei Dipl.-Ing. Alexandra Mehlhase von der TU-Berlin für die gute Zusammenarbeit sowie der Bereitstellung eines SVN-Servers zum Datenaustausch der Vorlesungen „MoSim (TU-Berlin)“, „Virtual Physics (TU-München)“ sowie unseren Vorlesungen „Modeling and Simulation of Mechatronic Systems“ und „Virtual Physics“.

Ein ganz besonderer Dank geht natürlich auch an das DoE, insbesondere an unseren Vorgesetzten, Herrn Dipl.-Ing. Horatiu Pilsan sowie an Dipl.-Ing. Dr. Franz Geiger und Dipl.-Ing. Dr. Reinhard Schneider, die uns nicht nur während des Studiums sehr viel beigebracht haben, sondern uns auch während unserer Zeit als Wissenschaftliche Mitarbeiter noch mit Wissen bereichert haben.

Viele Dank nochmals an Dipl.-Ing. Horatiu Pilsan, für den Freiraum und vor allem für die Möglichkeit, soviel Zeit in die Entwicklung des Skriptums sowie den dazugehörigen Vorlesungen zu investieren.

... mit dem Ziel das erlangte Wissen an die Studenten weiterzugeben ...

Markus Andres und Thomas Schmitt

Inhaltsverzeichnis

1. Grundlegendes	1
1.1. Was ist ein System?	1
1.2. Was ist ein Experiment?	2
1.3. Modellbildung - Warum?	3
1.4. Mathematische Modelle	4
1.4.1. Physikalische Modellbildung: White-Box Modell	4
1.4.2. Experimentelle Identifikation: Black-Box Modell	5
1.4.3. Semi-Empirische Modelle	5
1.4.4. Modelle die im Skriptum behandelt werden	5
1.5. Modelleigenschaften	5
1.5.1. Statische und dynamische Modelle	7
1.5.2. Kontinuierliche und diskrete Modelle	7
1.5.3. Modelle mit konzentrierten und verteilten Parametern	9
1.5.4. Lineare und nichtlineare Modelle	9
1.5.5. Zeitvariante und zeitinvariante Modelle	14
1.5.6. Deterministische und stochastische Modelle	15
1.5.7. Stabile und instabile Modelle	15
1.5.8. Modelleigenschaften die im Skriptum behandelt werden	16
1.6. Detaillierungsgrad von Modellen	16
1.7. Simulation	17
2. Einführung in die Zustandsraumdarstellung	19
2.1. Zustandsvariablen	20
2.2. Ein einführendes Beispiel	21
2.2.1. Bestimmen der Ein- und der Ausgangsgrößen	21
2.2.2. Bestimmen der Zustandsgrößen	21
2.2.3. Aufstellen der Zustandsgleichungen	22
2.2.4. Bilden der Zustandsraumdarstellung	24
2.3. Zustandsraumdarstellung	25
2.3.1. Eingrößensysteme (SISO-Systeme)	25
2.3.2. Mehrgrößensysteme (MIMO-Systeme)	27
2.3.3. Interpretation der Zustandsgleichung	28
2.3.4. Interpretation der Ausgangsgleichung	30
2.3.5. Graphische Interpretation der Zustandsraumdarstellung	30
2.3.6. Wozu eigentlich Zustandsraumdarstellung?	32

2.4.	Analytische Lösung der Zustandsgleichung	32
2.4.1.	Autonome skalare Systeme (Homogene Lösung)	32
2.4.2.	Inhomogene skalare Lösung	36
2.4.3.	Inhomogene Lösung	37
3.	Klassische Modellbildung: Beschreibung physikalischer Systeme mit DAEs	39
3.1.	Analyse der physikalischen Struktur	39
3.2.	Aufstellen der Komponentengleichungen	41
3.3.	Aufstellen der Topologiegleichungen	41
3.4.	Lösbares Gleichungssystem erstellen (<i>Kausalisieren/Sortieren</i>)	42
3.5.	Implementierung des Gleichungssystems	43
3.5.1.	Zustandsraumdarstellung (ODE)	43
3.5.2.	Blockschaltbild	45
3.5.3.	Übertragungsfunktion	46
3.6.	Simulationsergebnisse	49
3.6.1.	Simulation via Zustandsraumdarstellung	49
3.6.2.	Simulation via Blockschaltbild	50
3.6.3.	Simulation via Übertragungsfunktion	50
4.	Grundlegende Zusammenhänge physikalischer Systeme	53
4.1.	Elektrische Systeme	53
4.2.	Mechanisch-translatorische Systeme	54
4.3.	Mechanisch-rotatorische Systeme	55
4.4.	Fluss-Systeme (Hydraulische Systeme)	56
4.4.1.	Trägheit einer Flüssigkeit	56
4.4.2.	Flüssigkeitsspeicher - Der Tank	58
4.4.3.	Die Verengung (Querschnittsreduzierung)	59
4.5.	Entwickeln allgemeingültiger Gleichungen	59
5.	Einführung in die Theorie der Bondgraphen	63
5.1.	Akausale Bondgraphen	63
5.1.1.	Verlustbehaftete Elemente (R-Element)	64
5.1.2.	Energiespeichernde Elemente	66
5.1.3.	Aktive Elemente (Quellen)	67
5.1.4.	Verzweigungen	69
5.1.5.	Beispiele	71
5.1.6.	Lösen von Beispielen mittels standardisierter Lösungsansätze	76
5.2.	Kausale Bondgraphen - Kausale Analyse	82
5.2.1.	Kausalisierung der Quellen	85
5.2.2.	Kausalisierung der passiven Elemente	86
5.2.3.	Kausalisierung der Verzweigungen	88
5.2.4.	Modellierung der passiven elektrischen Schaltung mit kausalen Bondgraphen	91

5.3.	Aufstellen der Gleichungen anhand des Bondgraphen	93
5.3.1.	Zustandsraumdarstellung	93
5.3.2.	Blockschaltbild	95
5.4.	Weitere Bondgraph-Elemente: Energieumwandlung	95
5.4.1.	Widerstandsquellen	95
5.4.2.	Transformatoren	96
5.4.3.	Gyratoren	98
5.5.	Kausale Pfade	99
5.6.	Das Dualitätsprinzip - Duale Bondgraphen	103
5.7.	Kausalitätskonflikte in Bondgraphen - Kausale Analyse	105
5.7.1.	Einblicke in das Modell mittels kausaler Analyse	105
5.7.2.	Algebraische Schleifen in Bondgraphen	106
5.7.3.	Strukturelle Singularitäten	112
5.7.4.	Eliminieren von strukturellen Singularitäten	114
5.8.	Die vier Grundvariablen der Bondgraphen-Modellierung	123
5.9.	Modulierte Elemente	125
5.9.1.	Sensoren	126
5.9.2.	Modulierte Elemente zur reversiblen Energieumwandlung	127
5.9.3.	Modulierte Quellen	129
5.9.4.	Beispiele	129
6.	Objektorientierte Modellierung	137
6.1.	Zur Geschichte der Modellierung	137
6.2.	Motivation	138
6.3.	Modelica Grundlagen	142
6.3.1.	Was ist Modelica eigentlich?	143
6.3.2.	Modelica und Dymola	144
6.3.3.	Umsetzung in Modelica	144
6.3.4.	Modell-Kosmetik in Modelica	148
6.4.	Voraussetzungen für die komponentenbasierte Modellierung	152
6.4.1.	Schnittstellen	152
6.4.2.	Akausalität	155
6.5.	Dekomposition technischer Systeme in Klassen	157
6.5.1.	Klassen, Objekte und Schnittstellen	157
6.5.2.	Schnittstellen in Modelica	159
6.5.3.	Die Komponenten in Modelica	160
6.5.4.	Packages in Modelica	164
6.5.5.	Verschalten der Komponenten	165
6.6.	Objektorientiertheit in Programmierung und Modellierung	166
6.6.1.	Konzepte aus der objektorientierten Programmierung	167
6.6.2.	Verwandschaft der objektorientierten Modellierung zur objekt-orientierten Programmierung	168
6.6.3.	Objektorientierte Modellierung und Bondgraphen	171

6.7.	Die Fähigkeiten von Modelica	171
6.7.1.	Kausale Blöcke Blocks	172
6.7.2.	Komposition und Connect Statement	172
6.7.3.	Vektoren {} und Matrizen []	174
6.7.4.	Strukturen: record	175
6.7.5.	Bedingte Ausdrücke: if/then/elseif/then/else	176
6.7.6.	Eventbasierte Gleichungen: when	177
6.7.7.	Schleifen: for/in/loop	180
6.7.8.	Globale Variablen: inner und outer	181
6.7.9.	Prozedurale Algorithmen: algorithm	182
6.7.10.	Funktionen: function	183
6.7.11.	Die Vererbung: extends	184
6.7.12.	Austausch von Submodellen: replacable/redeclare	189
6.7.13.	Andere Programmiersprachen: C, Fortran	192
6.7.14.	Annotations: annotation	193
6.7.15.	Skripte in Modelica	194
6.8.	Strukturvariabilität	195
6.9.	Symbolische Vorverarbeitung	196
6.9.1.	Sortieren von Gleichungen: „Index 0“ Systeme	196
6.9.2.	Algebraische Schleifen: „Index 1“ Systeme	204
6.9.3.	Strukturelle Singularitäten: „höhere Index“ Systeme	209
6.9.4.	Eine Alternative: Direkte Lösung von impliziten Gleichungssystemen	213
6.9.5.	Zusammenfassung	214
7.	Simulation und Numerische Integrationsverfahren	217
7.1.	Einleitung und Allgemeines	217
7.2.	Die Annäherung an die Realität	218
7.2.1.	Fehler in der numerischen Integration	219
7.2.2.	Fehlerbegriffe	220
7.3.	Einschrittverfahren	221
7.3.1.	Eulerverfahren	221
7.3.2.	Runge Kutta Verfahren	235
7.3.3.	Rückwärtsinterpolation	242
7.4.	Mehrschrittverfahren	242
7.4.1.	Adams-Bashforth (AB)	244
7.4.2.	Backward Difference Formulae (BDF)	244
7.4.3.	Das Startup Problem	245
7.5.	Stabilität, Genauigkeit und steife Systeme	245
7.5.1.	Ungedämpfte Systeme	246
7.6.	Schrittweitensteuerung	247
7.6.1.	Schrittweitensteuerung und Mehrschrittverfahren	248
7.6.2.	Dense Output	248

7.6.3. Radau IIA Algorithmus	249
7.6.4. Inline Integration	250
7.7. Unstetigkeiten	250
7.8. Faustregeln bei der Wahl von Solver und Schrittweite	252
7.9. Quantized State Simulation	252
A. Systeme linearer Differentialgleichungen 1. Ordnung mit konstanten Koeffizienten	255
Literaturverzeichnis	260

1. Grundlegendes

Bevor wir in der Lage sind über das Thema „Modellbildung und Simulation Mechatronischer Systeme“ oder etwas detaillierter „Mathematische Modellierung und rechnerunterstützte Simulation (dynamischer) Mechatronischer Systeme“ zu sprechen, bedarf es einiger Definitionen und Erklärungen zu diesem Thema.

1.1. Was ist ein System?

Bei der Definition des Wortes „System“ wird bereits nach kurze Recherche ersichtlich, dass es weiteren Eingrenzungen bedarf, um zu einer einheitlichen Auffassung dieses Begriffs zu gelangen.

Ein Zitat von Prof. F. Cellier aus [Cel91] besagt:

„The largest possible system of all is the universe. Whenever we decide to cut out a piece of the universe such that we can clearly say what is inside that piece (belongs to that piece), and what is outside that piece (does not belong to that piece), we define a new system. A system is characterized by the fact that we can say what belongs to it and what does not, and by the fact that we can specify how it interacts with its environment. System definitions can furthermore be hierarchical. We can take the piece from before, cut out a yet smaller part of it, and we have a new system.“

Bernard Zeigler definiert ein System wie folgt:

„A system is a potential source of data.“

Man sieht also, dass es sehr unterschiedliche Definitionen des Begriffs „System“ gibt. Für Ingenieure kann folgendes festgehalten werden: Ein System ist eine Ansammlung von Elementen dessen Eigenschaften von Interesse sind bzw. welche es zu untersuchen gilt. Dabei kann ein System wiederum aus mehreren kleineren Systemen - sogenannten Sub-Systemen - zusammengesetzt sein. Sobald wir ein System definieren, legen wir im selben Zuge eine sogenannte *Systemgrenze* fest. Dadurch grenzen wir das System von seiner Umgebung ab. Dieser Schritt soll so durchgeführt werden, dass nur der Teil beschrieben wird, der auch von Interesse ist. Das System tauscht über die Systemgrenze Informationen mit der Umgebung aus, welche über *Eingänge* und *Ausgänge* - die eindeutig beschreibbar sein müssen - dargestellt werden können. [Abbildung 1.1](#) veran-

schaulicht diesen Sachverhalt graphisch.

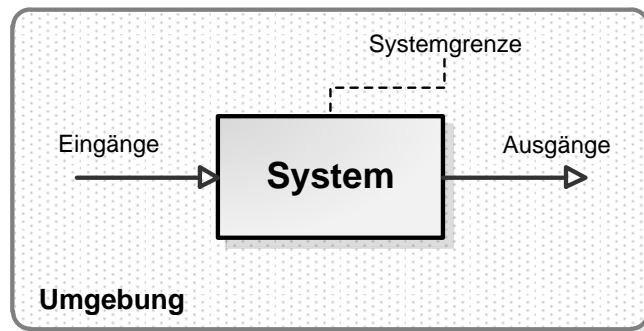


Abb. 1.1.: System welches über Ein- und Ausgänge mit dessen Umgebung interagiert.

Beispiel: Labornetzteil

Wir wollen einen elektrischen Verbraucher mit einer konstanten Spannung von 24V versorgen. Es soll ein System gefunden werden, welches dieser Anforderung gerecht wird. Die erste Ausgangsgröße - eine konstante Spannung von 24V - ist hier schnell gefunden, da diese bereits in der Problemstellung erwähnt wurde. Die zweite Ausgangsgröße ist ein lastabhängiger Strom. Um eine Systemgrenze festlegen zu können, bedarf es aber noch ein- oder mehrerer Eingangsgrößen. Dazu müssen wir uns Gedanken machen, woher die konstante Spannung und der dazugehörige Strom kommen. In anderen Worten, wir müssen uns fragen wie eine konstante Ausgangsspannung erzeugt werden kann. Elektrische Energie wird in Kraftwerken auf verschiedenste Arten erzeugt. Da wir aber nicht das gesamte Versorgungsnetz beschreiben wollen, nur um einen Verbraucher mit 24V Gleichspannung zu versorgen, muss dieses außerhalb unseres Systems liegen. Die vom Kraftwerk produzierte Energie, welche schlussendlich beim Endverbraucher im 230V Versorgungsnetz zur Verfügung steht, soll daher den Eingang in unser System darstellen. Unser System, das es zu beschreiben gilt, muss aus der 230V Wechselspannung am Eingang, eine 24V Gleichspannung am Ausgang erzeugen. Das System „Labornetzteil“ besteht daher im einfachsten Fall aus einem Transformator gefolgt von einer Gleichrichtung mit anschließender Glättung.

1.2. Was ist ein Experiment?

Basierend auf Bernard Zeiglers Definition eines Systems, definiert Prof. F. Cellier ein Experiment wie folgt:

„An experiment is the process of extracting data from a system by exercising it through its inputs.“

Um zu prüfen ob unser unser Labornetzteil bei einer Eingangsspannung von 230V, tatsächlich 24V Gleichspannung am Ausgang liefert, wird ein Prototyp gebaut und getestet. Es wird ein Experiment am System durchgeführt indem man ein Eingangssignal anlegt und beobachtet was am Ausgang passiert. Bei einem Experiment an einem Prototypen gibt es einiges zu beachten: Z.B. ist die Umgebungstemperatur in der Regel nicht konstant. Unser Labornetzteil muss bei niedrigen Temperaturen genauso gut funktionieren wie bei hohen Temperaturen. Unser System muss also bei einer sich verändernden Umgebung trotzdem dasselbe Ergebnis liefern. Genau genommen ist die Systemgrenze unseres Systems „Labornetzteil“ dadurch falsch definiert worden und muss um einen Eingang welcher die Umgebungstemperatur darstellt erweitert werden. Ferner muss das System mit sogenannten Störgrößen zurecht kommen. Unser Versorgungsnetz ist nur theoretisch eine reine Sinusfunktion. In der Realität sind dieser Versorgungsspannung mehrere Störungen überlagert mit dem unser System umgehen muss. Wie bereits bei diesem einfachen Beispiel ersichtlich ist, muss einiges beachtet werden, um sinnvolle Aussagen über ein System treffen zu können.

Weitere Probleme beim Durchführen von Experimenten:

- Experimente sind in der Regel sehr zeitaufwändig und kostenintensiv, da Prototypen entwickelt werden müssen.
- Ein kleiner Fehler im Prototyp kann bis zur vollständigen Zerstörung führen.
- Experimente können sehr gefährlich sein.
- Um das System zu testen muss das entsprechende Equipment vorhanden sein.

1.3. Modellbildung - Warum?

Um solchen Problemen aus dem Weg zu gehen können Aussagen über das Verhalten des Systems getroffen werden noch bevor es entwickelt wurde. Dies gelingt durch das Erstellen mathematischer Modelle welche das reale (physikalische) System möglichst genau beschreiben. Die mathematische Modellbildung hat also zum Einen immer dann ihre Berechtigung wenn, noch bevor das reale System existiert, Aussagen über dasselbe getroffen werden müssen. Dies ist dann wichtig wenn es beispielsweise um Machbarkeitsstudien geht bzw. wenn Abschätzungen getroffen werden müssen noch bevor der Prototyp entwickelt wurde. Zum Anderen werden Modelle oft benötigt, um bestehende Prototypen zu optimieren. Das Modell des bestehenden Systems kann dann modifiziert werden, sodass anhand der Simulationsergebnisse darüber entschieden werden kann, ob das neue System entwickelt wird oder nicht. Des Weiteren ist es bei realen Systemen oft nicht möglich jeden internen Systemzustand zu erfassen, was mit dem Modell ohne

weiteres möglich ist. Ferner können mittels eines Modells alle nur denkbaren Betriebsmodi getestet (simuliert) werden. Alle Probleme die so anhand diverser Experimente am realen System entstehen könnten, werden - wie bereits erwähnt - durch Modellbildung vermieden. All dies führt in der Regel auch zu einem besseren Verständnis der oft komplexen Zusammenhänge von technischen Systemen.

1.4. Mathematische Modelle

Ein mathematisches Modell eines realen Systems erhält man prinzipiell auf zwei verschiedene Arten (siehe [LG94]).

- (1) Durch physikalische Gesetze welche auf Naturgesetzen basieren die über mehrere Generationen bedeutender Wissenschaftlern entstanden sind.
- (2) Durch sogenannte System-Identifikation (experimentelle Identifikation), basierend auf Beobachtungen und Experimenten am realen System.

Beide Prozesse der Modellbildung sind in [Abbildung 1.2](#) veranschaulicht:

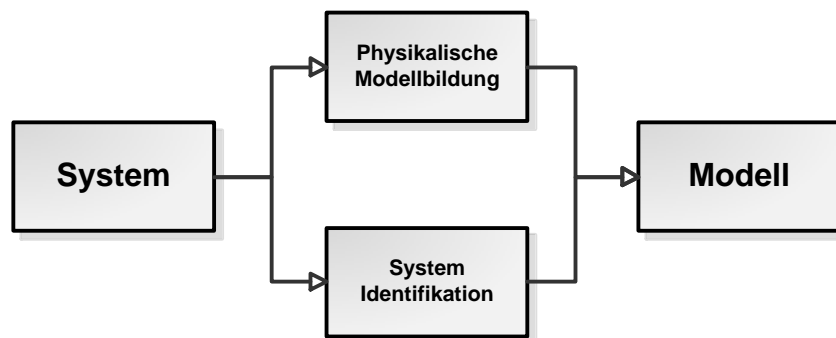


Abb. 1.2.: Modellbildung basierend auf physikalischen Gesetzen (physikalische Modellbildung) oder Beobachtungen und Experimenten (System-Identifikation)

1.4.1. Physikalische Modellbildung: White-Box Modell

Ziel der Modellbildung ist es, ein reales physikalisches System in Form eines mathematischen Modells abzubilden. Dabei soll das Modell das Verhalten des realen physikalischen Systems so gut wie es im jeweiligen Fall nötig ist beschreiben. Sobald das physikalische System mit einem geeigneten mathematischen Modell beschrieben wurde, kann es simuliert werden. Dazu werden im folgenden Skriptum „Modellbildung und

Simulation Mechatronischer Systeme“ mehrere Techniken vorgestellt, welche in den Kapiteln „Klassische Modellbildung: Beschreibung physikalischer Systeme mit DAEs“, „Einführung in die Theorie der Bondgraphen“ und „Objektorientierte Modellierung“ ausführlich behandelt werden und in [Abbildung 1.3](#) dargestellt sind.

Egal wie das zu beschreibende System modelliert wird, das Resultat ist immer ein mathematisches Modell in welchem die Interaktionen der einzelnen Variablen mittels Differentialgleichungen und/oder algebraischen Gleichungen beschrieben wird.

1.4.2. Experimentelle Identifikation: Black-Box Modell

Gelingt es nicht - aus welchen Gründen auch immer - das Modell durch physikalische Gesetze aufzustellen, wird das reale System aufgrund von Beobachtungen und Experimenten identifiziert. Ziel ist es natürlich auch hier das Modell so gut wie möglich an das reale System anzugleichen. Das System wird hier allein durch Messungen identifiziert. Der große Nachteil der experimentellen Identifikation besteht darin, dass das reale System vorhanden sein muss, um anhand diverser Messungen ein Modell zu generieren. Will man Aussagen über ein mögliches Systemverhalten treffen so ist dieser Ansatz unbrauchbar.

1.4.3. Semi-Empirische Modelle

Modelle die teilweise auf physikalischen Modellen und teilweise auf durch Messungen gewonnenen Modellen basieren, werden als semi-empirische Modelle bezeichnet. Diese kommen in der Praxis oft zum Einsatz, da es oft nur schwer möglich ist, physikalische Vorgänge mittels Gleichungen, hinreichend genau zu beschreiben.

1.4.4. Modelle die im Skriptum behandelt werden

Dieses Skriptum befasst sich ausschließlich mit der mathematischen Modellbildung realer Systeme basierend auf physikalischen Gesetzen. Für die Erstellung mathematischer Modelle via System-Identifikation sei u.a. auf [\[LG94\]](#) (Abschnitt III) bzw. [\[Unb00\]](#) (Kapitel 9) verwiesen. Semi-empirische Modelle werden in der Literatur nicht separat betrachtet, sondern sind eher Teil der physikalischen Modellbildung.

1.5. Modelleigenschaften

Es gibt eine Vielzahl von Modelleigenschaften, welche im Folgenden in Kategorien unterteilt werden.

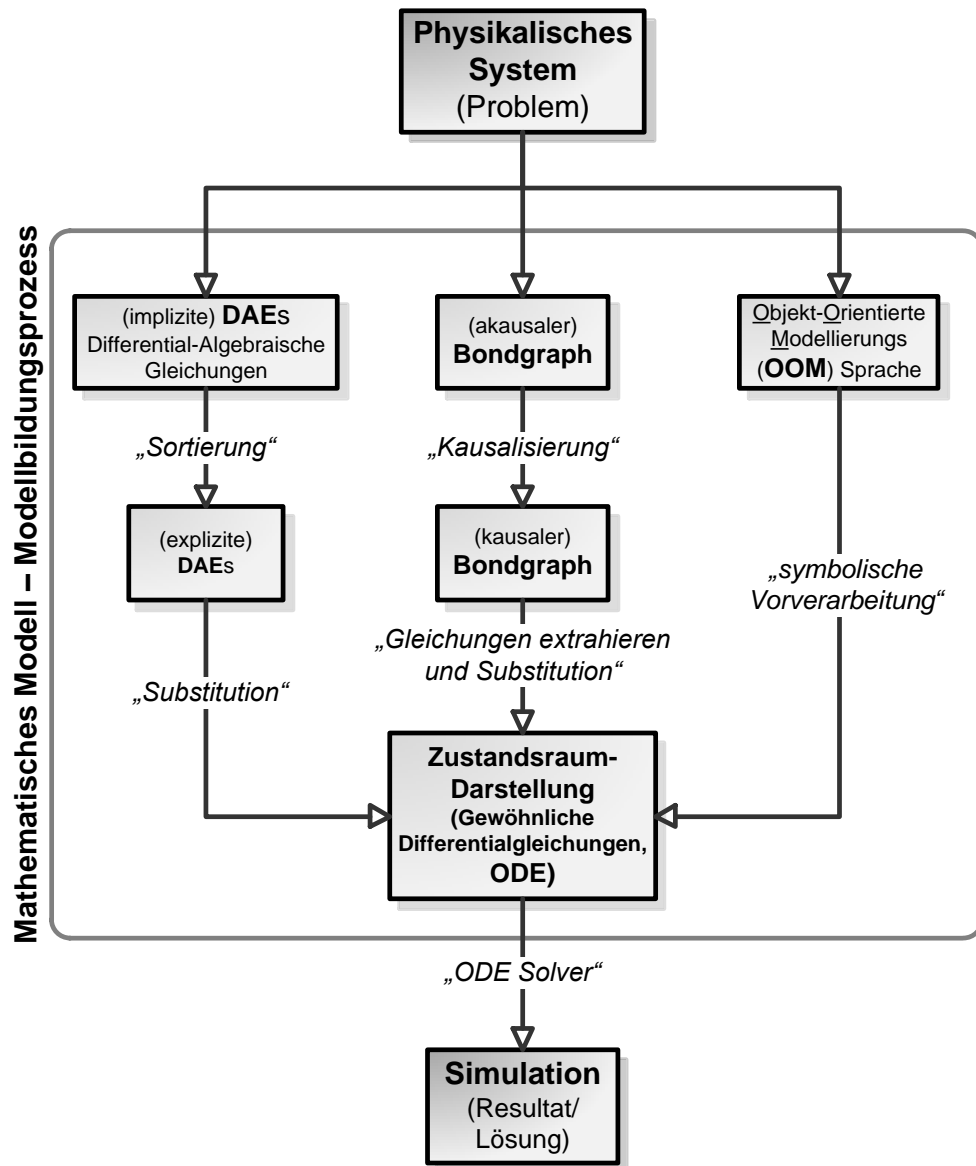


Abb. 1.3.: Wege vom realen physikalischen System zum mathematisch lösbaren Modell welche in diesem Skriptum behandelt werden

1.5.1. Statische und dynamische Modelle

Ein mathematisches Modell eines physikalischen Systems besteht üblicherweise aus einer Anzahl Variablen welche sich in Abhängigkeit der Zeit ändern. Sind die Variablen direkt miteinander verknüpft (proportional zueinander), so handelt es sich um eine statische Beziehung welche in der Mathematik mittels algebraischer Gleichungen beschrieben wird. Ein elektrischer Widerstand welcher der Beziehung

$$u = i \cdot R \tag{1.1}$$

genügt, ist bereits ein einfaches statisches Modell. Bei einer Kapazität hingegen sind die Variablen nicht direkt miteinander verknüpft. Aus diesem Grund wird das (dynamische) Verhalten der Kapazität mittels folgender Differentialgleichung beschrieben:

$$i(t) = C \cdot \frac{du(t)}{dt} \tag{1.2}$$

Löst man diese Differentialgleichung erhält man:

$$u(t) = \frac{1}{C} \int_0^t i(t) \cdot dt + u(0) \tag{1.3}$$

Hier wird ersichtlich, dass die Variablen (Strom $i(t)$ sowie Spannung $u(t)$) einer Kapazität nicht proportional zueinander sind. Durch den Integrator erhält dieses Element ein speicherndes Verhalten. Ferner ist das Verhalten - die Dynamik - der Kapazität abhängig von dessen Vorgeschichte, welche in Form einer sogenannten Anfangsbedingung $u(0)$ definiert wird. In anderen Worten: Wenn die Kapazität in der Vergangenheit auf einen gewissen Wert geladen wurde und diese Ladung nicht verloren gegangen ist, wird dieser gespeicherte Wert in Form der Anfangsbedingung $u(0)$ dargestellt. Die Spannung $u(t)$ ergibt sich also aus einem integrierenden (speichernden) Term und einem Term welcher die Anfangsbedingungen beinhaltet. Die Variable $u(t)$ gibt demnach Auskunft über den aktuellen Zustand unseres Systems. Solche Variablen werden fortan als Zustandsvariablen definiert und in [Kapitel 2](#) ausführlichst behandelt.

1.5.2. Kontinuierliche und diskrete Modelle

Bei einem kontinuierlichen Modell eines Systems besitzen die Modellvariablen zu jedem Zeitpunkt einen Wert. Kontinuierliche Modelle werden daher mit Differentialgleichungen beschrieben. Sobald die Werte von Modellvariablen nur noch zu bestimmten Zeitpunkten vorliegen handelt es sich um diskrete Modelle welche mittels Differenzgleichungen beschrieben werden. Dabei gilt es zu unterscheiden ob die Amplitude oder die Zeit diskretisiert wird. Eine Diskretisierung der Amplitude wird dabei als Quantisierung bezeichnet. Eine Diskretisierung der Zeit hingegen als Abtastung. Sind sowohl

Amplitude als auch Zeit diskretisiert spricht man von digitalen Signalen (siehe [Abbildung 1.4](#)) welche in digitalen Rechner vorliegen da die am Eingang eines Analog-Digital Wandlers anliegenden kontinuierlichen Signale nur zu gewissen äquidistanten Zeitpunkten abgetastet werden und dann auch nur mit einer gewissen Auflösung (z.B. 12Bit) erfasst werden können. Um auch zwischen Abtastintervallen des digitalen Signals einen

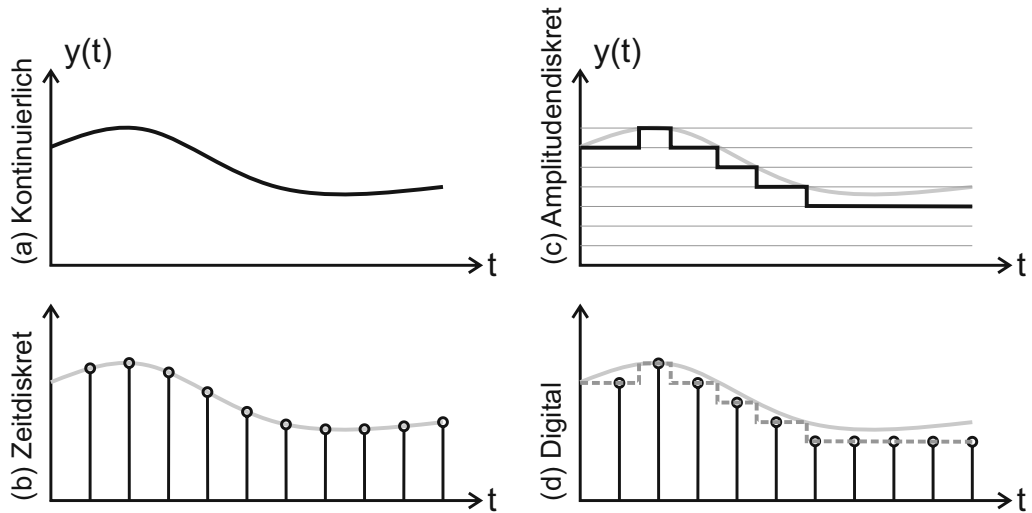


Abb. 1.4.: Kontinuierliche und diskrete Signalformen

Wert zu erhalten, werden sogenannte Sample-and-Hold Glieder eingesetzt (siehe [Abbildung 1.5](#)).

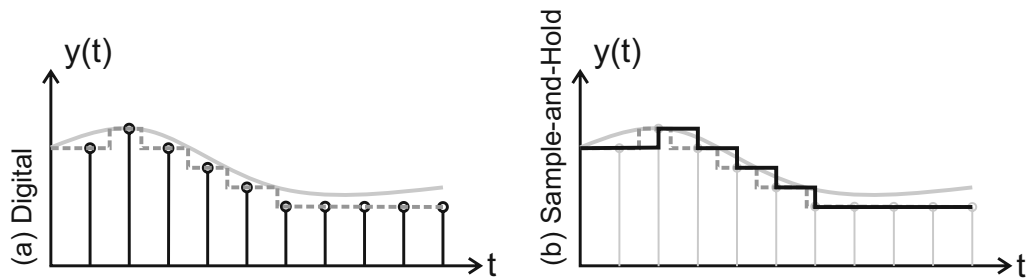


Abb. 1.5.: Digitales Signal welches durch den Einsatz eines Sample-und-Hold Gliedes zu jedem Zeitpunkt einen Wert liefert.

1.5.3. Modelle mit konzentrierten und verteilten Parametern

Sind die Variablen eines Systems nur von der Zeit abhängig, so spricht man von Systemmodellen mit konzentrierten (engl. lumped) Parametern. Solche Modelle werden mit *gewöhnlichen Differentialgleichungen* beschrieben. Sobald Modellvariablen zusätzlich vom Ort abhängen spricht man von Modellen mit verteilten (engl. distributed) Parametern. Diese Modelle werden mit *partiellen Differentialgleichungen* beschrieben. Ein Beispiel für ein Modell dessen Variablen von Raum und Zeit abhängen, ist die Wärmeleitung in einem Leiter. Die eindimensionale Wärmeleitungsgleichung (engl. heat equation) hat folgende Form:

$$\frac{\partial u}{\partial t} = \sigma \cdot \frac{\partial^2 u}{\partial x^2} \quad (1.4)$$

Die Temperatur u ist also nicht nur von der Zeit t abhängig sondern auch vom Ort x . Zum Schluss sei noch kurz erwähnt, dass die Lösung von Modellen mit verteilten Parametern für gewöhnlich durch spezielle Verfahren wie z.B. der sogenannten FEM (engl. Finite Element Method), der MOL (engl. Method of Lines) oder anderen Verfahren (siehe [CK06], Kapitel 6) approximiert wird, da eine analytische Lösung nur in den seltensten Fällen angewendet werden kann. Für unsere Wärmeleitungsgleichung würde eine numerische Lösung der partiellen Differentialgleichung durch eine Diskretisierung im Raum erreicht werden.

$$\left. \frac{\partial^2 u}{\partial x^2} \right|_{x=x_i} \approx \frac{u_{i+1} - 2 \cdot u_i + u_{i-1}}{\delta x^2}$$

Wobei der Index i eine bestimmte x-Koordinate im Raum repräsentiert und δx den Abstand zwischen zwei benachbarten diskreten Punkten (z.B. x_i, x_{i+1}) angibt. Nachdem der Raum diskretisiert wurde, erhalten wir folgenden Satz gewöhnlicher Differentialgleichungen:

$$\frac{\partial u_i}{\partial t} \approx \sigma \cdot \frac{u_{i+1} - 2 \cdot u_i + u_{i-1}}{\delta x^2}$$

1.5.4. Lineare und nichtlineare Modelle

Dieser Abschnitt befasst sich mit linearen und nichtlinearen Modellen. Dazu werden diese zuerst in statische und dynamische Modelle unterteilt. Erstere werden mit algebraischen Gleichungen beschrieben und letztere mit Differentialgleichungen. Das Hauptaugenmerk liegt hier auf der Linearisierung solcher Modelle, da dies in der Praxis des Öfteren von Nöten ist. Folgendes wurde teilweise aus [Unb05] entnommen.

Statische Modelle

Allgemein lässt sich eine statische Funktion (algebraische Gleichung) mittels der Beziehung

$$x(t) = f(u(t)) \quad (1.5)$$

beschreiben. Ein Modell eines Systems ist genau dann linear wenn die Superposition von Gleichung (1.5) für beliebige Linearkombinationen von Eingangsgrößen $u_i(t)$ der Form

$$\sum_{i=1}^n k_i \cdot x_i(t) = f\left(\sum_{i=1}^n k_i \cdot u_i(t)\right) \quad (1.6)$$

gilt. Wobei der Index $i = 1, 2, 3, \dots, n$ ist und k_i dabei konstante Werte darstellt. In anderen Worten: Wird das Modell hintereinander mit n verschiedenen Eingangsgrößen $u_i(t)$ beaufschlagt und jeweils der dazugehörige Ausgang $x_i(t)$ gemessen, so muss die Superposition aller Systemantworten genau dieselbe sein, als würde man die Eingangsgrößen addieren und auf einmal beaufschlagen. [Abbildung 1.6](#) veranschaulicht diesen Sachverhalt graphisch. Wenn dies nicht zutrifft ist das Modell nichtlinear.

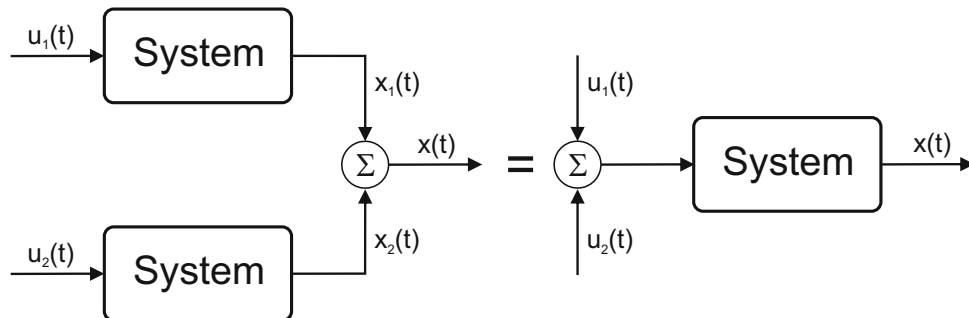


Abb. 1.6.: Superpositionsprinzip welches für lineare Modelle erfüllt sein muss.

Anders als bei linearen Modellen gibt es für die analytische Behandlung nichtlinearer Modelle (z.B. in der Regelungstechnik) weitaus weniger Möglichkeiten. Daher versucht man nichtlineare Modelle - wann immer es möglich ist - zu linearisieren. Im einfachsten Fall besitzt das nichtlineare Modell aus Gleichung (1.5) einen Arbeitspunkt (\bar{u}, \bar{x}) um den es dann linearisiert werden kann (siehe [Abbildung 1.7](#)).

In der Mathematik wird eine Linearisierung durchgeführt in dem die Funktion aus

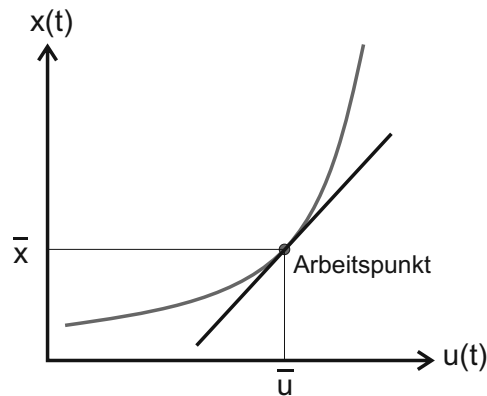


Abb. 1.7.: Linearisierung einer statischen Funktion im Arbeitspunkt

Gleichung (1.5) in eine Taylor-Reihe der Form

$$x = f(\bar{u}) + \underbrace{\frac{df}{du} \Big|_{u=\bar{u}} \cdot (u - \bar{u})}_{\text{linearer Term}} + \underbrace{\frac{1}{2!} \frac{d^2 f}{du^2} \Big|_{u=\bar{u}} \cdot (u - \bar{u})^2 + \dots}_{\text{quadratischer Term}} \quad (1.7)$$

entwickelt wird, wobei die Taylor-Reihe nach dem linearen Term abgebrochen wird, sprich Terme höherer Ordnung eliminiert werden. Wir erhalten

$$x = f(\bar{u}) + \frac{df}{du} \Big|_{u=\bar{u}} \cdot (u - \bar{u}) \quad (1.8)$$

beziehungsweise

$$x = \bar{x} + k \cdot (u - \bar{u}) \quad (1.9)$$

Wobei die Steigung k der Ableitung im Arbeitspunkt $\frac{df}{du} \Big|_{u=\bar{u}}$ entspricht.

Beispiel: Progressive Feder

Diese Linearisierung soll nun anhand eines einfachen Beispiels veranschaulicht werden. Dazu wird eine progressive Feder, welche durch die nichtlineare algebraische Gleichung

$$F(x) = k_1 \cdot x + k_2 \cdot x^3 \quad (1.10)$$

beschrieben wird, um den Ursprung und für große Auslenkungen linearisiert. Die Kennlinie dieser Feder ist in [Abbildung 1.8](#) dargestellt. Wenn eine progressive Feder ihren

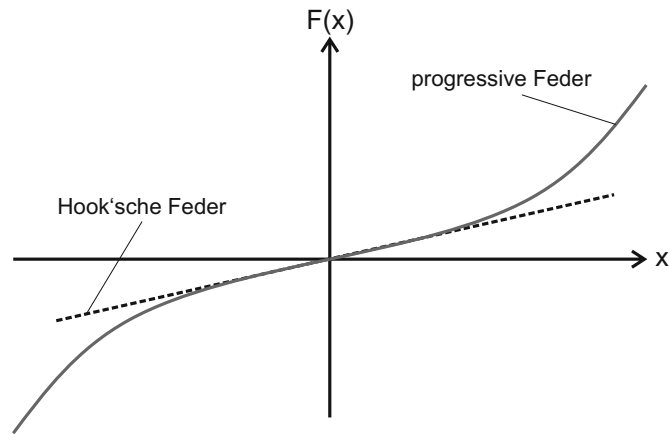


Abb. 1.8.: Kennlinie einer linearen (Hook'schen) Feder und einer progressiven Feder

Arbeitspunkt im Ursprung hat, so ist die Linearisierung denkbar einfach. Nachdem eine Linearisierung meistens nur im Bereich kleiner Auslenkungen (x ändert sich nur minimal) gilt, kann der zweite Term auf der rechten Seite von Gleichung (1.10) vernachlässigt werden. Dieser Schritt führt zur linearen (Hook'schen) Gleichung für die Feder:

$$F(x) = k_1 \cdot x \tag{1.11}$$

Anders ist das bei einer Linearisierung im Bereich größerer Auslenkungen sowie dies in [Abbildung 1.9](#) dargestellt ist. Um hier eine Linearisierung durchführen zu können,

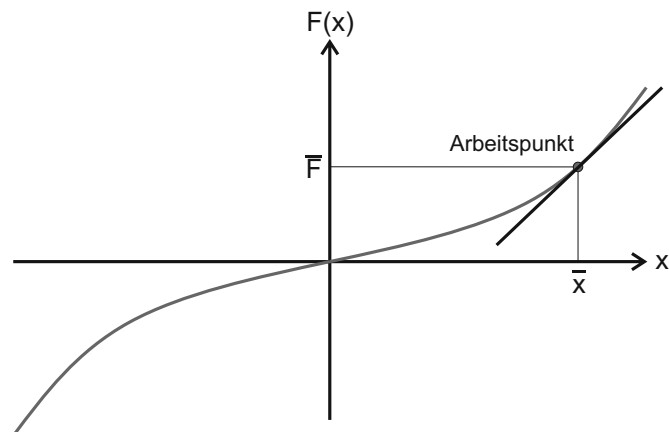


Abb. 1.9.: Linearisierung einer progressiven Feder im Bereich großer Auslenkungen
müssen wir eine Taylor-Reihe um den von Null verschiedenen Arbeitspunkt entwickeln

und wiederum nach dem linearen Term abbrechen. Dies führt zu folgender Gleichung:

$$\begin{aligned}
 F(x) &= \bar{F}(x) + \left. \frac{dF}{dx} \right|_{x=\bar{x}} \cdot (x - \bar{x}) = \bar{F}(x) + \bar{F}' \cdot (x - \bar{x}) \\
 &= k_1 \cdot \bar{x} + k_2 \cdot \bar{x}^3 + (k_1 + 3 \cdot k_2 \cdot \bar{x}^2) \cdot (x - \bar{x}) = \dots \\
 &= \underbrace{(k_1 + 3k_2 \cdot \bar{x}^2)}_k \cdot x - \underbrace{2 \cdot k_2 \cdot \bar{x}^3}_d
 \end{aligned} \tag{1.12}$$

Gleichung (1.12) liegt nun in der bekannten linearen Form ($y = k \cdot x + d$) vor. Das negative Vorzeichen von d wird in [Abbildung 1.9](#) ersichtlich indem die Gerade verlängert wird bis sie die y -Achse schneidet.

Nichtlineare Widerstände

Ein bekanntes Beispiel für ein nichtlineares System ist ein temperaturabhängiger Widerstand. Das Ohm'sche Gesetz $u = i \cdot R$ mit konstantem Widerstand R , ändert sich wie folgt:

$$u = i \cdot R(\vartheta) \tag{1.13}$$

Wobei $R(\vartheta)$ Auskunft darüber gibt, dass der Widerstand von der Temperatur abhängig ist, sprich $R = f(\vartheta)$. Mehr dazu in [Abschnitt 1.6](#).

Dynamische Modelle

Im Vergleich zu Gleichung (1.5) lassen sich dynamische Modelle physikalischer Systeme mittels der Beziehung

$$\dot{x}(t) = f(x(t), u(t)) \tag{1.14}$$

beschreiben. Wobei auch hier wiederum gilt, dass das Modell nur dann linear ist, wenn das Superpositionsprinzip (siehe Gleichung (1.6)) erfüllt ist. Um Gleichung (1.14) zu linearisieren, wird diese wiederum in eine Taylor-Reihe entwickelt, welche nach dem ersten Glied abgebrochen wird.

$$\dot{x}(t) = f(\bar{x}, \bar{u}) + \left. \frac{\partial f}{\partial x} \right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \cdot (x - \bar{x}) + \left. \frac{\partial f}{\partial u} \right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \cdot (u - \bar{u}) \tag{1.15}$$

Im nächsten Schritt müssen wir uns die Frage stellen, wann eine nichtlineare Differentialgleichung überhaupt linearisiert wird. In anderen Worten: Wir müssen einen sinnvollen Arbeitspunkt ermitteln, welcher sich bei einer Differentialgleichung dann einstellt, wenn das System eingeschwungen ist, sich $x(t)$ nicht mehr ändert und damit $\dot{x}(t) = 0$

ist. Wir erhalten:

$$0 = f(\bar{x}, \bar{u}) \quad (1.16)$$

Wobei \bar{x} und \bar{u} wiederum die Werte im Arbeitspunkt (vgl. [Abbildung 1.7](#)) verkörpern. Nun wird folgendes getan: Wenn sich ein dynamisches Modell in dessen Ruhelage befindet (eingeschwungen ist), dann wird es linearisiert indem es etwas um diese Ruhelage ausgelenkt wird, sodass

$$x(t) = \bar{x} + \Delta x(t) \quad (1.17)$$

$$u(t) = \bar{u} + \Delta u(t) \quad (1.18)$$

gilt. Die zeitliche Ableitung von $u(t)$ und $x(t)$ ergibt sodann

$$\dot{x}(t) = \Delta \dot{x}(t) \quad (1.19)$$

$$\dot{u}(t) = \Delta \dot{u}(t) \quad (1.20)$$

Dieser Sachverhalt lässt sich auch etwas anders erklären: Prinzipiell kann man jedes Signal durch einen Gleich- und einen Wechselanteil beschreiben. Sobald das System eingeschwungen ist, befindet es sich im Arbeitspunkt ($u(t) = \bar{u}$, $x(t) = \bar{x}$) was bedeutet, dass die Dynamik jetzt nur noch von den Wechselanteilen $\Delta u(t)$ und $\Delta x(t)$ abhängt.

Um eine lineare Differentialgleichung zu erhalten, betrachtet man also nur noch kleine Auslenkungen ($\Delta x(t)$ und $\Delta u(t)$) um die Ruhelage. Im nächsten Schritt werden die Gleichungen (1.17) bis (1.20) in (1.15) eingesetzt und wir erhalten:

$$\begin{aligned} \Delta \dot{x}(t) &= \underbrace{f(\bar{x}, \bar{u})}_{=0} + \left. \frac{\partial f}{\partial x} \right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \cdot (\bar{x} + \Delta x(t) - \bar{x}) + \left. \frac{\partial f}{\partial u} \right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \cdot (\bar{u} + \Delta u(t) - \bar{u}) \\ &= \left. \frac{\partial f}{\partial x} \right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \cdot \Delta x(t) + \left. \frac{\partial f}{\partial u} \right|_{\substack{x=\bar{x} \\ u=\bar{u}}} \cdot \Delta u(t) \\ &= A \cdot \Delta x(t) + B \cdot \Delta u(t) \end{aligned} \quad (1.21)$$

Obige Gleichung wird als Zustandsgleichung bezeichnet und im nächsten Kapitel „[Einführung in die Zustandsraumdarstellung](#)“ ausführlichst behandelt.

1.5.5. Zeitvariante und zeitinvariante Modelle

Wenn sich die Parameter eines Modells mit der Zeit verändern, also nicht konstant sind, handelt es sich um ein zeitvariantes Modell. Ändern sich die Parameter nicht, handelt es sich um ein sogenanntes zeitinvariantes Modell. Zeitvariante technische Systeme sind solche deren Parameter sich altersbedingt ändern (z.B. Akkus, Halbleiter). Sie sind daher in der Regel recht schwer zu modellieren, da sich die Alterung diverser

Bauteile durch Gleichungen nicht wirklich erfassen lässt. Um solche Aspekte in das Modell miteinfließen zu lassen, müssen daher Messungen über mehrere Jahre gemacht werden.

1.5.6. Deterministische und stochastische Modelle

In der Signalverarbeitung spricht man dann von deterministischen Signalen wenn deren Verlauf vorhersagbar ist (siehe [Mey06]). Anders ist dies bei stochastischen (zufälligen) Signalen dessen Verlauf nicht vorhersagbar ist. Ein bekanntes Beispiel für ein stochastisches Signal ist das Rauschen.

1.5.7. Stabile und instabile Modelle

Der Vollständigkeit halber werden stabile und instabile Modelle noch kurz erwähnt. Ein Modell ist genau dann stabil wenn dessen Ausgänge auf jedes beschränkte Eingangssignal $u(t)$ ein ebenfalls beschränktes Ausgangssignal $y(t)$ zur Folge hat. Dies wird auch als BIBO-Stabilität (engl. Bounded Input, Bounded Output) bezeichnet. [Abbildung 1.10](#) veranschaulicht die Stabilität von Systemen.

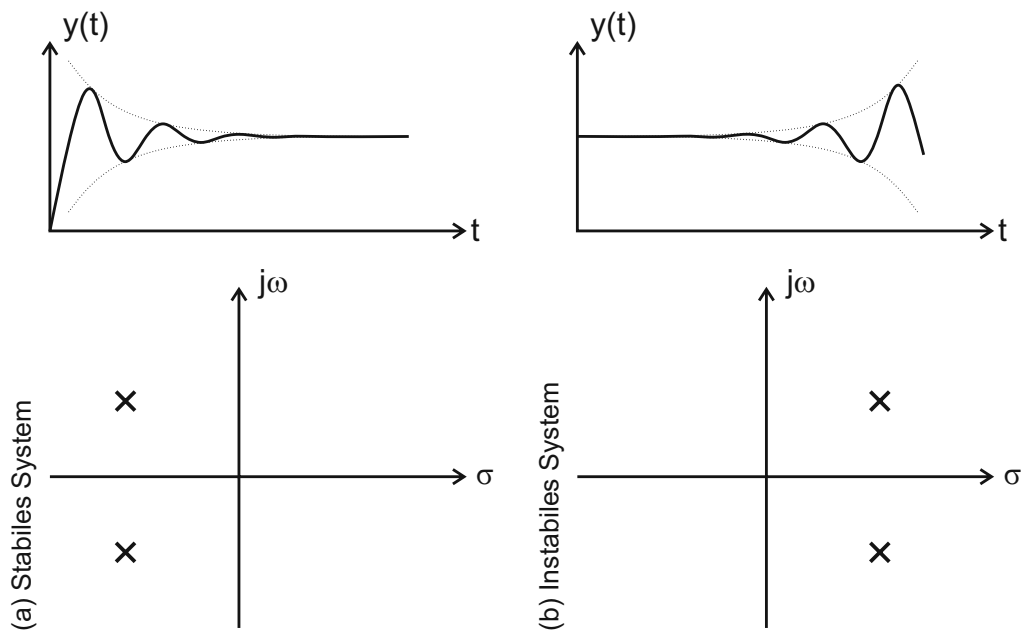


Abb. 1.10.: Stabilitätsverhalten: (a) Sprungantwort und Pole eines stabilen Systems, (b) Sprungantwort und Pole eines instabilen Systems

1.5.8. Modelleigenschaften die im Skriptum behandelt werden

Im vorliegenden Skriptum werden ausschließlich deterministische Modelle mit konzentrierten Parametern behandelt. Die Modelle können dabei sowohl statisch als auch dynamisch sein. Ferner kann es sich bei den Modellen um kontinuierliche sowie um diskrete Modelle handeln, wobei letztere ausschließlich in [Kapitel 6](#) behandelt werden.

1.6. Detaillierungsgrad von Modellen

Wie exakt ein mathematisches Modell das dazugehörige reale physikalische System approximiert hängt vom Detaillierungsgrad des Modells ab. Um das ganze so einfach wie möglich zu halten, soll dieser Sachverhalt anhand der bereits behandelten Gleichungen (1.1) und (1.13) veranschaulicht werden. Im einfachsten Fall wird ein elektrischer Widerstand mittels Gleichung (1.1) beschrieben. Damit wird angenommen, dass der Widerstandswert konstant ist. In der Realität trifft das allerdings nur selten zu. So ändert sich dieser beispielsweise wie in Gleichungen (1.13) beschrieben in Abhängigkeit der Temperatur ϑ . Im einfachsten Fall lässt sich diese Temperaturabhängigkeit mit dem linearen Koeffizienten α und der Temperaturdifferenz ($\Delta\vartheta = \vartheta - \vartheta_0$) darstellen. Die Temperaturabhängigkeit des Widerstands $R(\vartheta)$ wird dann durch die lineare Gleichung

$$R(\vartheta) = R(\vartheta_0) [1 + \alpha \cdot \Delta\vartheta] \quad (1.22)$$

ausgedrückt. Der lineare Koeffizient α hängt vom Material ab und wird meistens bei $\vartheta_0 = 20^\circ\text{C}$ angegeben. Gleichung (1.22) ist für kleine Temperaturbereiche ausreichend genau. Bei größeren Temperaturdifferenzen kann das Verhalten des Widerstands nicht mehr mittels linearer Gleichungen beschrieben werden. Im Prinzip wird nun genau dasselbe gemacht wie in [Unterabschnitt 1.5.4](#) beschrieben mit dem Unterschied, dass die Taylor-Reihe jetzt nicht mehr nach dem linearen Term sondern nach dem quadratischen Term abgebrochen wird. Wir erhalten:

$$R(\vartheta) = R(\vartheta_0) [1 + \alpha \cdot \Delta\vartheta + \beta \cdot \Delta\vartheta^2] \quad (1.23)$$

Wobei der Koeffizient β wiederum vom Material abhängt.

Man erkennt also, dass der nötige Detaillierungsgrad des Modells von der Anwendung abhängt. Für hohen Temperaturen ist das detaillierte Modell aus Gleichung (1.22) nötig. Für tiefe Temperaturen ist es jedoch komplett überflüssig und reduziert sich auf das Modell welches mit Gleichung (1.23) beschrieben wird.

Eine alternative Möglichkeit, um das Temperaturverhalten eines Widerstands zu beschreiben wäre die experimentelle Identifikation der Gleichungen (1.22) bzw. (1.23). Dazu wird die Umgebungstemperatur des Widerstands schrittweise erhöht und der

dazugehörige Widerstandswert gemessen. So entsteht eine Tabelle welche den Widerstand in Abhängigkeit der Temperatur beschreibt. Wird nun eine sogenannte Regression durchgeführt (z.B. mit Excel) so entsteht je nach Art der Regression - linear, quadratisch, etc - eine Gleichung in der die Temperaturkoeffizienten explizit als Wert dargestellt werden.

1.7. Simulation

Eine Simulation ist nichts anderes als ein Experiment, welches am mathematischen Modell durchgeführt wird. Eine Simulation wird mit Hilfe eines Computer Programms durchgeführt und ist eine numerische Approximation der analytischen Lösung des Modells. Das Thema Simulation wird in [Kapitel 7](#) erläutert. Dabei werden Eingänge vorgegeben und das Verhalten der Ausgänge beobachtet.

2. Einführung in die Zustandsraumdarstellung

Wenn von Modellbildung und Simulation gesprochen wird, sind damit immer zwei grundsätzlich unterschiedliche Prozesse gemeint: Der Prozess „Modellbildung“ ist eine klassische Ingenieursdisziplin, welche im Verlauf eines technischen Studiums einen Teil der Ausbildung darstellen sollte. Dabei wird das physikalische System durch ein mathematisches Modell beschrieben, welches dann dem Solver übergeben wird. Dazu muss das mathematische Modell bestimmte Voraussetzungen erfüllen, damit dieses von einem Solver korrekt interpretiert und simuliert werden kann. Der Prozess „Simulation“ hingegen fällt weitgehend in die Bereiche Mathematik und Informatik. Dabei werden, je nach Problemstellung, verschiedene Verfahren entwickelt, welche die numerische Lösung so gut wie möglich an die analytische Lösung approximieren. Auch wenn dies nicht zu den Aufgaben von Ingenieuren zählt, ist es jedoch hilfreich, wenn sie zumindest in groben Zügen verstehen, was es heißt, eine Lösung numerisch zu approximieren, und welcher der zur Verfügung stehenden Solver für das zu simulierende Modell am Besten geeignet ist.

Wie bereits erwähnt, muss ein Modell in einer bestimmten wohldefinierten Form vorliegen, um vom Solver korrekt interpretiert zu werden. Dabei hat sich eine Form durchgesetzt: die sogenannte Zustandsraumdarstellung (engl. state-space representation). Man kann auch sagen, dass durch die Zustandsraumdarstellung eine klare Trennung der Prozesse „Modellbildung“ und „Simulation“ geschaffen wird.

In der Mathematik spricht man üblicherweise nicht von einer Zustandsraumdarstellung. Vielmehr wird hier von *Systemen linearer gewöhnlicher Differentialgleichungen 1. Ordnung* (engl. ordinary differential equations, kurz ODE) gesprochen. Bringt man solche Systeme in eine Matrizendarstellung, so erhält man eine wesentlich kompaktere Darstellung desselben. Dies wird in Anhang A beschrieben. Für eine detaillierte, wirklich ausgezeichnete Einführung, in das Thema Differentialgleichungen sei auf [Pap01] verwiesen. In den Ingenieurwissenschaften wird diese (kompakte) Matrizengleichung als Zustandsdifferentialgleichung oder kurz Zustandsgleichung bezeichnet. Erweitert man diese noch um eine weitere Gleichung, der sogenannten Ausgangsgleichung, erhält man die Zustandsraumdarstellung physikalischer Systeme.

Die Zustandsraumdarstellung wird aber nicht nur als Input für Solver verwendet. Sie hat vor allem in der Regelungstechnik größte Bedeutung. Eine detaillierte, sehr gute Einführung in die Zustandsraumdarstellung ist in [Föl08] und [Unb00] zu finden.

2.1. Zustandsvariablen

Technische Systeme werden prinzipiell durch drei verschiedene Größen charakterisiert (Abbildung 2.1):

- Eingangsgrößen $\underline{u}(t)$
- Ausgangsgrößen $\underline{y}(t)$
- Zustandsgrößen $\underline{x}(t)$

Zustandsvariablen (oder Zustandsgrößen) wurden bereits in [Unterabschnitt 1.5.1](#) eingeführt und sollen im Folgenden noch detaillierter beschrieben werden.

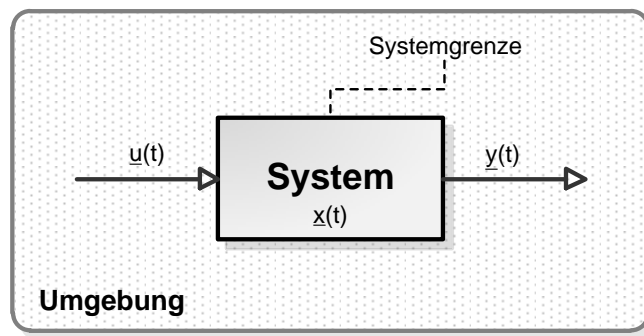


Abb. 2.1.: Charakteristische Größen technischer Systeme

Allgemein gilt: Zustandsgrößen sind physikalische Größen, die den aktuellen Status (Ladung eines Kondensators, magnetischer Fluss einer Induktivität) von energiespeichernden Elementen beschreiben. Aus der Kenntnis des Zustands zu einem gegebenen Zeitpunkt kann unter Berücksichtigung äußerer Einflussgrößen (Eingänge) das Verhalten des Systems für alle folgenden Zeitpunkte berechnet werden. Gleichungen, welche Energiespeicher beschreiben, enthalten immer eine zeitliche Ableitung, entweder in integraler oder differentieller Form. Man kann also jeden Energiespeicher über ein Integral beschreiben¹. In anderen Worten: Zustandsgrößen sind die Ausgangsgrößen von Integratoren. Wie aus der Mathematik bekannt ist, hat jeder Integrator eine Anfangsbedingung. Dies entspricht in der Zustandsraumdarstellung der „Beschreibung der Vergangenheit“, welche vor dem Start der Simulation liegt und allgemein mit \underline{x}_0 bezeichnet wird. Aus diesen Anfangsbedingungen und den Modellgleichungen können die Zustände somit für jeden Zeitpunkt berechnet werden.

¹Integratoren werden gegenüber Differentiatoren bevorzugt eingesetzt, weil viele gängige Lösungsverfahren (Solver) auf diese Form aufbauen.

2.2. Ein einführendes Beispiel

Im Folgenden wird die Zustandsraumdarstellung anhand eines einführenden Beispiels erläutert. Zuerst wird die Wahl der Eingangs- bzw. Ausgangsgrößen durchgeführt. Anschließend wird die Wahl der Zustandsgrößen erläutert. Ferner werden die Zustands- und die Ausgangsgleichung der Zustandsraumdarstellung interpretiert und deren Eigenschaften erläutert.

Dazu wird erneut eine elektrische Schaltung herangezogen, welche in [Abbildung 3.1](#) dargestellt ist.

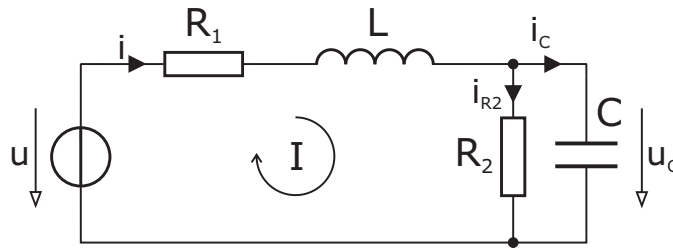


Abb. 2.2.: Passive elektronische Schaltung

2.2.1. Bestimmen der Ein- und der Ausgangsgrößen

Die Eingangsgröße ist in unserem Beispiel die Eingangsspannung.

$$u(t) = u \quad (2.1)$$

Die Ausgangsgröße ist frei wählbar. Die Ausgangsgrößen werden nach Art der Anwendung festgelegt. In unserem Beispiel sind die Ausgangsgrößen frei wählbar, je nachdem welche Größen von Interesse sind. In der Regelungstechnik müssen dementsprechend die zu regelnden Größen gewählt werden.

$$y(t) = u_C \quad (2.2)$$

2.2.2. Bestimmen der Zustandsgrößen

Die elektrische Schaltung besitzt zwei energiespeichernde Elemente und wird daher als *System 2. Ordnung* charakterisiert. Allgemein gilt: Die Ordnung eines Systems wird durch die Anzahl Zustandsvariablen, für welche die Anfangsbedingungen unabhängig voneinander bestimmt werden, definiert und entspricht der Anzahl der energiespeichernden Elemente. Wir benötigen also zwei Differentialgleichungen *1. Ordnung*, um

das System zu beschreiben.

$$u_C(t) = \frac{1}{C} \int_0^t i_C \cdot dt + u_C(0) \equiv x_1(t) \quad (2.3)$$

$$i(t) = \frac{1}{L} \int_0^t u_L(t) \cdot dt + i(0) \equiv x_2(t) \quad (2.4)$$

Die Zustandsgrößen

$$\underline{x}(t) = \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$$

der vorliegenden elektrischen Schaltung sind die Spannung $u_C(t)$ und der Strom $i(t)$. Die dazugehörigen Anfangsbedingungen werden mit $u_C(0)$ und $i(0)$ bezeichnet und künftig durch den Spaltenvektor \underline{x}_0 dargestellt.

Wir halten Folgendes fest: Wenn wir ein *System n-ter Ordnung* im Zustandsraum darstellen wollen, dann müssen wir dieses *System n-ter Ordnung* mittels n *Differentialgleichungen 1. Ordnung* beschreiben.

2.2.3. Aufstellen der Zustandsgleichungen

Ziel ist es nun, die zeitliche Änderung der Zustandsvariablen $x_1(t)$ und $x_2(t)$ zu beschreiben. Dazu werden die Gleichungen (2.3) und (2.4) nach der Zeit abgeleitet und lediglich durch bekannte Größen, sprich durch die Eingangsgröße $u(t)$ und die beiden Zustandsgrößen $x_1(t)$ und $x_2(t)$ selbst ausgedrückt.

Aufstellen der ersten Zustandsgleichung

Um die erste der beiden Zustandsgleichungen (\dot{x}_1) zu erhalten, bilden wir die zeitliche Ableitung von Gleichung (2.3):

$$\dot{x}_1 = \frac{1}{C} \cdot i_C \quad (2.5)$$

Um Gleichung (2.5) nun durch bekannte Größen ausdrücken zu können, wird von der Kirchhoff'schen Knotenregel Gebrauch gemacht.

$$\begin{aligned} i &= i_{R_2} + i_C \\ \Rightarrow i_C &= i - i_{R_2} \end{aligned} \quad (2.6)$$

Gleichung (2.6) wird nun in Gleichung (2.5) eingesetzt und die Ströme werden durch deren Komponentengleichungen substituiert.

$$\begin{aligned}\dot{x}_1 &= \frac{1}{C}(i - i_{R_2}) \\ \dot{x}_1 &= \frac{1}{C} \left(x_2 - \frac{1}{R_2} \cdot u_{R_2} \right)\end{aligned}\quad (2.7)$$

Da der Widerstand R_2 parallel zum Kondensator C ist, besitzen beide Komponenten dieselbe Spannung und wir erhalten:

$$\begin{aligned}\dot{x}_1 &= \frac{1}{C} \left(x_2 - \frac{1}{R_2} \cdot x_1 \right) \\ \dot{x}_1 &= -\frac{1}{C \cdot R_2} \cdot x_1 + \frac{1}{C} \cdot x_2\end{aligned}\quad (2.8)$$

Aufstellen der zweiten Zustandsgleichung

Dazu wird Gleichung (2.4) abgeleitet:

$$\dot{x}_2 = \frac{1}{L} \cdot u_L \quad (2.9)$$

Im nächsten Schritt wird u_L durch bekannte Größen ausgedrückt. Dazu wird Masche I herangezogen.

$$\begin{aligned}u &= u_{R_1} + u_L + u_C \\ u_L &= u - u_{R_1} - u_C\end{aligned}\quad (2.10)$$

Gleichung (2.10) wird nun in Gleichung (2.9) eingesetzt und die Spannungen werden wiederum durch die Eingangsgröße und die Zustandsgrößen ausgedrückt.

$$\begin{aligned}\dot{x}_2 &= \frac{1}{L} (u - u_{R_1} - u_C) \\ \dot{x}_2 &= \frac{1}{L} (u(t) - R_1 \cdot i - x_1) \\ \dot{x}_2 &= \frac{1}{L} (u(t) - R_1 \cdot x_2 - x_1) \\ \dot{x}_2 &= -\frac{1}{L} \cdot x_1 - \frac{R_1}{L} \cdot x_2 + \frac{1}{L} \cdot u(t)\end{aligned}\quad (2.11)$$

Nun ist es uns gelungen, die zeitliche Änderung der Zustandsvariablen durch die Eingangsgröße $u(t)$ und die Zustandsvariablen selbst auszudrücken. Nachdem es sich bei all diesen Größen um bekannte Größen handelt, sind auch die zeitlichen Ableitungen,

welche die Änderung der Zustandsvariablen beschreiben, bekannt. Mit Hilfe von Zustandsgleichungen sind wir daher in der Lage, das Verhalten unseres Systems eindeutig zu beschreiben. Eine detaillierte Interpretation der Zustandsgleichung erfolgt in [Unterabschnitt 2.3.3](#).

2.2.4. Bilden der Zustandsraumdarstellung

Nun werden die Gleichungen (2.8) und (2.11) in eine Vektor-Matrix Form übergeführt.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{1}{C \cdot R_2} & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R_1}{L} \end{pmatrix}}_A \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ \frac{1}{L} \end{pmatrix}}_b \cdot u(t) \quad (2.12)$$

Diese Gleichung wird fortan als Zustandsgleichung bezeichnet. Der Vollständigkeit halber, werden die Matrix \underline{A} und die Vektoren \underline{b} und \underline{c}^T sowie die Anfangsbedingungen \underline{x}_0 separat angeführt und wir erhalten:

Die Systemmatrix,

$$\underline{A} = \begin{pmatrix} -\frac{1}{C \cdot R_2} & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R_1}{L} \end{pmatrix}, \quad (2.13)$$

die Eingangs- oder Steuermatrix

$$\underline{b} = \begin{pmatrix} 0 \\ \frac{1}{L} \end{pmatrix} \quad (2.14)$$

und die Anfangsbedingungen

$$\underline{x}_0 = \begin{pmatrix} x_1(0) \\ x_2(0) \end{pmatrix} = \begin{pmatrix} u_C(0) \\ i(0) \end{pmatrix}. \quad (2.15)$$

Um zur Zustandsraumdarstellung zu gelangen, muss noch ein weiterer Schritt durchgeführt werden. Wir müssen noch eine Verbindung der Zustandsvariablen $\underline{x}(t)$ und der Ausgangsvariablen $\underline{y}(t)$ herstellen. Diese wird über eine weitere Gleichung, die sogenannte Ausgangsgleichung, hergestellt. Im vorliegenden Beispiel ist, wie bereits in Gleichung (2.2) beschrieben, die Ausgangsgröße gleich der Spannung u_C am Kondensator. Diese Spannung ist gleichzeitig die Zustandsvariable x_1 . Wir erhalten:

$$\underline{y} = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{\underline{c}^T} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (2.16)$$

Wobei \underline{c}^T als die Ausgangs- oder Beobachtungsmatrix bezeichnet wird.

Für den Fall, dass die Ausgänge (Ausgangsgrößen) Zustandsvariablen entsprechen, wird im Zeilenvektor \underline{c}^T an der entsprechenden Stelle eine Eins eingetragen. Ist der gewünschte Ausgang keine Zustandsvariable, muss er aus diesen errechnet werden.

2.3. Zustandsraumdarstellung

Die Zustandsraumdarstellung beschreibt, wie bereits erwähnt, ein System *n-ter Ordnung* mittels *n* Differentialgleichungen *1. Ordnung*. Die allgemeine Form der Zustandsraumdarstellung für Systeme bestehend aus *n* Differentialgleichungen 1. Ordnung hat folgende Form:

$$\dot{\underline{x}}(t) = \underline{f}(\underline{x}, \underline{u}, t) \quad (2.17)$$

$$\underline{y}(t) = \underline{g}(\underline{x}, \underline{u}, t) \quad (2.18)$$

Wobei \underline{f} und \underline{g} lineare oder nicht-lineare vektorielle Funktionen darstellen. Im Falle linearer zeitinvarianter Systeme (engl. linear time invariant, kurz LTI-Systeme), kann die vereinfachte in [Abschnitt 1.5.4](#) hergeleitete Form der Zustandsraumdarstellung herangezogen werden.

Im Folgenden behandeln wir die lineare Zustandsraumdarstellung für Eingrößensysteme, sogenannte SISO-Systeme (engl. Single-Input, Single-Output) und für Mehrgrößensysteme, sogenannte MIMO-Systeme (engl. Multiple-Input Multiple-Output).

2.3.1. Eingrößensysteme (SISO-Systeme)

In unserem einführenden Beispiel haben wir ein lineares Eingrößensystem im Zustandsraum dargestellt. Die Zustandsraumdarstellung für dieses Eingrößensystem lautet:

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{b} \cdot u(t), \quad \underline{x}(0) = \underline{x}_0 \quad (2.19)$$

$$\underline{y}(t) = \underline{c}^T \cdot \underline{x}(t) \quad (2.20)$$

Wobei $\underline{x}(t)$ ein $(n \times 1)$ Zustandsvektor, $y(t)$ der Ausgang, \underline{A} die sogenannte Systemmatrix mit einer Dimension von $(n \times n)$, \underline{b} ein $(n \times 1)$ Eingangsvektor und \underline{c}^T ein $(1 \times n)$ Ausgangsvektor ist.

Der Zustandsvektor $\underline{x}(t)$ zum Zeitpunkt $t = 0$ beinhaltet die Anfangsbedingungen $\underline{x}(0)$ welche in [Unterabschnitt 2.2.4](#) bereits mit \underline{x}_0 bezeichnet wurden.

[Abbildung 2.3](#) zeigt das Blockschaltbild der Zustandsraumdarstellung. Um eine allgemein gültige Form zu erhalten, fehlt allerdings noch ein Konstrukt welches die „Sprungfähigkeit“ eines Systems beschreibt. Dieses Konstrukt lässt sich aber sehr einfach über

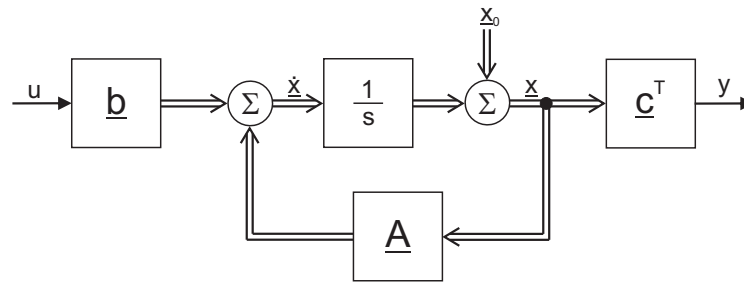


Abb. 2.3.: Blockschaltbild der Zustandsraumdarstellung für SISO-Systeme.

einen zusätzlichen Term in der Ausgangsgleichung bewerkstelligen. Wir erhalten:

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{b} \cdot u(t), \quad \underline{x}(0) = \underline{x}_0 \quad (2.21)$$

$$y(t) = \underline{c}^T \cdot \underline{x}(t) + d \cdot u(t) \quad (2.22)$$

Wobei d allgemein als Durchgangsmatrix bezeichnet wird und im Falle von Eingrößensystemen ein Skalar ist. [Abbildung 2.4](#) zeigt das vollständige Blockschaltbild der Zustandsraumdarstellung.

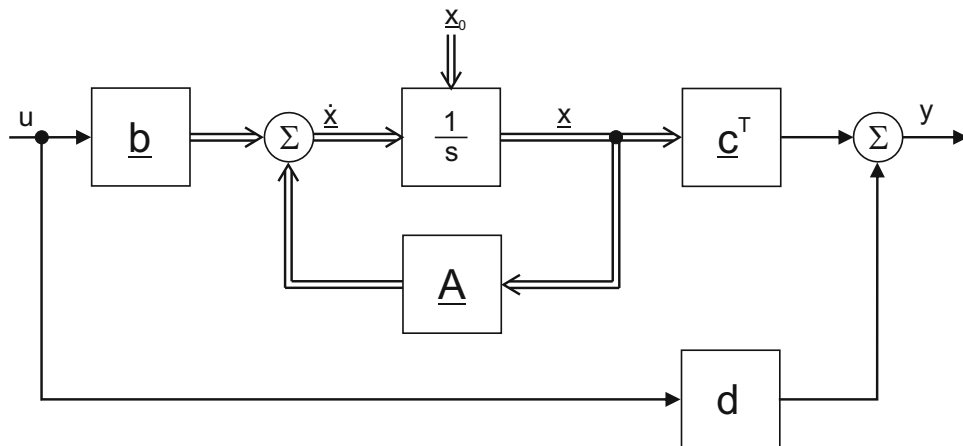


Abb. 2.4.: Blockschaltbild der Zustandsraumdarstellung für SISO-Systeme. Zusätzlich wurden die Anfangsbedingungen \underline{x}_0 als Eingang in den Integrator dargestellt, was darauf hindeuten soll, dass diese auch dort spezifiziert werden.

Anhand des Blockschaltbildes in [Abbildung 2.4](#) ist sehr gut ersichtlich, dass sich ein Sprung am Eingang über die Durchgangsmatrix d proportional auf den Ausgang auswirkt.

2.3.2. Mehrgrößensysteme (MIMO-Systeme)

Bisher haben wir den Zustandsraum für SISO-Systeme behandelt. Handelt es sich jedoch um MIMO-Systeme dann werden aus den Vektoren \underline{b} und \underline{c}^T sowie dem Skalar d die Matrizen \underline{B} , \underline{C} und \underline{D} und wir erhalten folgende allgemeingültige Darstellung:

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t), \quad \underline{x}(0) = \underline{x}_0 \quad (2.23)$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) + \underline{D} \cdot \underline{u}(t) \quad (2.24)$$

Wobei $\underline{x}(t)$ ein $(n \times 1)$ Zustandsvektor, \underline{y} ein $(m \times 1)$ Ausgangsvektor, \underline{A} die sogenannte Systemmatrix mit einer Dimension von $(n \times n)$, \underline{B} eine $(n \times r)$ Eingangsmatrix, \underline{C} eine $(m \times n)$ Ausgangsmatrix und \underline{D} die sogenannte Durchgangsmatrix mit einer Dimension von $(m \times r)$ ist.

Das dazugehörige Blockschaltbild ist in [Abbildung 2.5](#) dargestellt. Wie bereits erwähnt,

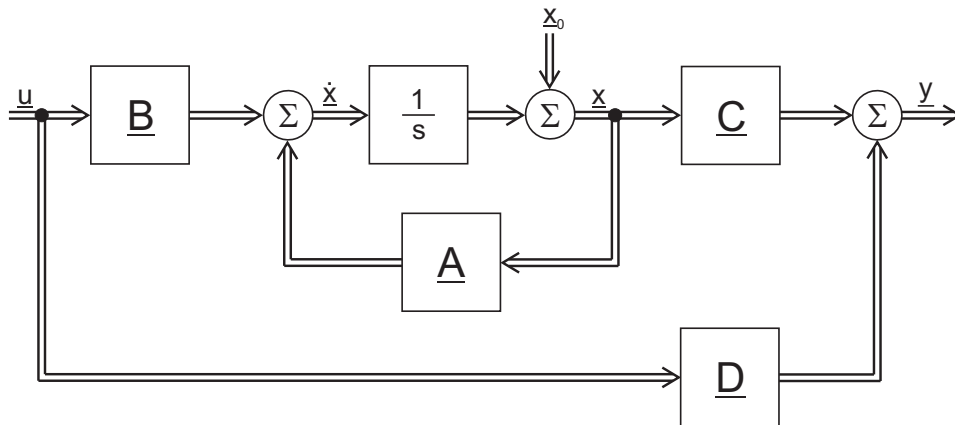


Abb. 2.5.: Blockschaltbild der Zustandsraumdarstellung für MIMO-Systeme

sind die Einträge der \underline{D} Matrix in den meisten Fällen gleich Null, außer der Ausgang hängt (zumindest teilweise) direkt proportional vom Eingang ab. Handelt es sich also um nicht-sprungfähige Systeme, reduziert sich die Zustandsraumdarstellung auf folgende Gleichungen:

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t), \quad \underline{x}(0) = \underline{x}_0 \quad (2.25)$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) \quad (2.26)$$

Das zugehörige Blockschaltbild der reduzierten Zustandsraumdarstellung ist in [Abbildung 2.6](#) dargestellt.

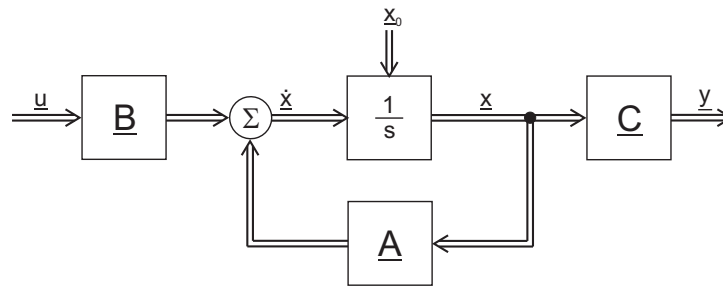


Abb. 2.6.: Reduziertes Blockschaltbild der Zustandsraumdarstellung.

2.3.3. Interpretation der Zustandsgleichung

Im Folgenden wird die Zustandsdifferentialgleichung, kurz Zustandsgleichung (siehe Gleichung (2.23)), welche die erste der beiden Gleichungen der Zustandsraumdarstellung verkörpert, etwas genauer interpretiert. Die Zustandsgleichung beschreibt die Dynamik eines physikalischen Systems. Dadurch, dass sich die zeitliche Änderung der Zustandsvariablen durch Eingangsgrößen und Zustandsvariablen selbst beschreiben lässt, ist diese zu jedem Zeitpunkt bekannt. Diese zeitliche Änderung $\dot{\underline{x}}(t)$ der Zustandsvariablen innerhalb eines bestimmten Zeitraums $[t_{start} \leq t \leq t_{end}]$ kann graphisch als Trajektorie dargestellt werden (siehe [Abbildung 2.7](#)).

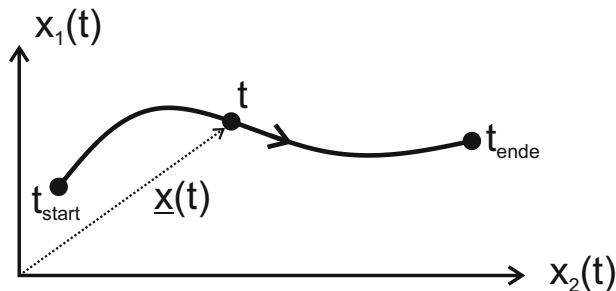


Abb. 2.7.: Graphische veranschaulichung der Trajektorie eines Systems bestehend aus zwei Zustandsvariablen

Der aktuelle Zustandsvektor $\underline{x}(t)$ ergibt sich also immer aus dem vorherigen Zustandsvektor \underline{x}_{alt} und den Zustandsänderungen $\dot{\underline{x}}(t)$ in einem infinitesimal kleinen Zeitabschnitt dt

$$\underline{x}(t) = \underline{x}_{aktuell} = \underline{x}_{alt} + \dot{\underline{x}}(t) \cdot dt \quad (2.27)$$

oder in der aus der Mathematik eher üblichen Formulierung für die Beschreibung infi-

infinitesimal kleiner Änderungen (vgl. Definition des Differentials):

$$\dot{\underline{x}}(t) = \frac{d\underline{x}(t)}{dt} = \frac{\underline{x}_{\text{aktuell}} - \underline{x}_{\text{alt}}}{dt} \quad (2.28)$$

Auf diese Weise werden Modelle auf einem digitalen Rechner (Computer) mittels expliziter Solver simuliert. Dabei lässt sich die Änderung der Zustandsvariablen allerdings nicht mit infinitesimal kleinen Schritten beschreiben, da das Ergebnis auf einem digitalen Rechner nur zu bestimmten äquidistanten Zeitpunkten, welche als Schrittweite h bezeichnet werden, berechnet wird. Ein Solver berechnet daher den neuen Zustandsvektor $\underline{x}(t+h)$ wie folgt:

$$\underline{x}(t+h) = \underline{x}(t) + \dot{\underline{x}}(t) \cdot h \quad (2.29)$$

Diese Gleichung beschreibt eine Taylor-Reihe, welche nach dem *linearen Term*² abgebrochen wird. Setzt man nun für $\dot{\underline{x}}(t)$ die lineare Zustandsgleichung ein erhält man schlussendlich die diskrete Form der Zustandsgleichung.

$$\begin{aligned} \underline{x}(t+h) &= \underline{x}(t) + (\underline{A} \cdot \underline{x}(t) + \underline{b} \cdot u(t)) \cdot h = \\ &= \underline{x}(t) + \underline{A} \cdot h \cdot \underline{x}(t) + \underline{b} \cdot h \cdot u(t) = \\ &= (\mathbb{1} + \underline{A} \cdot h) \cdot \underline{x}(t) + \underline{b} \cdot h \cdot u(t) \end{aligned} \quad (2.30)$$

Eine detaillierte Einführung in das Thema Solver folgt in [Kapitel 7](#).

Aus der Theorie der gewöhnlichen Differentialgleichungen ist bekannt, dass eine Differentialgleichung durch einen homogenen und einen partikulären Anteil beschrieben werden kann. Ist das physikalische System nicht fremderregt, wird dem System also keine erzwungene Bewegung erteilt, so besitzt die dazugehörige Differentialgleichung auch keinen partikulären Anteil. D.h., dass $u(t) = 0$ ist, und wir daher folgende reduzierte Zustandsgleichung erhalten:

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t), \quad \underline{x}(0) = \underline{x}_0 \quad (2.31)$$

Diese Gleichung beschreibt damit die Eigendynamik (Eigenverhalten) des Systems. Systeme, welche nicht fremderregt sind, werden auch als autonome Systeme bezeichnet. Die vollständige Information über das Eigenverhalten steckt also in der Systemmatrix \underline{A} . Berechnet man nun die Eigenwerte $\underline{\lambda}$ (Pole) dieser Matrix, so erhält man die vollständige Information über die Dynamik des Systems. Um die Eigenwerte zu erhalten, muss folgende Gleichung gelöst werden:

$$\det(\underline{A} - \underline{\lambda} \cdot \mathbb{1}) = 0 \quad (2.32)$$

²Verfahren, die eine Lösung approximieren, indem sie die Funktion in eine Taylor-Reihe entwickeln, welche nach dem linearen Term abgebrochen wird, werden als Solver 1. Ordnung bezeichnet (engl. 1st order solver).

Sobald diese bekannt sind, sind auch die Stabilitätseigenschaften des Systems bekannt. Diese wurde bereits in [Unterabschnitt 1.5.7](#) ([Abbildung 1.10](#) veranschaulicht. Befinden sich die Pole in der linken Halbebene der s-Ebene, dann ist das System stabil.

2.3.4. Interpretation der Ausgangsgleichung

Gleichung (2.24) der Zustandsraumdarstellung beschreibt den Zusammenhang zwischen Eingangsgrößen und Zustandsvariablen zu den Ausgangsgrößen. Ist eine Zustandsvariable gleichzeitig ein Ausgang des Systems, so wird eine Eins an die entsprechende Stelle der Ausgangsmatrix \underline{C} geschrieben. Dies wurde bereits im vorherigen Beispiel demonstriert. Gelegentlich kommt es vor, dass die Ausgangsgröße keine Zustandsvariable ist. Folglich ist der Eintrag an der entsprechenden Stelle in der Ausgangsmatrix \underline{C} ungleich Eins.

Angenommen uns interessiert der Strom i_{R_2} durch den Widerstand R_2 als Ausgangsgröße anstelle der Spannung u_C des Kondensators C .

$$y(t) = i_{R_2} \tag{2.33}$$

Dieser Strom ist allerdings keine Zustandsvariable. Um dennoch eine Verknüpfung mit dem Ausgang $y(t)$ herzustellen, muss dieser mittels Zustandsgrößen ausgedrückt werden:

$$i_{R_2} = \frac{1}{R_2} \cdot u_C = \frac{1}{R_2} \cdot x_1 \tag{2.34}$$

Daraus folgt unmittelbar die neue Ausgangsgleichung:

$$y(t) = \left(\frac{1}{R_2} \quad 0 \right) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{2.35}$$

Zusätzlich kann auch noch ein möglicher, direkt proportionaler Zusammenhang zwischen Eingang und Ausgang via Durchgangsmatrix \underline{D} angegeben werden. Im Vergleich zum dynamischen Teil, welcher mittels der Zustandsgleichungen beschrieben wird, stellt ($\underline{D} \cdot u(t)$) den statischen (nicht dynamischen) Teil dar.

2.3.5. Graphische Interpretation der Zustandsraumdarstellung

Gerade weil die Zustandsraumdarstellung ein zentrales Element der Modellbildung ist, werden die Systemeigenschaften im Folgenden graphisch interpretiert.

[Abbildung 2.8](#) veranschaulicht die graphische Interpretation der Zustandsgleichung (2.23). Die Lösung des Differentialgleichungssystems bestehend aus n Differentialgleichungen 1. Ordnung erfolgt entweder analytisch oder numerisch. Die analytische Lösung

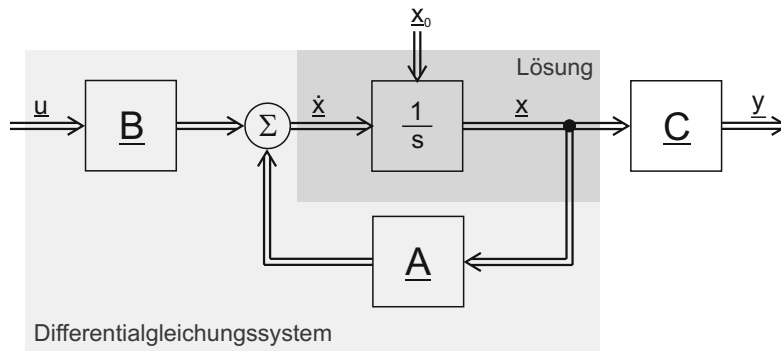


Abb. 2.8.: Graphische Interpretation der Zustandsgleichung

wird in [Abschnitt 2.4](#) behandelt, die numerische Lösung in [Kapitel 7](#). Letztere erfolgt in der Regel auf einem digitalen Rechner und wird als Simulation bezeichnet. Dabei wird das System, welches im Zustandsraum abgebildet ist, einem Solver übergeben.

In [Abbildung 2.9](#) ist der dynamische und der statische Teil der Zustandsraumdarstellung hervorgehoben. Physikalische Systeme setzen sich aus dynamischen und statischen

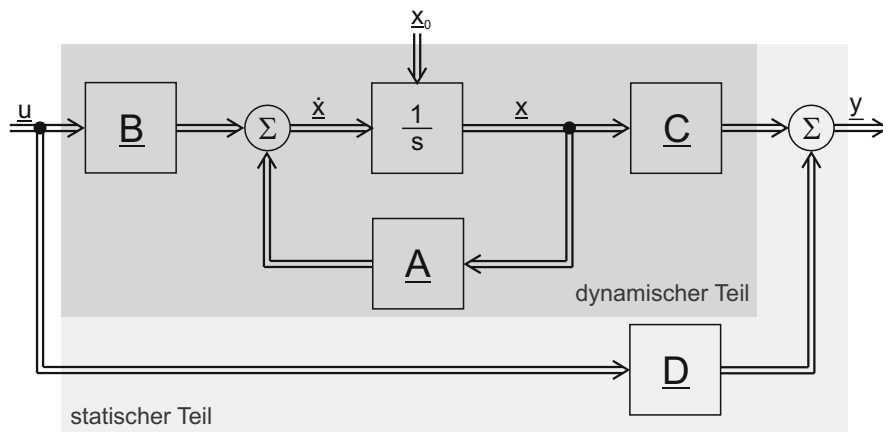


Abb. 2.9.: Graphische Interpretation der Zustandsraumdarstellung. Dynamischer und statischer Teil der Zustandsraumdarstellung

Anteilen zusammen. Erstere werden via Differentialgleichungen beschrieben, während letztere sogenannte algebraische Gleichungen verkörpern. Jedoch stehen solche (spezielle) algebraische Gleichungen in keinem Zusammenhang mit den Zustandsvariablen, wirken sich also direkt auf den Ausgang aus.

2.3.6. Wozu eigentlich Zustandsraumdarstellung?

Die Antwort auf diese Frage ist historisch begründet und wurde bereits zu Beginn dieses Kapitels zumindest teilweise beantwortet. Da zu Beginn der Simulationstechnik ausschließlich explizite Lösungsverfahren (Solver für gewöhnliche Differentialgleichungen, kurz ODE Solver) verwendet wurden, war für eine Simulation die Zustandsraumdarstellung zwingend nötig. Daraus ergab sich, dass sich auf der einen Seite Personen mit der Erstellung der Modelle beschäftigten, die anderen kümmerten sich darum, diese numerisch zu simulieren. Mit der Zustandsraumdarstellung wurde hier eine sehr gute Schnittstelle zwischen diesen beiden Fachgebieten eingeführt.

2.4. Analytische Lösung der Zustandsgleichung

Die Simulation physikalischer Systeme, welche in [Kapitel 7](#) ausführlich beschrieben wird, ist nichts Anderes, als die numerische Lösung gewöhnlicher Differentialgleichungen, welche dem Solver in Form der Zustandsraumdarstellung zur Verfügung gestellt wird.

2.4.1. Autonome skalare Systeme (Homogene Lösung)

Im Folgenden wird zusätzlich erklärt, wie Systeme im Zustandsraum analytisch gelöst werden. Dazu wird der Einfachheit halber die Zustandsgleichung eines autonomen LTI-Systems *1. Ordnung* herangezogen. Gleichung (2.31) reduziert sich daher zur skalaren Gleichung:

$$\dot{x}(t) = a \cdot x(t), \quad x(0) = x_0 \tag{2.36}$$

Lösung im Zeitbereich

Diese einfache gewöhnliche Differentialgleichung mit konstanten Koeffizienten lässt sich durch den *Separationsansatz* lösen. Wir erhalten:

$$\begin{aligned} \frac{dx(t)}{dt} &= a \cdot x(t) \\ \frac{1}{x} \cdot dx &= a \cdot dt \\ \int \frac{1}{x} \cdot dx &= \int a \cdot dt \\ \ln(x) + C &= a \cdot t \end{aligned}$$

Da die Integrationskonstante (der Parameter) C die Anfangsbedingung repräsentiert und daher separat und individuell bestimmt werden muss, können wir diese ohne das Vorzeichen zu ändern genauso gut auf die rechte Seite der Gleichung schreiben.

$$\begin{aligned}\ln(x) &= a \cdot t + C \\ x &= e^{a \cdot t + C} \\ x &= e^{a \cdot t} \cdot e^C\end{aligned}$$

Für den Ausdruck e^C schreiben wir abkürzend einfach nur C und erhalten schlussendlich:

$$x = C \cdot e^{a \cdot t} \tag{2.37}$$

Gleichung (2.37) wird als *allgemeine Lösung* bezeichnet. Dies liegt an der Integrationskonstante C , welche noch für eine spezielle Anfangsbedingung bestimmt werden muss. Durch lösen des Anfangswertproblems wird die allgemeine Lösung zur gesuchten Lösung. Dies wird nun anhand einiger konkreter Beispiele demonstriert.

Beispiel 1

Als erstes Beispiel lösen wir nun folgende Differentialgleichung:

$$\dot{x}(t) = a \cdot x, \quad x(t = 0) = x_0 = 1 \tag{2.38}$$

Wobei $a = 1$ sein soll. Wir erhalten die Lösung

$$x = C \cdot e^t,$$

wobei wir C noch für die Anfangsbedingung $x(t = 0) = 1$ bestimmen müssen. Setzen wir diese ein, so erhalten wir:

$$x(t = 0) = C \cdot e^0 = 1$$

Daraus folgt unmittelbar, dass

$$C = 1$$

ist und die *spezielle Lösung* der Differentialgleichung daher wie folgt lautet:

$$x = e^t \tag{2.39}$$

Beispiel 2

Als zweites Beispiel lösen wir die Differentialgleichung:

$$\dot{x}(t) = a \cdot x, \quad x_0 = 3 \tag{2.40}$$

Wobei $a = 1$ sein soll. Natürlich erhalten wir dieselbe allgemeine Lösung wie zuvor. Für C hingegen erhalten wir:

$$x(t = 1) = C \cdot e^0 = 3 \tag{2.41}$$

Daraus folgt, dass

$$C = 3$$

ist und sich die *spezielle Lösung* der Differentialgleichung daher wie folgt ergibt:

$$x = 3 \cdot e^t \tag{2.42}$$

Die grafische Interpretation der speziellen Lösungen (2.39) und (2.42) ist in [Abbildung 2.10](#) veranschaulicht. Es ist sehr gut ersichtlich, dass sich die Funktionen auf

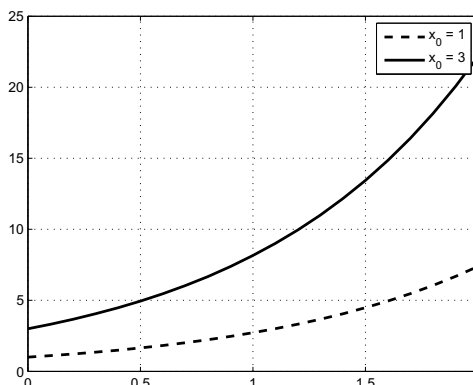


Abb. 2.10.: Lösungen der Differentialgleichung $\dot{x}(t) = x$ für zwei verschiedene Anfangsbedingungen x_0 .

Grund der Anfangsbedingungen unterschiedlich entwickeln. Weiters ist hier gut ersichtlich, dass die Systemmatrix \underline{A} (hier ein Skalar a) tatsächlich für die Stabilität des Systems verantwortlich ist. Um ein stabiles Verhalten zu garantieren, darf a ausschließlich negative Werte annehmen. Allgemein gilt, dass die Eigenwerte der Systemmatrix \underline{A} , welche via Gleichung (2.32) berechnet werden, negativ sein müssen, damit das System stabil ist. Die bisherigen Beispiele besitzen alle ein positiven Eigenwert und sind

daher ein instabil.

Beispiel 3

Als letztes Beispiel wollen wir ein stabiles System betrachten und setzen den Skalar a daher auf den Wert -1 . Wir erhalten:

$$\dot{x}(t) = -1 \cdot x, \quad x_0 = 1 \quad (2.43)$$

Aufgrund der Anfangsbedingung $x_0 = 1$ ist auch $C = 1$ und wir erhalten folgende spezielle Lösung der Differentialgleichung

$$x = e^{-t} \quad (2.44)$$

welche in [Abbildung 2.11](#) dargestellt ist:

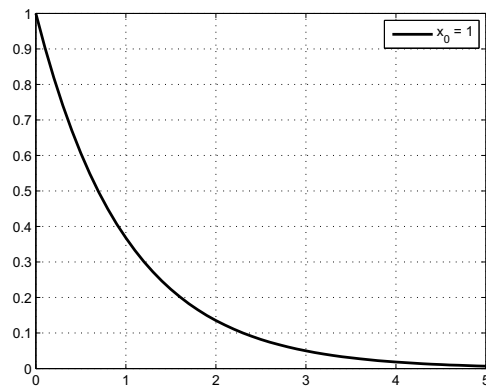


Abb. 2.11.: Lösung der Differentialgleichung $\dot{x}(t) = -1 \cdot x$ für die Anfangsbedingungen $x_0 = 1$.

Der vollständigkeit halber wird im Folgenden gezeigt wie solche System analytisch im Frequenzbereich gelöst werden können.

Lösung im Frequenzbereich

Gleichung (2.36) kann genauso gut via Laplace-Transformation im Frequenzbereich gelöst werden.

$$s \cdot X(s) - x_0 = a \cdot X(s) \quad (2.45)$$

Diese Gleichung wird nun nach $X(s)$ aufgelöst.

$$X(s) = \frac{1}{s-a} \cdot x_0 \quad (2.46)$$

Über eine Rücktransformation in den Zeitbereich erhalten wir als allgemeine Lösung:

$$x(t) = x_0 \cdot e^{a \cdot t} \quad (2.47)$$

Diese Gleichung entspricht Gleichung (2.37) welche im Zeitbereich ermittelt wurde. In Gleichung (2.47) ist gut ersichtlich, dass die Integrationskonstante C die Anfangsbedingung x_0 repräsentiert.

2.4.2. Inhomogene skalare Lösung

Dazu wird die homogene Gleichung (2.36) um einen inhomogenen (partikulären) Anteil erweitert. Wir erhalten somit eine skalare Version von Gleichung (2.23).

$$\dot{x}(t) = a \cdot x(t) + b \cdot u(t), \quad x(0) = x_0 \quad (2.48)$$

Nun kann diese Zustandsgleichung wiederum im Zeitbereich oder im Frequenzbereich gelöst werden. Der Einfachheit halber werden wir diesen Schritt lediglich im Frequenzbereich demonstrieren. Durch Anwenden der Laplace-Transformation erhalten wir:

$$s \cdot X(s) - x_0 = a \cdot X(s) + b \cdot U(s) \quad (2.49)$$

Auflösen nach $X(s)$ liefert

$$X(s) = \frac{1}{s-a} \cdot x_0 + \frac{1}{s-a} \cdot b \cdot U(s) \quad (2.50)$$

Durch Rücktransformation in den Zeitbereich erhält man die allgemeine Lösung.

$$x(t) = e^{a \cdot t} \cdot x_0 + \int_0^t e^{a(t-\tau)} b \cdot u(\tau) \cdot d\tau \quad (2.51)$$

Wobei der erste Term auf der rechten Seite des Gleichheitszeichens die homogene Lösung (Eigenbewegung des Systems) und der zweite Term die partikuläre Lösung (erzwungener Anteil) bezeichnet.

2.4.3. Inhomogene Lösung

Sobald ein physikalisches System mehr als eine Zustandsgröße aufweist entsteht ein System linearer Differentialgleichungen 1. Ordnung mit konstanten Koeffizienten welches entweder homogener oder inhomogener Natur sein kann. In beiden Fällen ist eine analytische Lösung nicht mehr so trivial. Hierzu wird die skalare Systemmatrix a aus Gleichung (2.51) einfach durch \underline{A} ersetzt (auf den Beweis, dass diese Substitution durchgeführt werden kann, sei hier verzichtet und auf [Unb00] verwiesen).

$$\underline{x}(t) = \underbrace{e^{\underline{A} \cdot t}}_{\underline{\phi}} \cdot \underline{x}_0 + \int_0^t \underbrace{e^{\underline{A}(t-\tau)}}_{\underline{\phi}(t-\tau)} \underline{b} \cdot u(\tau) \cdot d\tau \quad (2.52)$$

Wobei $\underline{\phi}$ als Fundamentalmatrix bezeichnet wird.

$$\underline{\phi} = e^{\underline{A} \cdot t} \quad (2.53)$$

Aufgrund bekannter Anfangsbedingungen \underline{x}_0 und des bekannten Verlaufs der Eingangsgröße $u(t)$, sind die Zustandsvariablen $\underline{x}(t)$, wie bereits in [Unterabschnitt 2.2.2](#) erwähnt, jederzeit bekannt. Wir wissen also exakt, wie sich unser System verhält.

3. Klassische Modellbildung: Beschreibung physikalischer Systeme mit DAEs

Prinzipiell lässt sich die mathematische Modellierung physikalischer Systeme in folgende Schritte unterteilen (siehe [Tabelle 3.1](#)): Um ein besseres Verständnis für diese Art der

(1) Analyse der physikalischen Struktur
(2) Aufstellen der Komponentengleichungen
(3) Aufstellen der Topologiegleichungen
(4) Lösbares Gleichungssystem erstellen (<i>Kausalisieren/Sortieren</i>)
(5) Implementierung des Gleichungssystems
(6) Simulationsergebnisse

Tab. 3.1.: Vorgangsweise bei der Erstellung eines mathematischen Modells.

Modellierung zu bekommen, wird die Beschreibung des Systems mittels DAEs anhand einer elektrischen Schaltung in den folgenden Abschnitten demonstriert. Es wird eine passive elektrische Schaltung, wie sie in [Abbildung 3.1](#) dargestellt ist, modelliert. Eine detailliertere Beschreibung ist in [\[Cel91\]](#) und [\[CK06\]](#) zu finden. Obwohl es in der Praxis kaum mehr der Fall ist, dass elektronische Schaltungen mittels DAEs beschrieben werden, da es sehr viele Modellierungssprachen gibt welche eine direkte Implementierung der Schaltung ermöglichen (z.B. Spice oder Modelica), wird diese Methode im Folgenden vorgestellt. Grund für dieses Beispiel ist, dass elektronische Schaltungen sehr einfach durch DAEs repräsentiert werden können und sich daher hervorragend anbieten, um das Prinzip zu erläutern. Für die Simulation in [Abschnitt 3.6](#) werden folgende Werte verwendet: $u = 30V$, $R_1 = 10\Omega$, $R_2 = 1k\Omega$, $L = 1mH$ und $C = 22\mu F$

3.1. Analyse der physikalischen Struktur

Hier wird versucht ein reales System mittels grundlegenden technischen Elementen zu beschreiben. Die technischen Elemente müssen dabei mathematisch beschreibbar sein, um eine spätere Berechnung des Gesamtsystems zu ermöglichen. Einfache Beispiele sind Federn, Dämpfer, Massen etc. im mechanischen Bereich oder Widerstände,

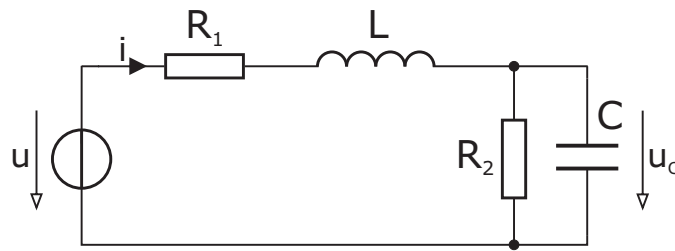


Abb. 3.1.: Passive elektronische Schaltung

Induktivitäten, Kapazitäten etc. im elektrischen Bereich. Komplexere Elemente sind verformbare Bleche oder Stäbe, Transistoren, Ventile und ähnliches. Meistens erstellt man aus den einzelnen Elementen, eine technische Skizze, welche im Falle elektrischer oder hydraulischer Systeme als Schaltplan bezeichnet wird.

Je genauer die physikalische Struktur bekannt ist bzw. je mehr Information wir über diese besitzen, desto besser ist auch das daraus resultierende Modell. Dies ist allerdings nicht in allen Fällen tatsächlich nötig und verlangsamt die anschließende Simulation teils deutlich. Man sollte sich daher zuerst einige Gedanken über den Einsatzbereich des zu erstellenden Modells machen. Im ersten Schritt ist in vielen Fällen ein recht einfaches (lineares) Modell des physikalischen Systems ausreichend. Es ist sogar eine sehr gebräuchliche Vorgangsweise, mit einfachen Modellen zu beginnen und die Komplexität schrittweise zu erhöhen, um so Fehler zu vermeiden. Diese Vorgangsweise nennt man *Top-down* Methode. Im Gegensatz dazu ist auch in manchen Fällen die *Bottom-up* Methode anzuwenden. Da dies allerdings Methoden sind, welche von einem Ingenieur von Fall zu Fall intuitiv erkannt werden, wird hier nicht weiter auf diese Methoden eingegangen.

In diesem konkreten Fall (wie meistens in der Elektronik) entfällt Punkt (1) „Analyse der physikalischen Struktur“ aufgrund der gegebenen elektrischen Schaltung.

Gleichungen und Unbekannte

Die Schaltung besteht aus fünf Komponenten (eine Spannungsquelle und vier passive Bauteile). Der Zustand jeder Komponente wird durch zwei Variablen eindeutig beschrieben¹. Hier wären dies die am Bauteil anliegende Spannung und der dadurch fließende Strom. Es sind also zehn Variablen für die Beschreibung des Systems nötig. Um ein lösbares Gleichungssystem zu erhalten, müssen nun auch zehn Gleichungen gefunden werden.

¹Wenn auch z.B. beim elektrischen Widerstand sich die eine Größe über das ohmsche Gesetz direkt aus der anderen Berechnen lässt

3.2. Aufstellen der Komponentengleichungen

Die Komponentengleichungen beschreiben, wie der Name schon sagt, das Verhalten einzelner Komponenten des Systems. Im vorliegenden Beispiel sind dies das ohmsche Gesetz sowie Gleichungen für Induktivität und Kapazität. Sie stellen meist einen Zusammenhang zwischen den beiden Größen her, welche das Element beschreiben. In diesem Fall wird also ein Zusammenhang zwischen Strom und Spannung hergestellt. Teilweise (wie hier bei der Spannungsquelle) wird auch einfach eine der beiden Größen durch das Element vorgegeben.

Die Gleichungen ergeben wie folgt:

$$u = 30V \quad (3.1)$$

$$u_{R_1} - i \cdot R_1 = 0 \quad (3.2)$$

$$u_L - L \cdot \frac{di}{dt} = 0 \quad (3.3)$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \quad (3.4)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad (3.5)$$

3.3. Aufstellen der Topologiegleichungen

Im Unterschied zu den Komponentengleichungen beschreiben die Topologiegleichungen die Verknüpfung der einzelnen Komponenten (Bauteile) miteinander.

Drei triviale Gleichungen würden viele Ingenieure erst gar nicht in Betracht ziehen. Sie ergeben sich aus der Serienschaltung der Spannungsquelle und von R_1 und L , sowie der Parallelschaltung von R_2 und C .

$$i = i_{R_1} \quad (3.6)$$

$$i_{R_1} = i_L \quad (3.7)$$

$$u_{R_2} = u_C \quad (3.8)$$

Um die unbekanntenen Größen bestimmen zu können, benötigen wir noch zwei weitere Gleichungen. Diese werden mittels Kirchhoff'scher Regeln bestimmt und beschreiben den Zusammenhang der Systemvariablen.

$$u - u_{R_1} - u_L - u_C = 0 \quad (3.9)$$

$$i - i_{R_2} - i_C = 0 \quad (3.10)$$

Die Komponentengleichungen sowie die Topologiegleichungen wurden so erstellt, dass

jeweils auf der rechten Seite des Gleichheitszeichens eine Null steht. Gleichungen in dieser Form werden als **implizite DAEs** bezeichnet.

3.4. Lösbares Gleichungssystem erstellen (Kausalisieren/Sortieren)

Um eine explizite Darstellung der DAEs zu erhalten, müssen die Gleichungen sortiert und in eine lösbare Form² gebracht werden, d.h. dass jeweils nur bekannte Größen auf der rechten Seite des Gleichheitszeichens stehen dürfen. Man spricht dann nicht mehr von Gleichungen sondern von Zuweisungen. Mit anderen Worten: Die *akausalen* Gleichungen wurden *kausalisiert*³. Anstelle des Gleichheitszeichens einer Gleichung wird innerhalb einer Zuweisung zur eindeutigen Kennzeichnung fortan der Zuweisungsoperator „:=“ verwendet.

Die bekannten Größen (Inputs) und die Unbekannte (Output) sind durch das Kausalisieren eindeutig bestimmt. Die Variable auf der linken Seite des Gleichheitszeichens einer Zuweisung stellt den Output dar und die Variablen auf der rechten Seite die Inputs. An dieser Stelle sei noch erwähnt, dass Zuweisungen (kausalisierte Gleichungen) nur einen Output besitzen können. Mathematisch ausgedrückt, heißt das nichts anderes, als dass in einer Gleichung auch nur eine Unbekannte berechnet werden kann. Dieser Prozess wird auch als *horizontales Sortieren* bezeichnet. Damit die kausalisierten Gleichungen von einem Programm interpretiert werden können, müssen diese zusätzlich *vertikal sortiert* werden, d.h. dass eine Variable erst dann benutzt wird, wenn sie bereits berechnet wurde. Den Prozess des horizontalen- und vertikalen Sortierens bezeichnet man als *symbolische Vorverarbeitung* (mehr dazu in Abschnitt 6.9).

²Lösbar bedeutet hier, dass diese von einem einfachen, z.B. in der Programmiersprache C erstellten, Programm verarbeitet werden können

³Es handelt sich nicht um die physikalische Ursache-Wirkungs-Beziehung sondern um die sogenannte rechnerische Kausalität (engl. computational causality)

Diese führt zu folgendem Ergebnis:

$$u := 30V \quad (3.11)$$

$$u_{R_1} := i \cdot R_1 \quad (3.12)$$

$$i_{R_2} := \frac{1}{R_2} \cdot u_C \quad (3.13)$$

$$u_L := u - u_{R_1} - u_C \quad (3.14)$$

$$\frac{di}{dt} := \frac{1}{L} \cdot u_L \quad (3.15)$$

$$i_C := i - i_{R_2} \quad (3.16)$$

$$\frac{du_C}{dt} := \frac{1}{C} \cdot i_C \quad (3.17)$$

Bei der Entwicklung mathematischer Modelle per „Block und Bleistift“ wird das Kausalisieren im Allgemeinen etwas unstrukturiert und unüberlegt durchgeführt. Bei Programmen müssen hingegen geeignete Algorithmen bzw. Heuristiken formuliert werden, welche ein erfolgreiches horizontales- und vertikales Sortieren ermöglichen. Die Schritte und die damit verbundenen Probleme welche bei einer symbolischen Vorverarbeitung nötig sind, werden in [Abschnitt 6.9](#) vorgestellt.

3.5. Implementierung des Gleichungssystems

Es gibt verschiedenste Möglichkeiten das erhaltene Gleichungssystem berechnen zu lassen. Diese stellen unterschiedliche Anforderungen an den Modellierer. Sie reichen von der manuellen Implementierung in einer Programmiersprache (inklusive Lösungsalgorithmus) bis hin zur Lösung in speziell dafür erstellten Programmen (MATLAB, Dymola etc.). Im Folgenden werden drei der in der Ingenieurspraxis am gängigsten Verfahren beschrieben:

- (1) Zustandsraumdarstellung
- (2) Blockschaltbild
- (3) Übertragungsfunktion

3.5.1. Zustandsraumdarstellung (ODE)

Die **expliziten DAEs** werden nun zusammengesetzt. Die Unbekannten auf der rechten Seite werden durch die Ausdrücke durch die sie definiert sind substituiert. Schlussendlich führt dieser Schritt zu gewöhnlichen Differentialgleichungen erster Ordnung (ordinary differential equations - (*explicit*) **ODE**), der Zustandsraumdarstellung.

Um nun von obigen Gleichungen, unter Berücksichtigung der Kausalität, auf die Zustandsraumdarstellung zu gelangen sind, wie schon in [Abschnitt 2.3](#) beschrieben, folgende Schritte notwendig:

Bestimmen der Ein- und Ausgangsgrößen

Die Eingangsgröße ist in unserem Beispiel gleich der Eingangsspannung.

$$u(t) = u \tag{3.18}$$

Als Ausgangsgröße soll die Spannung u_C am Kondensator herangezogen werden

$$y(t) = u_C \tag{3.19}$$

Bestimmen der Zustandsgrößen

In unserem Beispiel sind die Zustandsgrößen die Ausgangsgrößen des Kondensator und der Induktivität in ihrer integralen Form herangezogen. Diese ergeben sich über Integration aus den Gleichungen (3.15) und (3.17).

$$u_C = \frac{1}{C} \int i_c \cdot dt \equiv x_1 \tag{3.20}$$

$$i = \frac{1}{L} \int u_L \cdot dt \equiv x_2 \tag{3.21}$$

Das erste Element $\dot{x}_1(t)$ des Spaltenvektors $\dot{\underline{x}}(t)$ erhält man wie folgt: Man leite die Zustandsgleichung (3.20) nach der Zeit t ab⁴ und substituiere dann solange bis nur noch Eingangs- und Zustandsgrößen in der Gleichung vorhanden sind.

Aufstellen der ersten Zustandsgleichung

$$\begin{aligned} \frac{du_C}{dt} = \dot{x}_1 &= \frac{1}{C} \cdot i_c \\ \dot{x}_1 &= \frac{1}{C} (i - i_{R_2}) = \frac{1}{C} \left(i - \frac{u_C}{R_2} \right) \end{aligned} \tag{3.22}$$

Durch Einsetzen der Zustandsgrößen und gegebenenfalls der Eingangsgröße erhält man

⁴Oder man verwendet direkt Gleichung (3.17).

schlussendlich folgende Gleichung:

$$\begin{aligned}\dot{x}_1 &= \frac{1}{C} \left(x_2 - \frac{1}{R_2} \cdot x_1 \right) \\ \dot{x}_1 &= -\frac{1}{R_2 \cdot C} \cdot x_1 + \frac{1}{C} \cdot x_2\end{aligned}\quad (3.23)$$

Aufstellen der zweiten Zustandsgleichung

Die Zustandsgleichung für \dot{x}_2 wird analog zu \dot{x}_1 gebildet:

$$\begin{aligned}\frac{di}{dt} = \dot{x}_2 &= \frac{1}{L} \cdot u_L \\ \dot{x}_2 &= \frac{1}{L} (u - x_2 \cdot R_1 - x_1) \\ \dot{x}_2 &= -\frac{1}{L} \cdot x_1 - \frac{R_1}{L} \cdot x_2 + \frac{1}{L} \cdot u\end{aligned}\quad (3.24)$$

Bilden der Zustandsraumdarstellung

In Matrixschreibweise erhält man:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -\frac{1}{C \cdot R_2} & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R_1}{L} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{L} \end{pmatrix} \cdot u(t)\quad (3.25)$$

$$y(t) = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\quad (3.26)$$

Wie erwartet stimmt das Ergebnis exakt mit dem aus [Unterabschnitt 2.2.4](#) (Gleichungen (2.12) und (2.16)) überein. Anhand dieser Ergebnisse sind wir bereits in der Lage das System zu simulieren. Mehr dazu in [Abschnitt 3.6](#).

3.5.2. Blockschaltbild

Das Blockschaltbild erhält man aus den kausalisierten Gleichungen (3.11-3.17) (Zuweisungen) indem man für jede Zuweisung die dazugehörigen Blöcke zeichnet und diese miteinander verbindet. Die Blöcke der kausalen Komponentengleichungen sind in [Abbildung 3.2](#) dargestellt. Die kausalen Topologiegleichungen werden mit *Summen* realisiert (siehe [Abbildung 3.3](#)). Durch Zusammenfügen aller Blöcke erhalten wir das endgültige Blockschaltbild welches in [Abbildung 3.4](#) dargestellt ist.

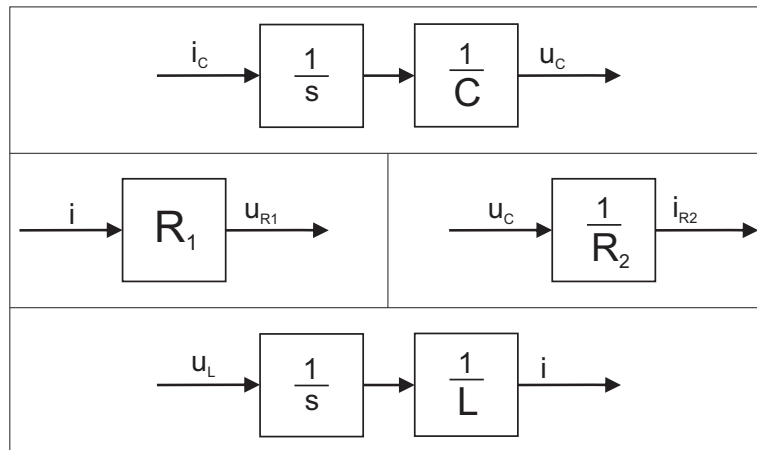


Abb. 3.2.: Kausale Komponentengleichungen dargestellt als Blöcke

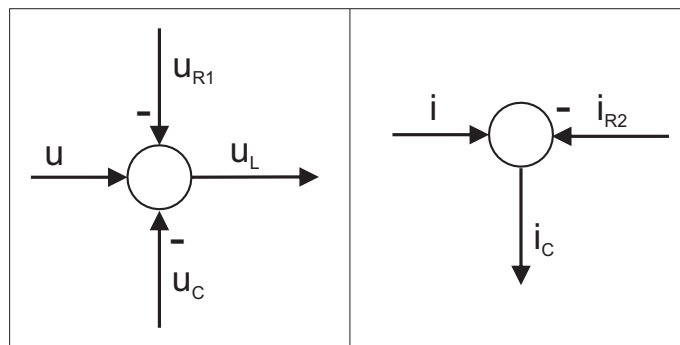


Abb. 3.3.: Kausale Topologiegleichungen dargestellt als Blöcke

3.5.3. Übertragungsfunktion

Die Übertragungsfunktion des Systems wird entweder aus der Zustandsraumdarstellung oder durch Bilden des Verhältnisses Ausgang zu Eingang ermittelt:

Übertragungsfunktion abgeleitet aus der Zustandsraumdarstellung

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t) \quad (3.27)$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) + \underline{D} \cdot \underline{u}(t) \quad (3.28)$$

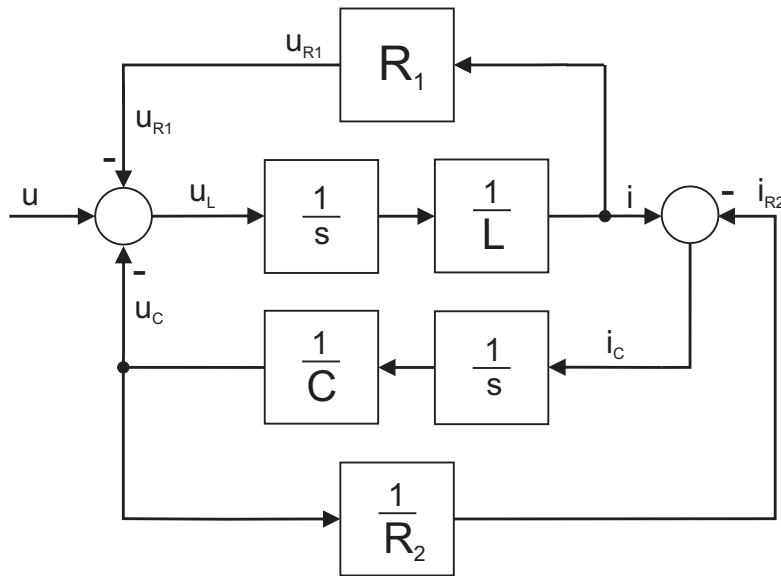


Abb. 3.4.: Blockschaltbild der elektrischen Schaltung

Dazu wird die *Laplace-Transformation* auf beide Gleichungen angewendet:

$$s\underline{X}(s) = \underline{A} \cdot \underline{X}(s) + \underline{B} \cdot \underline{U}(s) \quad (3.29)$$

$$\underline{Y}(s) = \underline{C} \cdot \underline{X}(s) + \underline{D} \cdot \underline{U}(s) \quad (3.30)$$

Wobei, für das behandelte Beispiel, $\underline{D} = 0$ zu setzen ist und \underline{B} und \underline{C} wiederum Vektoren sind ($\underline{B} \rightarrow \underline{b}$, $\underline{C} \rightarrow \underline{c}^T$).

Umformen von Gleichung (3.29) nach $\underline{X}(s)$ und Einsetzen in Gleichung (3.30) ergibt:

$$\underline{X}(s) = (s\mathbf{1} - \underline{A})^{-1} \cdot \underline{b} \cdot \underline{U}(s) \quad (3.31)$$

$$\underline{Y}(s) = \underline{c}^T \cdot (s\mathbf{1} - \underline{A})^{-1} \cdot \underline{b} \cdot \underline{U}(s) \quad (3.32)$$

Die Übertragungsfunktion des Systems erhält man, indem der Ausgang $\underline{Y}(s)$ (Gleichung (3.31)) durch den Eingang $\underline{X}(s)$ (Gleichung (3.32)) dividiert wird.

$$G = \frac{\underline{Y}(s)}{\underline{U}(s)} = \underline{c}^T \cdot (s\mathbf{1} - \underline{A})^{-1} \cdot \underline{b} \quad (3.33)$$

Durch Einsetzen von A , b und c^T erhalten wir:

$$G = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} s + \frac{1}{R_2 \cdot C} & -\frac{1}{C} \\ \frac{1}{L} & s + \frac{R_1}{L} \end{pmatrix}^{-1} \cdot \begin{pmatrix} 0 \\ \frac{1}{L} \end{pmatrix} \quad (3.34)$$

Durch Lösen obiger Gleichung erhält man schlussendlich:

$$G = \frac{R_2}{R_2 + (R_1 + sL)(1 + sC \cdot R_2)} \quad (3.35)$$

Übertragungsfunktion durch Bilden des Verhältnisses Ausgang zu Eingang

Die Ausgangsspannung u_C liegt unmittelbar an den Komponenten R_2 und C an. Die Eingangsspannung hingegen an den Komponenten R_1 , L und der Parallelschaltung aus C und R_2 .

Die Impedanz ist wie folgt definiert

$$Z = \operatorname{Re}(Z) + j \operatorname{Im}(Z) = R + jX \quad (3.36)$$

Wobei für X je nachdem ob es sich um eine Induktivität oder eine Kapazität handelt die entsprechenden Blindwiderstände eingesetzt werden.

$$X_L = j\omega L \quad (3.37)$$

$$X_C = \frac{1}{j\omega C} \quad (3.38)$$

Die Übertragungsfunktion G wird wie folgt gebildet:

$$G = \frac{u_C}{u} = \frac{C \parallel R_2}{R_1 + X_C + C \parallel R_2} \quad (3.39)$$

Für die Parallelschaltung erhalten wir:

$$\begin{aligned} C \parallel R_2 &= \frac{\frac{1}{j\omega C} \cdot R_2}{R_2 + \frac{1}{j\omega C}} = \\ &= \frac{R_2}{1 + j\omega C \cdot R_2} \end{aligned} \quad (3.40)$$

Einsetzen von Gleichung (3.40) in Gleichung (3.39) ergibt:

$$\begin{aligned} G &= \frac{\frac{R_2}{1+j\omega C \cdot R_2}}{R_1 + X_C + \frac{R_2}{1+j\omega C \cdot R_2}} = \frac{R_2}{R_2 + (R_1 + j\omega L)(1 + j\omega C \cdot R_2)} \Big|_{j\omega=s} = \\ &= \frac{R_2}{R_2 + (R_1 + sL)(1 + sC \cdot R_2)} \end{aligned} \quad (3.41)$$

Wie man hier sehr gut erkennt, entspricht Gleichung (3.41) der zuvor ermittelten Gleichung (3.35).

3.6. Simulationsergebnisse

3.6.1. Simulation via Zustandsraumdarstellung

Im Folgenden werden die erstellten Modelle mit MATLAB/Simulink simuliert. Bis jetzt wurde anhand von DAEs eine Zustandsraumdarstellung der passiven elektronischen Schaltung erstellt. Die vier Matrizen der Zustandsraumdarstellung werden nun mit MATLAB ausgewertet (Code 3.1).

```

1  % Example 1: Modeling an electric circuit
2
3  % Input
4  u = 30;
5
6  % Parameter
7  R1 = 10; R2 = 1000;
8  L = 1E-3;
9  C = 22E-6;
10
11 % State-Space Representation of the System
12 a11 = -1/(R2*C);
13 a12 = 1/C;
14 a21 = -1/L;
15 a22 = -R1/L;
16
17 A = [a11, a12; a21, a22];
18 b = [0; 1/L];
19 cT = [1 0];
20 d = 0;
21
22 sys = ss(A, b, cT, d)           % creating the state-space model
23
24 % Simulation of the State-Space Model
25 t = [ 0 : 1e-5 : 1.4e-3 ];    % time base
26 u = u * ones(size(t));        % Input vector
27 x0 = zeros(2, 1);             % initial conditions
28 y = lsim(sys, u, t, x0);      % LTI simulation
29
30 plot(t, y, 'LineWidth', 2);
31 grid on;

```

Code 3.1: Matlab Code der elektrischen Schaltung, modelliert im Zustandsraum

Das Ergebnis der Simulation ist in [Abbildung 3.5](#) ersichtlich.

Die eigentliche Simulation des Modells erfolgt in Zeile 28 von [Code 3.1](#). Hier wird ein expliziter Solver mit einer bestimmter Ordnung aufgerufen, welcher auf lineare Systeme optimiert ist. Eine detaillierte Beschreibung dieser Solver folgt in [Kapitel 7](#).

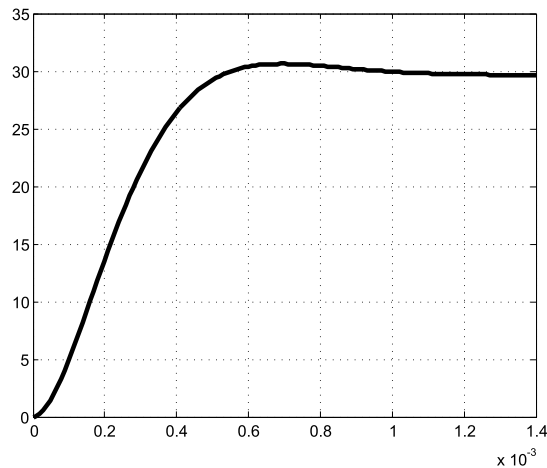


Abb. 3.5.: Simulationsergebnis bei einer Eingangsspannung von $30V_{DC}$

3.6.2. Simulation via Blockschaltbild

Das Blockschaltbild aus [Abbildung 3.4](#) wird nun mit Simulink erstellt ([Abbildung 3.6](#) zeigt das Blockschaltbild).

Das Ergebnis der Simulation ist identisch zu dem in [Abbildung 3.5](#).

3.6.3. Simulation via Übertragungsfunktion

Übertragungsfunktion via Zustandsraumdarstellung

Die Simulation des Zustandsraums wurde bereits in [Unterabschnitt 3.6.1](#) durchgeführt. Um die Übertragungsfunktion simulieren zu können wird das Zustandsraummodell in MATLAB mit Hilfe der Funktion `tf()` in eine Übertragungsfunktion umgewandelt und anschließend mit der Funktion `step()` simuliert. Die Simulation des vorherigen Beispiels ([Code 3.1](#)) wird bis zu *Zeile 22* übernommen. Die restlichen Zeilen werden mit durch [Code 3.2](#) ersetzt:

```

1 G = u*tf(sys); % Transfer Function of the System
2 step(G); % Stepresponse of the system

```

Code 3.2: MATLAB-Code der elektrischen Schaltung, modelliert via Übertragungsfunktion welche aus der Zustandsraumdarstellung abgeleitet wurde

In *Zeile 1* von [Code 3.2](#) wurde die Übertragungsfunktion mit dem Eingang u multipliziert. Dies ist notwendig da die Funktion `step()` eine Amplitude von eins erzeugt.

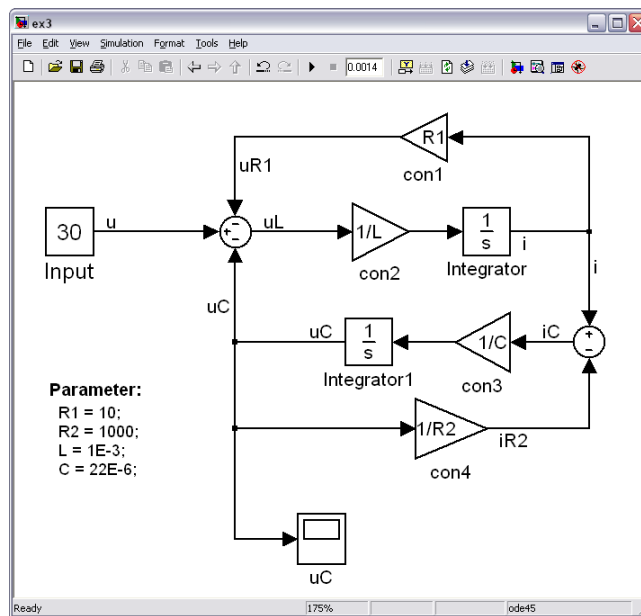


Abb. 3.6.: Blockschaltbild der elektronischen Schaltung

Wir hingegen benötigen jedoch die Antwort des Systems auf eine Eingangsspannung von $u = 30V$.

Übertragungsfunktion via Verhältnis Ausgang zu Eingang

Hier wird nun Gleichung (3.41) verwendet. Dies wird wiederum mit der Funktion $\text{tf}()$ realisiert. Diesmal werden die Koeffizienten allerdings direkt eingegeben: Dazu muss Gleichung (3.41) noch ausmultipliziert werden, sodass die Ordnung des Systems ersichtlich wird. Wir erhalten:

$$G = \frac{R_2}{R_2 + R_1 + s \cdot (C \cdot R_1 \cdot R_2 + L) + s^2 \cdot (L \cdot C \cdot R_2)} \quad (3.42)$$

Im Folgenden wird der dazugehörige MATLAB-Code dargestellt (siehe Code 3.3). Die Parameter werden wiederum gleich eingegeben wie in Code 3.1 gezeigt.

```
1 G = u*tf([R2], [L*C*R2 (C*R1*R2 + L) (R1+R2)]);
2 step(G);
```

Code 3.3: float, MATLAB-Code der elektrischen Schaltung, modelliert via Übertragungsfunktion

4. Grundlegende Zusammenhänge physikalischer Systeme

In diesem Abschnitt werden die mathematischen Modelle physikalischer Systeme aus verschiedenen Domänen mit DAEs beschrieben. Es sollen die Gleichungen einfacher Systeme 2. Ordnung hergeleitet werden. Dies wird für elektrische, mechanisch-translatorische und mechanisch-rotatorische Systeme gezeigt. Ferner wird ein einfaches so genanntes „Fluss-System“ modelliert. „Einfach“ darum, weil die Flüssigkeit, welche sich im Medium befindet, als inkompressibel angenommen wird.

Am Ende dieses Abschnitts soll ersichtlich werden, dass allen Systemen eines gemeinsam ist, nämlich deren identische mathematische Beschreibung. Dieses Wissen bilden die Basis für die Einführung in die *Theorie der Bondgraphen*.

4.1. Elektrische Systeme

Abbildung 4.1 zeigt einen elektrischen RLC-Schwingkreis. Dieser besitzt folgende fun-

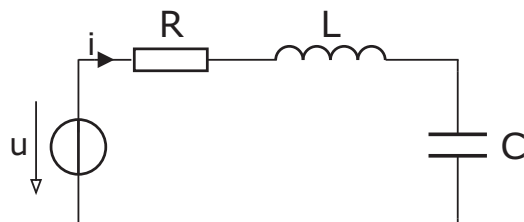


Abb. 4.1.: Schaltung eines elektrischen RLC-Schwingkreises

damentale Größen welche das System charakterisieren.

- Spannung u
- Strom i

Um eine Differentialgleichung, welche das System beschreibt, zu erhalten, wird die Kirchhoff'sche Maschenregel angewendet.

$$u = u_L + u_R + u_C \quad (4.1)$$

Durch Substitution der einzelnen Spannungen mit den Komponentengleichungen, welche die jeweiligen Elemente beschreiben, erhalten wir folgende Gleichung:

$$u = \underbrace{L \frac{di}{dt}}_{(I)} + \underbrace{R \cdot i}_{(II)} + \underbrace{\frac{1}{C} \int i \cdot dt}_{(III)} \quad (4.2)$$

Eine genauere Betrachtung der Terme (I), (II) und (III) des Ausdrucks auf der rechten Seite des Gleichheitszeichens von Gleichung (4.2) liefert folgende Informationen: Term (II) beschreibt die Verluste der Schaltung. Dies ist hier durch den elektrischen Widerstand R gekennzeichnet. Die elektrische Leistung, welche am Widerstand entsteht, wird irreversibel in Wärme umgewandelt. Die Terme (I) und (III) beschreiben das Speichern von Energie. Dies ist bei Term (III) sofort ersichtlich, da dieser in integraler Form vorliegt. Formt man Term (I) wie folgt um, so ist auch hier ersichtlich, dass Energie gespeichert wird.

$$u_L = L \frac{di}{dt} \quad (4.3)$$

$$i = \frac{1}{L} \int u_L \cdot dt \quad (4.4)$$

In Term (III) wird demnach der Strom i gespeichert, während in Term (I) die elektrische Spannung u gespeichert wird.

4.2. Mechanisch-translatorische Systeme

Als mechanisch-translatorisches Pendant wird das System in [Abbildung 4.2](#) herangezogen welches in der Mechanik als (*gedämpfter*) *Feder-Masse Schwinger* bekannt ist. Die

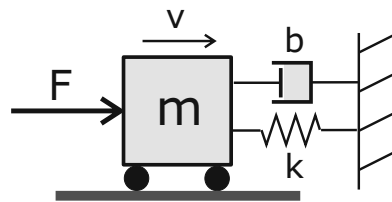


Abb. 4.2.: Feder-Masse Schwinger

fundamentalen Größen des Systems sind:

- Kraft F
- Geschwindigkeit v

Die Differentialgleichung, welche das System beschreibt, erhält man durch Bilden der Summe aller Kräfte, welche auf die Masse m wirken (*Prinzip von D'Alembert*):

$$F = F_m + F_b + F_k \quad (4.5)$$

Durch Substitution der Kräfte mit den dazugehörigen Komponentengleichungen ergibt sich folgende Differentialgleichung *2. Ordnung*:

$$\begin{aligned} F &= m \cdot \ddot{x} + b \cdot \dot{x} + k \cdot x = \\ &= \underbrace{m \cdot \dot{v}}_{(I)} + \underbrace{b \cdot v}_{(II)} + \underbrace{k \int v \cdot dt}_{(III)} \end{aligned} \quad (4.6)$$

Die Konstante b in Term (II) beschreibt wiederum ein verlustbehaftetes Element, welches im Falle eines Feder-Masse Schwingers einem Dämpfer entspricht. Allgemein gilt, dass mechanische Verluste nichts anderes sind als Reibung, d.h. dass auch hier die auftretende mechanische Leistung irreversibel in Wärme umgewandelt wird. Term (I) und (III) beschreiben das Speichern von Kraft bzw. Geschwindigkeit. Das Speichern der Kraft F_m wird ersichtlich wenn man Term (I) folgendermaßen umformt:

$$F_m = m \cdot \dot{v} \quad (4.7)$$

$$v = \frac{1}{m} \int F_m \cdot dt \quad (4.8)$$

4.3. Mechanisch-rotatorische Systeme

Ein mechanisch-rotatorisches System *2. Ordnung* ist in Abbildung 4.3 dargestellt. Die

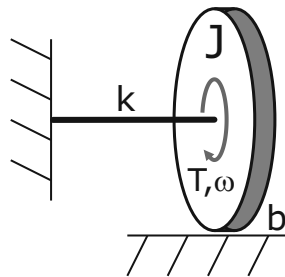


Abb. 4.3.: Mechanisch-rotatorisches System 2. Ordnung

fundamentalen Größen des Systems sind:

- Moment T

- Winkelgeschwindigkeit ω

Durch Bilden der Summe aller Momente welche auf die Masse wirken, die im Falle einer Rotation durch deren Trägheit J beschrieben wird, ergibt sich die Beziehung:

$$\begin{aligned}
 T &= T_J + T_b + T_k = \\
 &= J \cdot \ddot{\varphi} + b \cdot \dot{\varphi} + k \cdot \varphi = \\
 &= \underbrace{J \cdot \dot{\omega}}_{(I)} + \underbrace{b \cdot \omega}_{(II)} + \underbrace{k \int \omega \cdot dt}_{(III)}
 \end{aligned} \tag{4.9}$$

Term (II) beschreibt die rotatorischen Reibverluste. Term (I) und (III) beschreiben das Speichern von Energie, wobei Term (I) wiederum in integraler Form dargestellt wird, um das Speichern des Trägheitsmoments T_J zu veranschaulichen.

$$T_J = J \cdot \dot{\omega} \tag{4.10}$$

$$\omega = \frac{1}{J} \int T_J \cdot dt \tag{4.11}$$

4.4. Fluss-Systeme (Hydraulische Systeme)

Bei Systemen welche die Bewegung (Fluss) eines Mediums beschreiben, benötigen wir etwas mehr Hintergrundwissen um eine mathematisches Modell erstellen zu können, weshalb an dieser Stelle kein System 2. Ordnung herangezogen wird. Vielmehr wird auf die grundlegenden Eigenschaften solcher Systeme eingegangen. Der Einfachheit halber werden die Flüssigkeiten als inkompressibel angenommen, was für hydraulische Systeme durchaus legitim ist.

4.4.1. Trägheit einer Flüssigkeit

Die Trägheit einer Flüssigkeit wird schematisch durch eine lange Leitung (Rohr) dargestellt (siehe [Abbildung 4.4](#)).

Die fundamentalen Größen sind:

- Druck p
- Volumenstrom (Volumenfluss) q

Die Druckdifferenz an den Rohrenden wird durch den einfachen Zusammenhang

$$\Delta p = p_1 - p_2 \equiv p \tag{4.12}$$

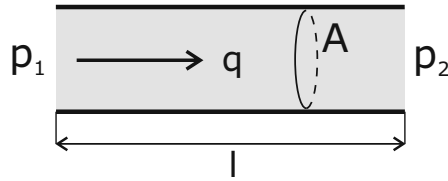


Abb. 4.4.: Trägheit einer Flüssigkeit dargestellt durch eine langes Rohr

beschrieben.

Die Trägheit einer Flüssigkeit lässt sich durch eine Druckdifferenz erklären welche dem Speichern von Druck entspricht. Um diesen Zusammenhang mathematische Beschreiben zu können, wird das *2. Newton'sche Gesetz* (Bewegungsgleichung) herangezogen.

$$F = m \cdot a = m \frac{dv}{dt} \quad (4.13)$$

Mit $F = p \cdot A$ und $m = \rho \cdot V = \rho \cdot l \cdot A$ wird

$$p \cdot A = \rho \cdot l \cdot A \cdot \frac{dv}{dt} \quad (4.14)$$

Den Volumenstrom q erhält man durch Multiplikation der Geschwindigkeit v mit der Querschnittsfläche A des Rohres.

$$q = v \cdot A \quad (4.15)$$

Durch Ableiten von Gleichung (4.15) und Einsetzen in Gleichung (4.14) erhalten wir:

$$\begin{aligned} p \cdot A &= \rho \cdot l \cdot \frac{dq}{dt} \\ p &= \underbrace{\frac{\rho \cdot l}{A}}_{L_f} \cdot \frac{dq}{dt} \end{aligned} \quad (4.16)$$

Wobei L_f als Trägheit (kg/m^4) der Flüssigkeit bezeichnet wird. Auflösen obiger Gleichung nach dem Volumenstrom q liefert

$$q = \frac{1}{L_f} \int p \cdot dt \quad (4.17)$$

In einem langen Rohr wird durch die Trägheit der Flüssigkeit also tatsächlich eine gewisse Druckdifferenz p gespeichert.

4.4.2. Flüssigkeitsspeicher - Der Tank

Abbildung 4.5 zeigt einen Flüssigkeitsspeicher. Das Volumen V des Tanks erhält man

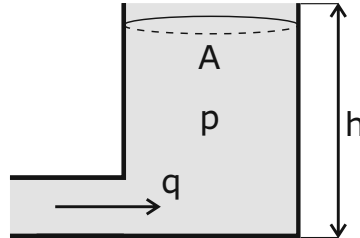


Abb. 4.5.: Speichern von Flüssigkeit - der Tank

indem dessen Querschnittsfläche A mit der Höhe h multipliziert wird.

$$V = A \cdot h \quad (4.18)$$

Wobei das Volumen V ebenfalls durch Speichern des Volumenstroms q beschrieben werden kann:

$$V = \int q \cdot dt \quad (4.19)$$

Den Druck am Boden des Tanks erhält man indem die Höhe h mit der Dichte ρ und der Gravitationskonstante g multipliziert wird.

$$p = \rho \cdot g \cdot h \quad (4.20)$$

Diese Gleichung beschreibt den *hydrostatischen* Druck und ist auch als *Pascal'sches Gesetz* bekannt. Sie beschreibt den Druck innerhalb einer ruhenden (statischen) Flüssigkeit unter dem Einfluss der Gravitation. Der Druck p hängt daher einzig und allein von dem *Füllstand* h des Tanks ab. Durch Einsetzen von Gleichung (4.18) in Gleichung (4.20) erhalten wir

$$p = \rho \cdot g \cdot \frac{V}{A} \quad (4.21)$$

wobei der konstante Ausdruck $\frac{\rho \cdot g}{A}$ durch die Konstante $\frac{1}{C_f}$ ersetzt wird. C_f deutet dabei auf eine Kapazität hin. Hält man sich nun die elektrische Domäne vor Augen, stellt man schnell fest, dass ein Kondensator Ladung speichert. Die zeitliche Änderung der Ladung entspricht dabei dem elektrischen Strom ($\frac{dq}{dt} = i$). Bei Fluss-Systemen entspricht der Volumenstrom q nichts anderem als dem elektrischen Strom i . Daher ist es völlig legitim die Speicherung einer Flüssigkeit in einem Tank mit einem kapazitiven Element zu beschreiben. Dies wird ersichtlich indem Gleichung (4.19) in Gleichung

(4.21) eingesetzt wird.

$$p = \frac{1}{C_f} \int q \cdot dt \quad (4.22)$$

4.4.3. Die Verengung (Querschnittsreduzierung)

Abbildung 4.6 zeigt eine Verengung in der Mitte eines Rohres.

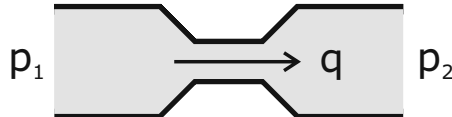


Abb. 4.6.: Verengung in der Mitte eines Rohres

Die Druckdifferenz $p_1 - p_2$ welche in Gleichung (4.12) beschrieben wurde, wird hier mit folgendem Ausdruck gleichgesetzt.

$$p = R_f \cdot q \quad (4.23)$$

Trifft eine Flüssigkeit auf eine Verengung, so entspricht dies einem Widerstand (resistives Element) welcher einen Druckabfall verursacht. Diese Druckdifferenz p tritt hier also nicht auf Grund der Trägheit der Flüssigkeit auf. Diese wird hier vernachlässigt. In anderen Worten: Es wird angenommen, dass das Rohr sehr kurz ist. Die Druckdifferenz resultiert in diesem Fall einzig und allein aus der Verengung des Rohres. Auch in diesem Fall entstehen Verluste in Form von Wärme. Diese kommen durch die Verwirbelungen des Medium zu Stande.

4.5. Entwickeln allgemeingültiger Gleichungen

Tabelle 4.1 veranschaulicht die Gleichungen aller physikalischen Systeme welche zuvor hergeleitet wurden.

Tabelle 4.1 unterteilt die Fähigkeit des Speicherns physikalischer Größen in zwei Gruppen. Zum einen werden so genannte *Flussvariablen* und zum anderen *Potentialvariablen* gespeichert. Des Weiteren wird sehr gut ersichtlich, dass in jedem Fall zwei Variablen miteinander verknüpft werden. Daher ist es sinnvoll für jede Zeile aus Tabelle 4.1 eine allgemeingültige Form zu entwickeln.

Wir definieren folgende, allgemeingültige Variablen:

- Flussvariable: $f(t)$

	Elektrische Systeme	Translatorische Systeme	Rotatorische Systeme	Hydraulische Systeme
<i>Wärmeverluste</i>	$u_R = R \cdot i$	$F_b = b \cdot v$	$T_b = b \cdot \omega$	$p = R_f \cdot q$
<i>Fluss-speicherung</i>	$u_c = \frac{1}{C} \int i \cdot dt$	$F_k = k \int v \cdot dt$	$T_k = k \int \omega \cdot dt$	$p = \frac{1}{C_f} \int q \cdot dt$
<i>Potential-speicherung</i>	$i = \frac{1}{L} \int u_L \cdot dt$	$v = \frac{1}{m} \int F_m \cdot dt$	$\omega = \frac{1}{J} \int T_J \cdot dt$	$q = \frac{1}{L_f} \int p \cdot dt$

Tab. 4.1.: Beziehungen, um Verluste und das Speichern von Fluss- und Potentialvariablen zu beschreiben

- Potentialvariable: (Einsatzvariable) $e(t)$

Betrachten wir [Tabelle 4.1](#) erneut. Die Konstanten der ersten Zeile beschreiben resistive (verlustbehaftete) Elemente. Wir führen daher eine neue Konstante ein um diese Elemente zu beschreiben und nennen sie γ . Die zweite Zeile beschreibt das Speichern einer Flussvariablen. Dies wird durch kapazitive Elemente verwirklicht. Kapazitive Elemente werden von nun an durch das Kürzel β gekennzeichnet. In der dritten Zeile wird die Potentialspeicherung beschrieben. Diese wird durch induktive Elemente (Trägheitselemente) beschrieben und fortan mit α bezeichnet. [Tabelle 4.2](#) veranschaulicht die verallgemeinerten Gleichungen:

	Allgemeingültige Beziehung
<i>Wärmeverluste</i>	$e(t) = \gamma \cdot f(t)$
<i>Fluss-speicherung</i>	$e(t) = \frac{1}{\beta} \int f(t) \cdot dt$
<i>Potential-speicherung</i>	$f(t) = \frac{1}{\alpha} \int e(t) \cdot dt$

Tab. 4.2.: Gleichungen welche die Speicherung von Energie bzw. die Verluste innerhalb einer Domäne beschreiben

[Tabelle 4.3](#) zeigt welche Konstanten der entsprechenden Domäne für α , β und γ eingesetzt werden müssen.

Es ist demnach tatsächlich der Fall, dass jedes physikalische System, unabhängig von der Domäne, mit denselben mathematischen Gleichungen beschrieben werden kann.

	$\mathbf{e(t)}$	$\mathbf{f(t)}$	α	β	γ
<i>Elektrische Systeme</i>	$u \text{ (V)}$	$i \text{ (A)}$	L	C	R
<i>Translatorische Systeme</i>	$F \text{ (N)}$	$v \text{ (m/s)}$	m	$\frac{1}{k}$	b
<i>Rotatorische Systeme</i>	$T \text{ (Nm)}$	$\omega \text{ (rad/s)}$	J	$\frac{1}{k}$	b
<i>Hydraulische Systeme</i>	$p \text{ (N/m}^2\text{)}$	$q \text{ (m}^3\text{/s)}$	L_f	C_f	R_f

Tab. 4.3.: Zusammenhang der allgemeingültigen Variablen und den Variablen der verschiedenen Domäne

Überlegungen

Was ergibt eigentlich das Produkt aus $e(t)$ und $f(t)$?

Halten wir uns als Beispiel die elektrische Domäne vor Augen. Hier gilt folgendes:

$$e(t) = u \quad (4.24)$$

$$f(t) = i \quad (4.25)$$

Nun ist sofort ersichtlich, dass das Produkt aus $e(t)$ und $f(t)$ der elektrischen Leistung P entspricht.

$$P = u \cdot i = e(t) \cdot f(t) \quad (4.26)$$

Es ist daher ganz egal welche Domäne betrachtet wird, das Produkt aus Flussvariable $f(t)$ und Potentialvariable $e(t)$ ist immer die Leistung P (W). Die Leistung beschreibt eine bestimmte Menge an Energie, welche in einer gewissen Zeit t aufgebracht wird. Die Leistung kann daher auch in Joule/Sekunde (J/s) angegeben werden. In anderen Worten: Die Leistung entspricht einem Energiefluss.

Fazit

Wenn physikalische Systeme anhand derselben Gleichungen mathematisch beschrieben werden können, so ist auch der Energiefluss P derselbe. Bezogen auf die zuvor beschriebenen Systeme bedeutet das, dass das elektrische System in [Abschnitt 4.1](#) und die mechanischen Systeme in [Abschnitt 4.2, 4.3](#) denselben Energiefluss besitzen.

Ferner gilt, dass alle physikalischen Systeme dem Energieerhaltungssatz unterliegen

welcher besagt, dass Energie in einem *geschlossenen System*¹ erhalten bleibt, was bedeutet, dass diese weder erzeugt noch verbraucht werden kann.

¹Ein geschlossenes System ist ein System das keinerlei Wechselwirkungen mit der Umgebung aufweist.

5. Einführung in die Theorie der Bondgraphen

Mit den allgemeingültigen Beziehungen welche in [Kapitel 4](#) entwickelt wurden und der Tatsache, dass Energieflüsse durch physikalische Systeme anhand dieser Beziehungen modelliert werden können, wird nun eine graphische Modellierungstechnik vorgestellt, welche unter dem Namen *Bondgraphen* (engl. bond graphs) bekannt ist.

Bondgraphen wurden 1959 von H.M. Paynter entwickelt und 1961 erstmals in [[Pay61](#)] publiziert. Weiterentwicklungen wurden u.a. von D.C. Karnopp, D.L. Margolis und R.C. Rosenberg durchgeführt und in [[KMR06](#)] publiziert. Nicht zu vergessen sind Wissenschaftler wie Prof. F. Cellier [[Cel91](#)] und P. Breedveld, deren Forschungen sich u.a. ebenfalls mit Bondgraphen beschäftigen.

Der Grundgedanke der Bondgraphen ist die graphische Darstellung von Energieflüssen welche in einem physikalischen System auftreten. Dabei werden so genannte (*Power-*) *Bonds* (Leistungs-Verbindungen) verwendet, um diese Energieflüsse zu beschreiben.

5.1. Akausale Bondgraphen

Die Bondgraphenmodellierung befasst sich intensiv mit der Erhaltung der Energie in einem physikalischen System.

Energieerhaltung

Grundsätzlich haben alle physikalischen Systeme eines gemeinsam: Energie in einem geschlossenen System bleibt erhalten und kann nur durch drei Mechanismen verändert werden:

- **Speichern:** Energie kann, wie bereits in [Kapitel 4](#) gezeigt, gespeichert werden.
- **Transportieren:** Energie kann durch ein physikalisches System fließen (Energiefluss).
- **Umwandeln:** Energie kann reversibel oder irreversibel umgewandelt werden.

Bondgraphen beruhen auf den oben genannten Mechanismen und stellen Energieflüsse, wie bereits erwähnt, graphisch dar. Energieflüsse selbst sind die Ableitung der Energie bezüglich der Zeit, also Leistung P . Die Energieflüsse P werden in der Bondgraphenmodellierung als halber Pfeil (Bond) dargestellt. Der Energiefluss berechnet sich dabei als Produkt zweier Variablen, der Potentialvariable e (engl. effort, potential) und der Flussvariable f (engl. flow) ¹, welche bereits in [Abschnitt 4.5](#) eingeführt wurden, wobei die Potentialvariable $e(t)$ oberhalb und die Flussvariable $f(t)$ unterhalb der Harpune dargestellt wird (siehe [Abbildung 5.1](#)).

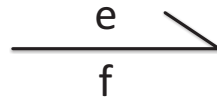


Abb. 5.1.: Gerichtete Harpune (Bond) zur Modellierung des Energieflusses

Ein Bond stellt das Fundament der Bondgraphenmodellierung dar und gibt an in welche Richtung² sich die Leistung ausbreitet. In anderen Worten: Ein Bond ist bereits ein Element welches den Mechanismus „Transportieren von Energie“ der Energieerhaltung verkörpert. Somit kümmern wir uns fortan um die verbleibenden Mechanismen der Energieerhaltung. Im nächsten Schritt wäre es daher sinnvoll energiespeichernde Elemente zu beschreiben. Bevor wir uns diesen Elementen widmen, wird jedoch eine „passives“ Element zur Umwandlung von Energie vorgezogen. In der Literatur wird dieses Element zumeist als so genanntes „verlustbehaftetes“³ Element bezeichnet. Diese Bezeichnung ist im Grunde genommen nur dann korrekt wenn man die Thermodynamik außer acht lässt, sprich wenn die Einflüsse des Wärmeflusses nicht betrachtet werden. Die Energieerhaltung lehrt uns, dass Energie erhalten bleibt, was bedeutet, dass diese nicht verloren gehen kann. Ein verlustbehaftetes Element als solches gibt es also demnach nicht. Die Energie die in diesem Sinne verloren geht wird irreversibel in Wärme umgewandelt. Verlustbehaftete Elemente stellen also nur einen Verlust auf der passiven Seite des Elements dar. Im ersten Schritt wird dieses Element als „verlustbehaftetes“ Element dargestellt.

5.1.1. Verlustbehaftete Elemente (R-Element)

Abbildung 5.2 zeigt den Bondgraphen eines verlustbehafteten Elements.

¹In seltenen Fällen werden sogenannte *Pseudo Bonds* definiert, bei denen das Produkt aus Potential- und Fluss-Variable nicht Leistung ergibt. Da dies allerdings an der ursprünglichen Idee der Bondgraphen vorbei geht, wird auf diese hier nicht weiter eingegangen.

²Die Richtung in welche sich die Leistung ausbreitet wird durch einen halben Pfeil symbolisiert

³Verlustbehaftete Elemente sind in physikalischen Systemen z.B. elektrische Widerstände oder mechanische Dämpfer

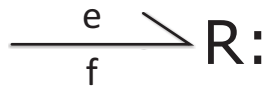


Abb. 5.2.: Passives verlustbehaftetes Element (R-Element)

Wie wir bereits wissen, ist der Zusammenhang zwischen Potential- und Flussvariable bei verlustbehafteten Elementen durch die Beziehung $e(t) = \gamma \cdot f(t)$ (siehe erste Zeile in [Tabelle 4.2](#)) gegeben. Hier tritt der Vorteil von Bondgraphen zum ersten Mal klar zum Vorschein: Es ist völlig egal ob wir einen mechanischen Dämpfer, eine Verengung (Querschnittsreduzierung) in einem Rohr oder einen elektrischen Widerstand modellieren wollen, die dazugehörigen Bondgraphen sind absolut identisch. D.h., dass auch die Energieflüsse identisch sind. Die Bondgraphen eines elektrischen RLC-Schwingkreises und eines mechanischen Feder-Masse Schwingers (Systeme 2. Ordnung) sind daher ebenfalls identisch. Ganz allgemein gilt, dass physikalische Systeme mit derselben mathematischen Beschreibung auch denselben Bondgraphen besitzen (siehe [Abbildung 5.3](#)).

elektrisch	mechanisch	hydraulisch

Abb. 5.3.: Verlustbehaftetes R-Element und dessen Interpretation in den verschiedenen Domänen (vgl. [Tabelle 4.3](#))

Bondgraphen werden aus diesem Grund als *domänenunabhängige* Modellierungstechnik bezeichnet.

5.1.2. Energiespeichernde Elemente

In physikalischen Systemen gibt es exakt zwei grundlegende Arten Energie zu speichern. Zum Einen kann das Potential (zweite Zeile in Tabelle [Tabelle 4.2](#)) und zum Anderen der Fluss (dritte Zeile in [Tabelle 4.2](#)) gespeichert werden. Die Notation der Bondgraphen bezeichnet potentialspeichernde Elemente als I-Elemente (engl. „inertia“) und flussspeichernde Elemente im Gegensatz dazu als C-Elemente (engl. „capacity“).

I-Element

[Abbildung 5.4](#) zeigt den Bondgraphen eines potentialspeichernden Elements.

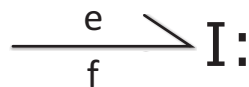


Abb. 5.4.: Passives potentialspeicherndes Element (I-Element)

Der Zusammenhang zwischen Potential- und Flussvariable ist in der dritten Zeile von [Tabelle 4.2](#) dargestellt. Handelt es sich bei der mathematischen Beschreibung unseres physikalischen Systems z.B. um ein elektrisches System so wird $e(t)$, $f(t)$ und die Konstante α mit den entsprechenden Elementen aus [Tabelle 4.3](#) substituiert und wir stellen sofort fest, dass es sich hierbei um eine Induktivität L handelt, welche folgender Beziehung genügt:

$$i = \frac{1}{L} \int u \cdot dt \quad (5.1)$$

Ist das System hingegen mechanisch-translatorischer Herkunft ist sofort ersichtlich, dass das I-Element eine Masse verkörpert und wir erhalten:

$$v = \frac{1}{m} \int F \cdot dt \quad (5.2)$$

[Abbildung 5.5](#) zeigt die Interpretation der I-Elemente in der jeweiligen Domäne.

C-Element

[Abbildung 5.6](#) zeigt ein flussspeicherndes Element.

Der Zusammenhang zwischen Potential- und Flussvariable ist in der zweiten Zeile von [Tabelle 4.2](#) dargestellt. Halten wir uns auch hier wiederum die elektrische Domäne vor

elektrisch	mechanisch	hydraulisch

Abb. 5.5.: Potentialspeicherndes I-Element und dessen Interpretation in den verschiedenen Domänen (vgl. Tabelle 4.3)

$$\begin{array}{c} e \\ \diagdown \\ \hline f \end{array} C:$$

Abb. 5.6.: Passives flussspeicherndes Element (C-Element)

Augen so ist gut ersichtlich, dass das C-Element der elektrischen Kapazität C entspricht, und daher gilt:

$$u = \frac{1}{C} \int i \cdot dt \quad (5.3)$$

Abbildung 5.7 zeigt die Interpretation der C-Elemente in den verschiedenen Domänen.

Nachdem wir nun mit den passiven Komponenten (R-, L- und C-Elemente) physikalischer Systeme vertraut sind, widmen wir uns im nächsten Schritt den aktiven Elementen, den so genannten Quellen.

5.1.3. Aktive Elemente (Quellen)

Im Gegensatz zu den passiven Elementen, welche im Falle eines R-Elements Energie konsumieren (irreversibel in Wärme umwandeln) und im Falle eines I- bzw. C-Elements Energie speichern, stellen Quellen dem System Energie zur Verfügung. An dieser Stelle muss wiederum erwähnt werden, dass Energie weder erzeugt noch verbraucht werden kann. Damit wir jedoch Modelle erstellen können, müssen wir an einer vernünftigen Stelle eine Abgrenzung des Modells oder des zu modellierenden Systems vornehmen.

elektrisch	mechanisch	hydraulisch

Abb. 5.7.: Flussspeicherndes C-Element und dessen Interpretation in den verschiedenen Domänen (vgl. Tabelle 4.3)

Quellen stellen eben eine solche Möglichkeit dar, das System an einer sinnvollen Stelle abgrenzen zu können. So ist es z.B. nicht zielführend bei der Simulation eines elektronischen Schaltkreises das elektrische Verteilernetz inklusive der energieliefernden Kraftwerke zu modellieren⁴. Es ist also hier sinnvoll die Quelle der betrachteten Energieform, in diesem Fall ein Netzgerät oder etwas ähnliches, ausreichend genau zu modellieren. Oft würde hier eine ideale Spannungsquelle ausreichen.

Quellen lassen sich in zwei Arten unterteilen. Es gibt Potential- und Flussquellen. Potentialquellen werden durch das Kürzel *Se* gekennzeichnet, Flussquellen durch *Sf*. Abbildung 5.8 veranschaulicht die beiden Quellen.

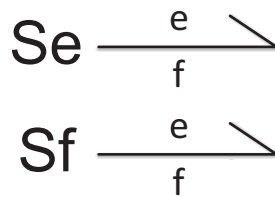


Abb. 5.8.: Aktive Elemente: Oben - Potentialquelle; Unten - Flussquelle

⁴Würde dieser Gedanke weitergeführt werden, müsste man auch die Energiequelle für dieses Kraftwerk (Wasserkraft, Wind, thermische Kraft) modellieren und würde irgendwann bei der Quelle unserer Energie - der Sonne - ankommen.

5.1.4. Verzweigungen

Was uns jetzt noch fehlt, um die physikalischen Systeme aus Kapitel 4 beschreiben zu können, ist die Modellierung der Topologiegleichungen. In anderen Worten: Wir müssen uns noch um das Zusammenspiel der einzelnen Komponenten kümmern. Die Bondgraphenmodellierung bietet hier eine recht einfache und komfortable Lösung, um die Topologiegleichungen korrekt modellieren zu können. Diese werden durch Verzweigungen (engl. junctions) dargestellt.

Bezogen auf *elektrische Systeme* bedeutet das folgendes:

- Bauteile welche in Serie geschaltet sind, durch die derselbe Strom (Fluss) fließt, werden durch eine „1-Junction“ verknüpft. D.h. dass die Kirchhoff'sche Maschenregel angewendet wird welche besagt, dass $\Sigma u = 0$.
- Bauteile welche parallel geschaltet sind, an denen dieselbe Spannung (Potential⁵) anliegt, werden durch eine „0-Junction“ verknüpft. D.h. dass die Kirchhoff'sche Knotenregel verwendet werden muss ($\Sigma i = 0$).

Bei *mechanischen Systemen* kann folgendes gesagt werden:

- Schnittkräfte (engl. cut-forces) werden mittels einer „0-Junction“ modelliert. Damit ist auch vordefiniert, dass nur Feder- und Dämpfer-Elemente an solch eine Verzweigung angeschlossen werden dürfen. Nachdem eine „0-Junction“ angibt, dass $\Sigma v = 0$, bzw. überall dieselbe Kraft wirkt wird zusätzlich klar, dass dies nur für Dämpfer und Federn zutrifft.
- Anders ist das bei Massen (bzw. Trägheiten). Diese können nur an „1-Junctions“ angeschlossen werden, da eine Masse auch nur eine Geschwindigkeit v besitzen kann. Eine „1-Junction“ besitzt nur einen Fluss. Daher gilt, dass $\Sigma F = 0$ was dem so genannten D'Alembert Prinzip entspricht.

Bei *Fluss-Systemen* (hydraulischen Systemen) kann folgendes gesagt werden:

- Handelt es sich um denselben Druck so werden „0-Junctions“ verwendet ($\Sigma q = 0$). Dies trifft beispielsweise für ein Rohr zu, welches in der Mitte einen Tank besitzt.
- Handelt es sich um denselben Fluss so wird das durch eine „1-Junction“ veranschaulicht ($\Sigma p = 0$). Dies trifft für Rohre zu welche an einer Stelle eine Verengung aufweisen.

1-Junction

Abbildung 5.9 veranschaulicht eine „1-Junction“.

⁵Genauer gesagt handelt es sich bei dem Bondgraphen Potential im elektrischen Sinne um das elektrische Potential.

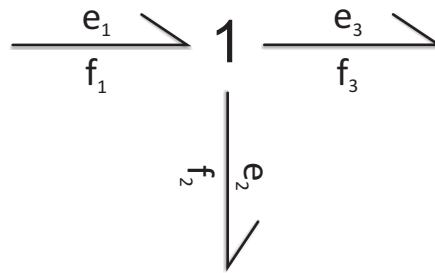


Abb. 5.9.: Serien-Verzweigung: 1-Junction

Die Topologiegleichungen ergeben sich durch die „1-Junction“ automatisch zu:

$$f_1 = f_2 = f_3$$

und

$$\sum_{n=1}^3 e = 0 \rightarrow e_1 - e_2 - e_3 = 0$$

0-Junction

Bei einer „0-Junction“ wie sie in [Abbildung 5.10](#) dargestellt ist, sind die Beziehungen genau umgekehrt.

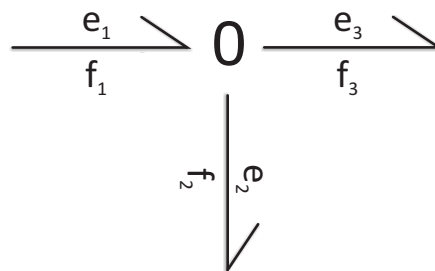


Abb. 5.10.: Parallel-Verzweigung: 0-Junction

Die Topologiegleichungen ergeben sich durch die „0-Junction“ automatisch zu:

$$e_1 = e_2 = e_3$$

und

$$\sum_{n=1}^3 f = 0 \rightarrow f_1 - f_2 - f_3 = 0$$

Vereinfachungen

Welche Aussage hat folgender Bondgraph (Abbildung 5.11)?

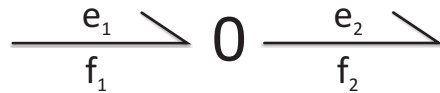


Abb. 5.11.: Zwei Bonds angeschlossen an eine 0-Junction

Ein Bond führt zur Verzweigung hin und einer von dieser weg. Daher gilt:

$$P_{in} = e_1 \cdot f_1 = e_2 \cdot f_2 = P_{out} \quad (5.4)$$

Die zugeführte Leistung entspricht daher exakt der abgeführten Leistung, weshalb diese 0-Junction auch genau so gut weggelassen werden kann, was bedeutet, dass entweder der linke oder der rechte Bond übrig bleibt.

Mittels der in diesem Abschnitt vorgestellten Bondgraph-Elemente sind wir nun in der Lage die bisher vorgestellten Beispiele aus den Abschnitten 3 und 4 zu modellieren.

5.1.5. Beispiele

Im Folgenden werden die Systeme aus Kapitel 4 mittels Bondgraphen modelliert. Anschließend wird gezeigt wie sich Bondgraphen für elektrische und mechanische Systeme sehr strukturiert erstellen lassen. Dazu wird zum Einen die passive elektronische Schaltung aus Kapitel 3 modelliert und zum Anderen ein Feder-Masse Schwinger. Abschließend wird der Bondgraph eines hydraulischen Systems erstellt.

Elektrischer RLC-Schwingkreis

Abbildung 5.12 veranschaulicht die Schaltung des RLC-Schwingkreises.

Der erste Schritt bei der Entwicklung eines Bondgraphen ist die Auswahl der Quelle (Source). Bei einer elektronischen Schaltung ist dieser Punkt in der Regel sehr einfach durchzuführen. Spannungsquellen entsprechen einer *Potentialquelle* (*Se*), Stromquellen hingegen einer *Flussquelle* (*Sf*). Die Spannungsquelle lässt sich demnach wie in

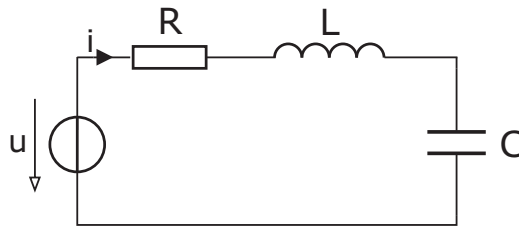


Abb. 5.12.: Schaltung eines elektrischen RLC-Schwingkreises

Abbildung 5.13 modellieren. In obiger Abbildung sind die passiven Komponenten der

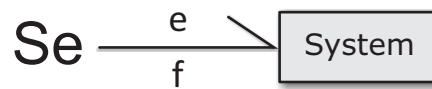


Abb. 5.13.: Modellieren der Spannungsquelle - effort source

Schaltung als System-Block (Black-Box) dargestellt. Im nächsten Schritt gilt es die Verzweigungen zu bestimmen und somit den System-Block durch das vollständige Modell zu ersetzen. Im vorliegenden Beispiel handelt es sich um eine Serienschaltung. Anhand der in [Unterabschnitt 5.1.4](#) vorgestellten Verzweigungen ist gut ersichtlich, dass Bauteile, welche in Serie geschaltet sind, mit einer 1-Junction (Strom = flow als Bezugsgröße) modelliert werden. Die Bauteile werden nun einfach via Bonds an die Verzweigung angeschlossen. Die Bonds zeigen dabei in Richtung der Bauteile. [Abbildung 5.14](#) veranschaulicht den vollständigen Bondgraphen des RLC-Schwingkreises.

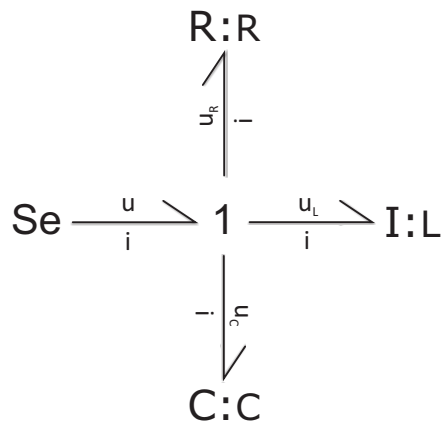


Abb. 5.14.: Bondgraph einer Spannungsquelle mit anschließender Serienschaltung von drei Bauelementen

Dieser Bondgraph stellt dieselbe Information zur Verfügung wie die Differentialgleichung in [Abschnitt 4.1](#) mit dem Unterschied, dass die Gleichungen versteckt sind. In [Abschnitt 5.2](#) wird gezeigt, wie sich die Gleichungen aus dem Bondgraphen extrahieren lassen.

Ferner erhalten wir Information über den Energiefluss durch die elektrische Schaltung, was einen Vorteil gegenüber der Differentialgleichung darstellt. Weiters ist dadurch sehr gut ersichtlich welche Elemente Energie liefern und welche Elemente Energie konsumieren.

Feder-Masse Schwinger

Modellieren wir nun unser erstes mechanisches System. Dazu verwenden wir wiederum das in [Abschnitt 4.2](#) vorgestellte System eines Feder-Masse Schwingers (siehe [Abbildung 5.15](#))

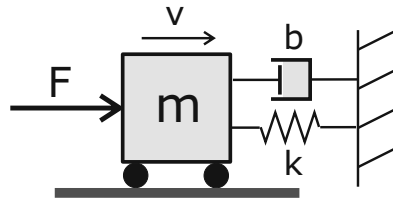


Abb. 5.15.: Feder-Masse Schwinger

Wir starten auch hier wieder mit der Auswahl der Quelle. Unser System wird von einer Kraft F beaufschlagt (fremderregt). In mechanischen Systemen beschreiben Kräfte die Potentialvariablen und Geschwindigkeiten die Flussvariablen. Daraus folgt unmittelbar, dass die Kraft mit einer Potentialquelle Se modelliert wird. Die Kraft wirkt auf die Masse m , welche sich dadurch mit der Geschwindigkeit v zu bewegen beginnt. Anders als bei elektrischen Systemen, bei denen alle Bauteile sowohl an 0-Junctions als auch an 1-Junctions angeschlossen werden können, sind die an die Verzweigungen anzuschließenden Elemente bei mechanischen Systemen fix vorgegeben. Eine Masse besitzt nur eine Geschwindigkeit v (die Geschwindigkeit vor der Masse entspricht der Geschwindigkeit nach der Masse) und kann daher nur an eine 1-Junction angeschlossen werden. [Abbildung 5.16](#) veranschaulicht den ersten Teil des Bondgraphen.

Federn und Dämpfer hingegen, bewirken eine Geschwindigkeitsreduktion, d.h. dass die Geschwindigkeit vor einem Dämpfer bzw. einer Feder, höher ist als danach. Die Kraft aber bleibt dieselbe. Im Bondgraphen wird das durch eine 0-Junction modelliert. Die Feder und der Dämpfer sind auf der oberen Seite fix an die Wand montiert, d.h. dass dort die Geschwindigkeit $v_1 = 0$ ist. Demnach entspricht die Geschwindigkeitsdifferenz an der Feder genau der Geschwindigkeit der Masse. Wir benötigen daher keine 0-Junction

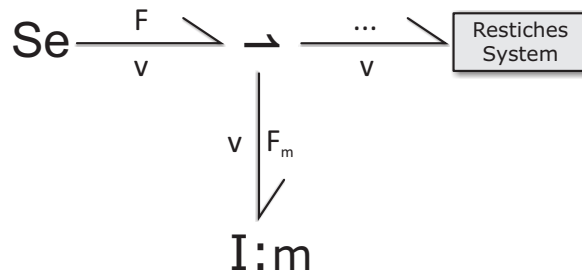


Abb. 5.16.: Erster Teil des Bondgraphen des Feder-Masse Schwingers

um die Geschwindigkeitsdifferenz zu modellieren. Die Feder und der Dämpfer können direkt an die 1-Junction angeschlossen werden, da alle Element dieselbe Geschwindigkeit besitzen. Der Bondgraph des Feder-Masse Schwingers ist in [Abbildung 5.17](#) dargestellt.

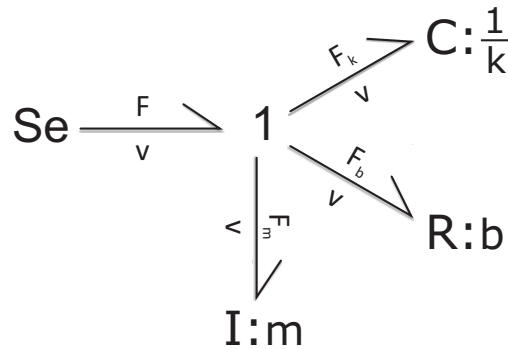


Abb. 5.17.: Bondgraph des Feder-Masse Schwingers

Mechanisch-rotatorisches System 2. Ordnung

In unserem zweiten mechanischen Beispiel wird das in [Abschnitt 4.3](#) modellierte System welches in [Abbildung 5.18](#) dargestellt ist, modelliert.

Anstelle einer Kraft wirkt hier ein Moment T auf die Masse, welche durch deren Trägheit J beschrieben wird. Diese Momentenquelle entspricht wiederum einer Potentialquelle Se . Auf Grund des Moments beginnt sich die Masse mit der Winkelgeschwindigkeit ω zu drehen. Die Masse kann wiederum nur eine Winkelgeschwindigkeit besitzen, was mit einer 1-Junction modelliert wird. [Abbildung 5.19](#) zeigt den Bondgraphen des ersten Teilsystems bestehend aus Potentialquelle, 1-Junction und der Trägheit J .

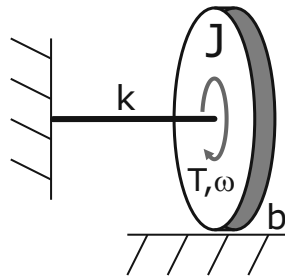


Abb. 5.18.: Mechanisch-rotatorisches System 2. Ordnung

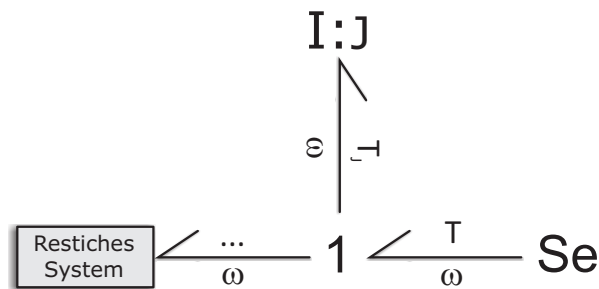


Abb. 5.19.: Erster Teil des Bondgraphen des rotierenden Systems

Die Masse selbst berührt an einer Stelle den Boden was zu einer Reibung b führt. Weiters ist die Masse über eine Welle mit der Torsionssteifigkeit k an einer Wand befestigt. An beiden Wänden ist die Winkelgeschwindigkeit $\omega = 0$, d.h. nachdem das System dadurch nur eine Winkelgeschwindigkeit besitzt, erhalten wir folgenden Bondgraphen (siehe [Abbildung 5.20](#)).

Wie bereits in [Kapitel 4](#) erwähnt wurde, sind die Bondgraphen der eben gezeigten Beispiele identisch.

Fluss-System (hydraulisches System)

Das hydraulische System ist in [Abbildung 5.21](#) dargestellt.

Als Quelle wird eine Potentialquelle Se herangezogen welche sicherstellt, dass ein gewisser Druck p_1 am linken Rohrende herrscht. Die dadurch in Bewegung gesetzte Flüssigkeit trifft auf eine Verengung R_f , welche eine Druckdifferenz hervorruft. Nachdem die Flüssigkeit die Verengung passiert hat, wird diese in einem Behälter mit einer gewissen Kapazität C_f gespeichert. Ferner stellen wir fest, dass die Flüssigkeit im gesamten System denselben Fluss q besitzt. Wir erhalten daher den in [Abbildung 5.22](#) dargestellten Bondgraphen des Systems.

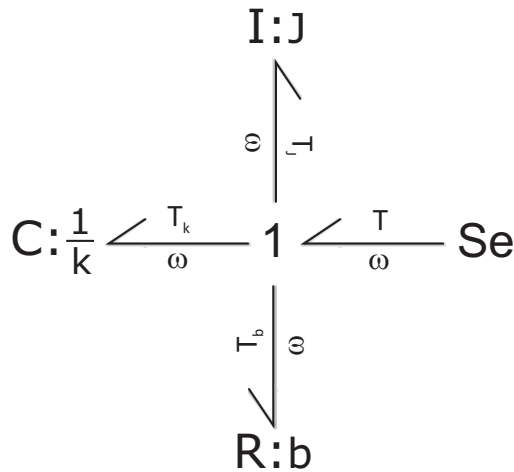


Abb. 5.20.: Bondgraph des rotierenden Systems

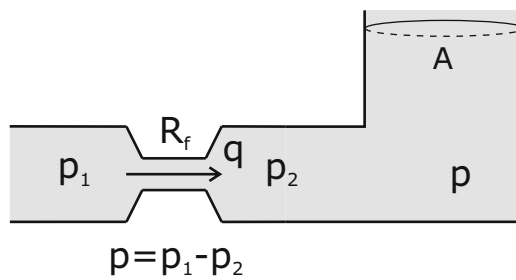


Abb. 5.21.: Hydraulisches System

5.1.6. Lösen von Beispielen mittels standardisierter Lösungsansätze

Im Folgenden wird gezeigt wie sich Bondgraphen für elektrische und mechanische Systeme sehr strukturiert und fehlerfrei erstellen lassen.

Für *elektrische Systeme* gilt:

- Jedes Potential wird durch eine 0-Junction dargestellt.
- Jedes Bauteil wird an eine 1-Junction angeschlossen welche sich zwischen zwei 0-Junctions befindet. Tritt Reibung auf so wird zusätzlich ein R-Element an die Verzweigung angeschlossen.
- Zum Modellieren der Masse (engl. ground) wird eine Potentialquelle welche eine Spannung von $Se = 0$ liefert, verwendet.

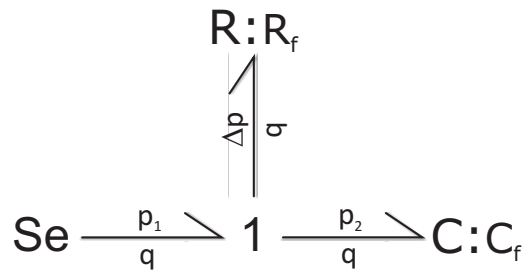


Abb. 5.22.: Bondgraph des hydraulisches System

Für *mechanische Systeme* gilt:

- Jede Masse (Trägheit) wird an eine 1-Junction angeschlossen. Tritt an der Masse zusätzlich Reibung auf, so wird ein R-Element an die 1-Junction angeschlossen.
- Federn und Dämpfer werden an eine 0-Junction angeschlossen, da hier eine Geschwindigkeitsdifferenz auftritt.
- Zum Modellieren von Wänden wird eine Flussquelle welche einen Fluss von $Sf = 0$ liefert, verwendet.

Passive elektronische Schaltung

Im Folgenden wird das System welches bereits in [Kapitel 3](#) vorgestellt wurde, mittels Bondgraphen modelliert. Diesmal wird jedoch ein *standardisierter Lösungsansatz* vorgestellt, mit Hilfe dessen man sehr strukturiert zu einer Lösung gelangt.

Gerade wenn man mit Bondgraphen noch nicht sehr vertraut ist, hat sich für die Modellierung elektrischer Schaltungen ein recht einfaches Verfahren etabliert.

Betrachten wir nochmals unsere Schaltung aus [Kapitel 3](#) ([Abbildung 3.1](#) auf Seite 40) welche in [Abbildung 5.23](#) detaillierter dargestellt ist. Durch Anlegen der Spannung u

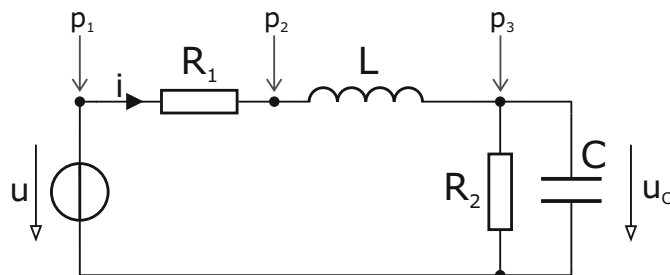


Abb. 5.23.: Passive elektronische Schaltung mit eingezeichneten Potentialen

stellen sich drei unterschiedliche Potentiale (p_1 , p_2 und p_3) ein. Das Potential p_1 ist vor dem Widerstand R_1 , das Potential p_2 stellt sich nach R_1 ein und das Potential p_3 nach der Induktivität L . Jedes dieser Potentiale wird im Bondgraph durch eine 0 -Junction dargestellt. Die Elemente die sich zwischen den jeweiligen 0 -Junctions befinden werden mittels 1 -Junctions dargestellt, welche besagen, dass der Strom der in das Element fließt, dem Strom entspricht welcher aus dem Element fließt. Unsere Schaltung wird daher wie folgt modelliert (Abbildung 5.24).

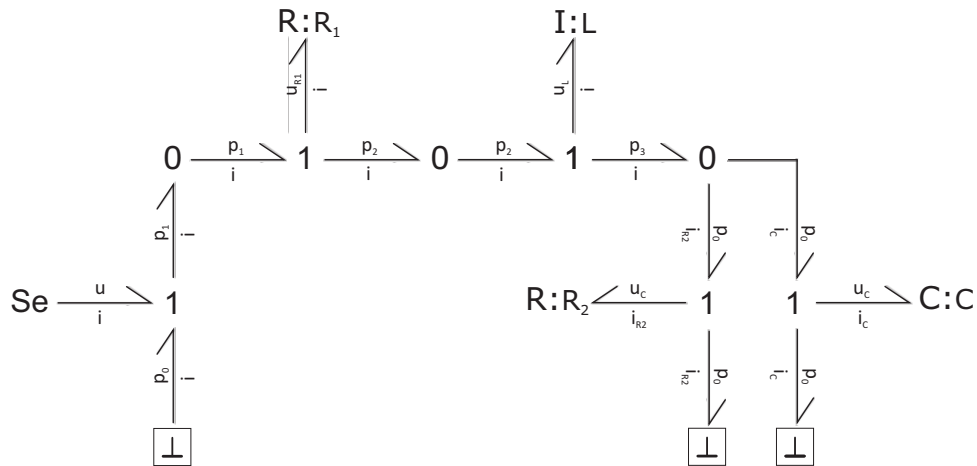


Abb. 5.24.: Bondgraph der elektrischen Schaltung. Anstelle der symbolisch angedeuteten Masse, wäre es besser eine Spannungsquelle $Se = 0$ zu verwenden.

Der Bondgraph der elektrischen Schaltung stellt, wie schon zu Beginn dieses Abschnitts erwähnt wurde, den Energiefluss durch die Schaltung dar. Hält man sich nun den Energiefluss durch die Schaltung gedanklich vor Augen, so kann man den in [Abbildung 5.24](#) erstellten Bondgraph der elektrischen Schaltung graphisch hinterlegen (siehe [Abbildung 5.25](#)). Dieser Schritt ist wiederum sehr gut für Einsteiger in die Thematik der Bondgraphen, da hier sofort ersichtlich ist, wie die einzelnen Komponenten der elektrischen Schaltung im Bondgraph dargestellt werden.

Im nächsten Schritt kann der Bondgraph vereinfacht werden. Das Potential p_0 welches die Masse (engl. ground) repräsentiert, beträgt 0 Volt. Sobald eine der beiden Variablen eines Bonds den Wert 0 aufweist, ist auch die Leistung $P = e \cdot f = 0$, was bedeutet, dass kein Energiefluss stattfindet. Diese Bonds können eliminiert werden (siehe [Abbildung 5.26](#)).

Da zur linken und zur mittleren 0 -Junction sowie zur linken und zu den rechten zwei 1 -Junctions jeweils nur ein Bond hin- und einer wegführt, also $P_{in} = P_{out}$ ist, können diese ebenfalls eliminiert werden. Durch Eliminieren der mittleren 0 -Junction sind zwei 1 -Junctions direkt miteinander verbunden. Nachdem beide denselben Fluss aufweisen,

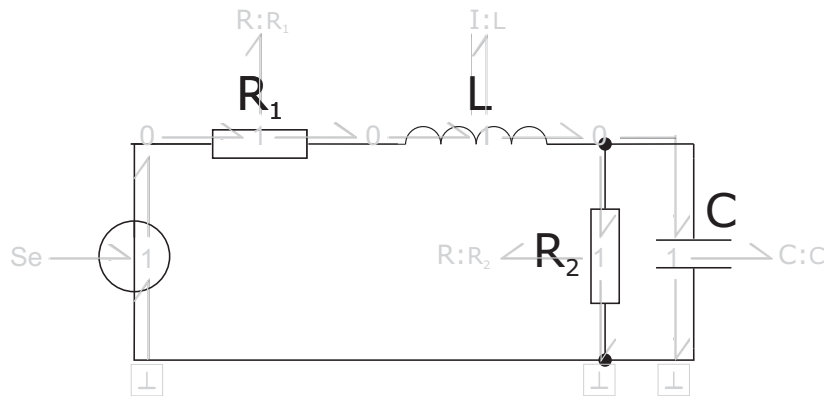


Abb. 5.25.: Elektrische Schaltung mit hinterlegtem Bondgraph

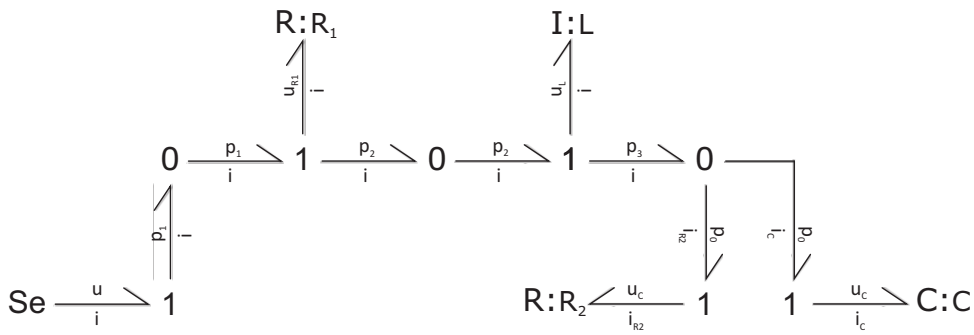


Abb. 5.26.: Reduzierter Bondgraph der elektrischen Schaltung

verschmelzen diese zu einer *1-Junction*. [Abbildung 5.27](#) veranschaulicht den vereinfachten Bondgraph der elektrischen Schaltung.

Feder-Masse Schwinger

Hier wird das Beispiel aus [Abbildung 5.15](#) erneut modelliert. Ausgangspunkt ist der in [Abbildung 5.16](#) dargestellte Bondgraph. Die Kraft F , welche auf die Masse wirkt, setzt diese mit der Geschwindigkeit v in Bewegung. Dadurch werden in der Feder und im Dämpfer Kräfte hervorgerufen. In der Mechanik werden Systeme frei gemacht, um diesen Sachverhalt zu demonstrieren. Dazu werden die Elemente (Feder und Dämpfer) in der Mitte durchgeschnitten, was die Schnittkräfte zum Vorschein bringt. Die Kraft die auf der einen Seite der Feder wirkt, muss dadurch auf der anderen Seite genau in die entgegengesetzte Richtung wirken damit deren Summe Null ergibt (sonst würde ja eine zusätzliche Kraft entstehen). Nachdem das System frei gemacht wurde, müssen

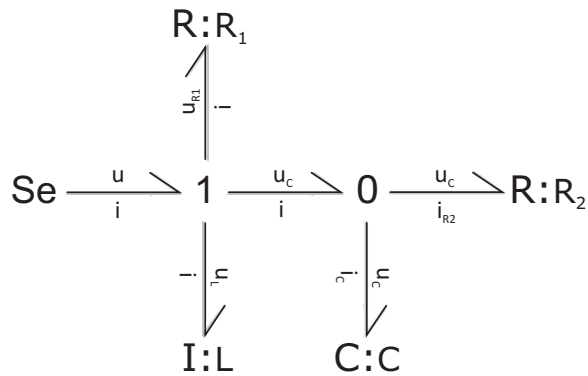


Abb. 5.27.: Vereinfachter Bondgraph der elektrischen Schaltung.

sich die Kräfte welche auf die Masse m wirken zu Null ergeben (siehe Prinzip von D'Alembert, Gleichung (4.5)). Das Prinzip von D'Alembert wird im Bondgraphen durch eine 1-Junction repräsentiert. [Abbildung 5.28](#) veranschaulicht die Schnittkräfte. Die

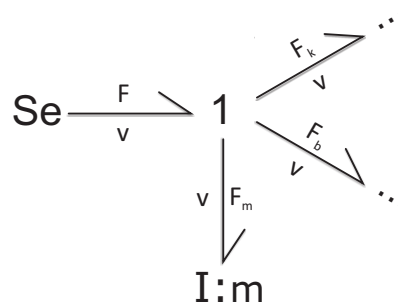


Abb. 5.28.: Anwendung des Prinzips von D'Alembert im Bondgraphen des Feder-Masse Schwingers

Feder und der Dämpfer sind auf der oberen Seite fix an die Wand montiert, d.h. dass dort die Geschwindigkeit $v_1 = 0$ ist. Diese Elemente müssen daher an eine 0-Junction angeschlossen werden. Die Mauer wird dabei durch eine Flussquelle modelliert, welche garantiert, dass kein Fluss auftritt, sprich das System mit $v_1 = 0$ speist. Der vollständige Bondgraph ist in [Abbildung 5.29](#) veranschaulicht.

Die Flussquelle garantiert, dass die Geschwindigkeit Null ist. Dies wird erreicht indem v_1 explizit auf Null gesetzt wird. D.h. aber auch, dass die Leistung gleich Null ist. In anderen Worten: Es findet kein Energiefluss statt. Die beiden Bonds können genauso gut weggelassen werden (siehe [Abbildung 5.30](#)). Damit sind die 0-Junctions ebenfalls überflüssig. Zwei Bonds an einer Verzweigung können immer zu einem Bond zusammengefasst werden, da die zugeführte Leistung der abgeführten Leistung entspricht.

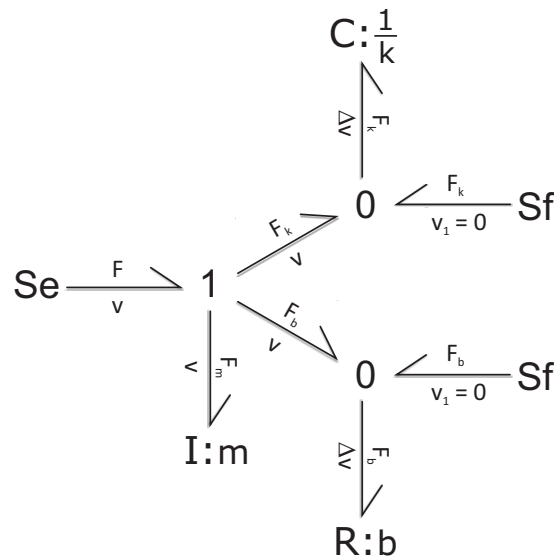


Abb. 5.29.: Bondgraph des Feder-Masse Schwingers

Der vereinfachte vollständige Bondgraph ist in [Abbildung 5.31](#) dargestellt.

Das es sich hier genauso um einen Serienschwingkreis handelt, wie das beim RLC-Schwingkreis der Fall war, ist beim Feder-Masse Schwinger nicht unbedingt ersichtlich.

Mechanisch-rotatorisches System 2. Ordnung

Abschließend wird das Beispiel aus [Abbildung 5.18](#) erneut modelliert. Ausgangspunkt ist der in [Abbildung 5.19](#) dargestellte Bondgraph. Das Moment T , welches auf die Masse wirkt versetzt diese in Rotation, welche durch die Winkelgeschwindigkeit ω beschrieben wird. Dadurch werden in der Welle und zwischen der Masse und dem Boden Momente erzeugt.

Die Masse selbst berührt an einer Stelle den Boden was zu einer Reibung b führt. Weiters ist die Masse über eine Welle mit der Torsionssteifigkeit k an einer Wand befestigt. Die Wand kann wiederum mit einer Flussquelle Sf modelliert werden, wobei die Winkelgeschwindigkeit $\omega = 0$ ist. Eine (Winkel-) Geschwindigkeitsdifferenz wird mittels einer 0-Junction modelliert. [Abbildung 5.32](#) veranschaulicht den vollständigen Bondgraphen des rotatorischen Systems.

Der Bondgraph aus [Abbildung 5.32](#) kann wiederum vereinfacht werden. Da beide Flussquellen garantieren, dass $\omega = 0$ ist, findet kein Energiefluss statt. Diese Bonds können daher weggelassen werden. Die zwei 0-Junction besitzen nun jeweils noch zwei Bonds.

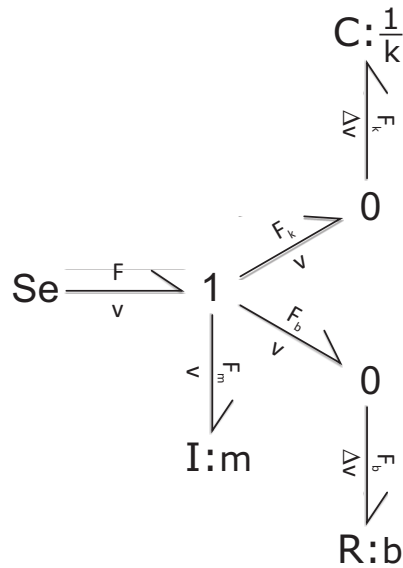


Abb. 5.30.: Vereinfachter Bondgraph des Feder-Masse Schwingers

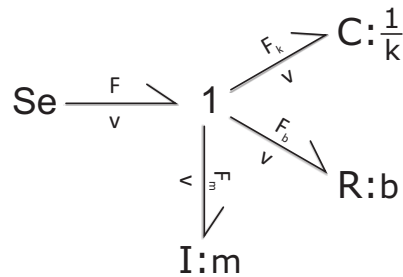


Abb. 5.31.: Vollständig vereinfachter Bondgraph des Feder-Masse Schwingers

D.h., dass die zugeführte Leistung der abgeführten Leistung entspricht (siehe [Abbildung 5.11](#)). Wir erhalten folgenden vereinfachten Bondgraphen (siehe [Abbildung 5.33](#)).

5.2. Kausale Bondgraphen - Kausale Analyse

Bis jetzt wurden die Komponenten- sowie die Topologiegleichungen mittels Bondgraphen modelliert. Anschließend wurden erste Bondgraphen einfacher physikalischer Systeme erstellt. Natürlich handelt es sich bei solchen Modellen bereits um korrekte und simulierbare Modelle. Beispielsweise lassen sich die erstellten Bondgraphen mittels Dy-

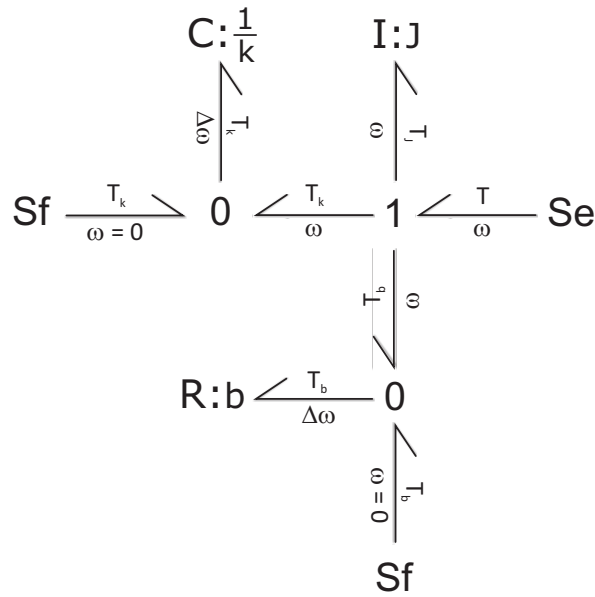


Abb. 5.32.: Bondgraph des rotierenden Systems

mola und der frei verfügbaren BondLib⁶ implementieren. Auch andere Tools wie z.B. 20sim von [Bro99] übernehmen das Kausalisieren der Gleichungen für den Benutzer. Ist man hingegen nicht im Besitz einer Software, welche die Implementierung von Bondgraphen zulässt, so gewinnen die kausalen Bondgraphen an Bedeutung. Sobald der Bondgraph kausalisiert wurde, gelangt man nämlich sehr leicht zu einem äquivalenten Blockschaltbild oder der Zustandsraumdarstellung. Des Weiteren ist es für den Modellierer ein Vorteil, wenn die Kausalität bekannt ist, da dadurch ein noch tieferer Einblick ins System gegeben ist.

Abbildung 5.34 veranschaulicht die zwei Möglichkeiten einen Bond zu kausalisieren.

Wie wir bereits wissen, definiert jeder Bond zwei Variablen. Oberhalb des Bonds steht immer die Potentialvariable $e(t)$ und unterhalb des Bonds die Flussvariable $f(t)$. Infolgedessen benötigen wir auch zwei Gleichungen, um Werte für $e(t)$ und $f(t)$ zu berechnen.

Definition

Der Kausalitätsbalken (vertikaler Querstrich) gibt immer an auf welcher Seite der Fluss berechnet wird. In anderen Worten: Dort wo sich der Kausalitätsbalken befindet ist

⁶Open-source Library für Dymola/Modelica mit der Bondgraphen erstellt werden können. <https://modelica.org/libraries>

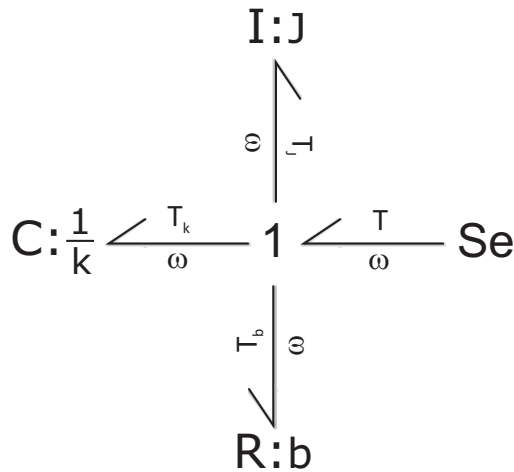


Abb. 5.33.: Vereinfachter Bondgraph des rotierenden Systems

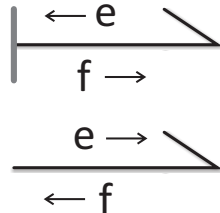


Abb. 5.34.: Kausaler Bond mit dazugehöriger Signalausbreitung (durch Pfeile symbolisiert)

immer der Fluss der Output⁷ (Flussausbreitung führt weg vom Kausalitätsbalken) und das Potential e der Input (Potentialausbreitung führt zum Kausalitätsbalken).

Es ist dabei wichtig zu verstehen, dass die Kausalität keine physikalischen Eigenschaften des Systems beschreibt. Sie legt fest welche Gleichungen für die Bestimmung der nötigen Variablen verwendet werden. Es entspricht also der in [Abschnitt 3.4](#) erwähnten horizontalen Sortierung welche in [Abschnitt 6.9](#) genauer vorgestellt wird.

Befindet sich der Kausalitätsbalken also beim jeweiligen Element (egal ob das Element passiver oder aktiver Natur ist), so ist immer der Fluss der Output und das Potential der Input. Befindet sich der Kausalitätsbalken jedoch auf der anderen Seite des Elements so ist das Potential der Output und demnach der Fluss der Input (siehe [Abbildung 5.35](#)).

⁷Der Output einer Gleichung stellt die Unbekannte dar, welche in dieser Gleichung berechnet wird. In anderen Worten: Die Gleichung wird zu einer Zuweisung.

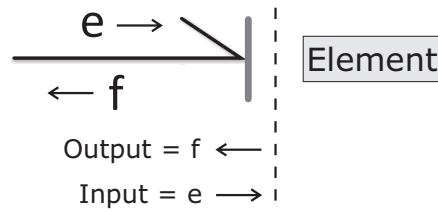


Abb. 5.35.: Beispiel: Der Kausalitätsbalken befindet sich auf der Seite des Elements. Aus der Sicht des Elements (in Richtung strichlierter Linie) ist der Fluss der Output und das Potential der Input.

Ein Bond gibt an in welche Richtung sich die Leistung ausbreitet. Der Kausalitätsbalken hingegen gibt an in Welche Richtung sich die Potential-Information bzw. die Fluss-Information ausbreitet.

5.2.1. Kausalisierung der Quellen

Bei einer Potentialquelle berechnet die Quelle das Potential, d.h. dass sich der Kausalitätsbalken auf der Seite der Harpune befinden muss (siehe [Abbildung 5.36](#)). In anderen Worten: Bei einer elektrischen Spannungsquelle wird die Spannung von der Quelle geliefert. Daher breitet sich die Potential-Information in dieselbe Richtung wie die Leistung aus.

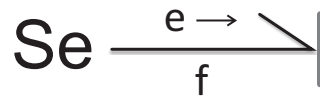


Abb. 5.36.: Kausale Potentialquelle - Signalausbreitung der Potentialvariable $e(t)$ zeigt in Richtung des Kausalitätsbalkens (die Quelle versorgt das System mit Spannung).

Eine Flussquelle verhältet sich genau umgekehrt. Hier befindet sich der Kausalitätsbalken auf der Seite der Quelle (siehe [Abbildung 5.27](#)).

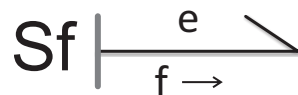


Abb. 5.37.: Kausale Flussquelle - Signalausbreitung der Flussvariable $f(t)$ führt weg vom Kausalitätsbalken (die Quelle versorgt das System mit Strom).

5.2.2. Kausalisierung der passiven Elemente

(Kausales) R-Element

Die Kausalität von R-Elementen ist frei (siehe [Abbildung 5.38](#)).

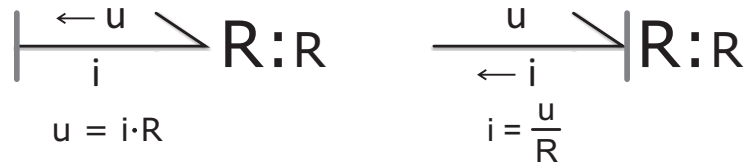


Abb. 5.38.: Kausales R-Element am Beispiel eines elektrischen Widerstandes. Links: Der Kausalitätsbalken befindet sich auf der gegenüberliegenden Seite des Elements, daher ist hier die Spannung u der Output und der Strom i der Input. Rechts: Der Kausalitätsbalken befindet sich auf der Seite des Elements. Daher ist i der Output und u der Input.

Je nachdem welche Variable den Output bzw. den Input des Widerstandes repräsentiert, wird der Kausalitätsbalken des R-Elements entsprechend gewählt. Dies hängt im Allgemeinen von der Topologie, welche via 0- und 1-Junctions beschrieben wird, ab.

(Kausales) I-Element

Bei energiespeichernden Elementen ergibt sich die Kausalität nach dem Wunsch Integratoren anstatt Differentiatoren zu wählen. Die Gründe dafür wurden bereits in [Unterabschnitt 2.2.2](#) erläutert. Sobald ein physikalisches System simuliert wird, wird der Integrator durch ein so genanntes numerisches Integrationsverfahren approximiert (siehe [Kapitel 7](#)). Zu Beginn der Simulation muss daher ein Anfangswert der Zustandsgröße bekannt sein. Dieser wird auch als Anfangsbedingung bezeichnet. Der nächste Wert der Zustandsgröße wird dann ein Zeitintervall (Schrittweite) h später berechnet. Dieser Wert ergibt sich aus dem Anfangswert und dem Modell des Systems. Würde man anstatt eines Integrators hingegen einen Differentiator verwenden, so müssten Werte in der Zukunft bekannt sein, was nicht ohne weiteres möglich ist. Aus diesem Grund und in Anbetracht der Tatsache, dass es physikalisch gesehen keine (idealen) Differentiatoren gibt, werden, wann immer es möglich ist, Integratoren verwendet. In komplexeren Modellen kommt es jedoch auch vor, dass aufgrund der Topologie nicht jedes energiespeichernde Element mittels Integratoren beschrieben werden kann. Tritt solch ein Fall auf, so spricht man von einer sogenannten *strukturellen Singularität* (siehe [Abschnitt 5.7.3](#)).

Bei einer Induktivität L welche der Gleichung

$$u_L = L \cdot \frac{di_L}{dt} \quad (5.5)$$

genügt, wird wie folgt vorgegangen:

Man forme Gleichung (5.5) so um, dass der Output (Zustandsvariable) auf der linken Seite des Gleichheitszeichens steht. Dadurch dass der Output klar definiert ist, wird aus der Gleichung unmittelbar eine Zuweisung, was wiederum durch den Zuweisungsoperator " := " gekennzeichnet wird:

$$i_L := \frac{1}{L} \int u_L \cdot dt \quad (5.6)$$

Nun erstelle man das dazugehörige Blockschaltbild (siehe [Abbildung 5.39](#)).

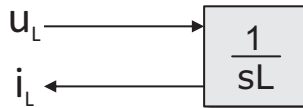


Abb. 5.39.: Blockschaltbild der kausalen Induktivität

Aus dem Blockschaltbild ist sofort ersichtlich, dass die Spannung u_L den Input und der Strom i_L den Output darstellt. Bezogen auf Bondgraphen bedeutet das wiederum, dass der Kausalitätsbalken auf der Seite der Harpune des I-Elements stehen muss (siehe [Abbildung 5.40](#)).

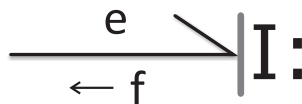


Abb. 5.40.: Kausalisiertes I-Element. Im Falle einer Induktivität wird α mit L substituiert

(Kausales) C-Element

Eine Kapazität C wird durch die Gleichung

$$i_c = C \cdot \frac{du_C}{dt} \quad (5.7)$$

beschrieben. Werden nun dieselben Schritte wie beim I-Element durchgeführt so ergibt sich die Kausalität des Elements wie in [Abbildung 5.41](#) dargestellt.



Abb. 5.41.: Kausalisiertes C-Element (links) mit dazugehörigem Blockschaltbild (rechts). Die Konstante β wird bei einer Kapazität mit C substituiert.

5.2.3. Kausalisierung der Verzweigungen

1-Junction

Wie in [Unterabschnitt 5.1.4](#) bereits beschrieben wurde, besagt eine 1-Junction, dass der Fluss aller Bonds welche zur Verzweigung hinführen oder von der Verzweigung wegführen identisch ist. Es gilt:

$$f_1 = f_2 = f_3 \quad (5.8)$$

Daraus folgt, dass die Summe der Potentiale gleich 0 sein muss $\sum e = 0$.

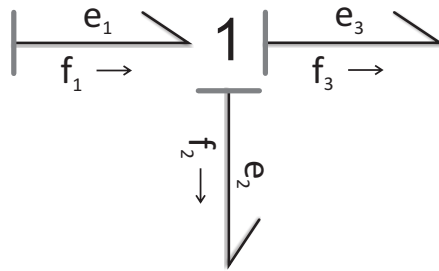
$$e_1 - e_2 - e_3 = 0 \quad (5.9)$$

Nachdem in einer 1-Junction die Flussvariablen aller beteiligten Bonds gleich sind, kann der Fluss nur aus einer Gleichung berechnet werden. Wäre das nicht der Fall, so wäre das Gleichheitszeichen in (5.8) schlichtweg falsch. An der Stelle an welcher der Fluss berechnet wird, also den Flow-Input in die Verzweigung darstellt, muss gleichzeitig das Potential den Output darstellen.

Eine 1-Junction besitzt also eine Potentialgleichung welche exakt einen Output liefert. Im vorliegenden Beispiel ([Abbildung 5.42](#)) muss das Potential e_1 den Output darstellen. Die akasale Gleichung (5.9) wird zu folgender kausaler Gleichung:

$$e_1 := e_2 + e_3 \quad (5.10)$$

Da eine 1-Junction denselben Fluss aufweist, kann dieser nur an einem Bond berechnet werden, sprich den Flow-Input in die Verzweigung darstellen. Demnach muss eine solche Verzweigung genau $(n - 1)$ -Kausalitätsbalken besitzen.

Abb. 5.42.: Kausalisierte 1-Junction. Der Fluss f_1 stellt den Input dar.

Beispiel

Als Beispiel wird die Serienschaltung einer Stromquelle und zweier Widerstände herangezogen (Abbildung 5.43)

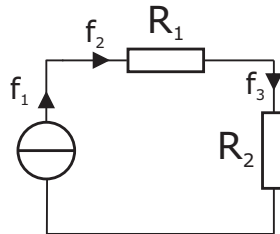


Abb. 5.43.: Stromquelle mit zwei Serienwiderständen

Die Stromquelle wird mittels einer Flussquelle Sf modelliert. Da durch alle Elemente derselbe Strom fließt ($f_1 = f_2 = f_3$), wird eine 1-junction verwendet, um dies zu modellieren. [Abbildung 5.44](#) zeigt den dazugehörigen Bondgraphen der elektrischen Schaltung. Hier ist klar ersichtlich warum bei einer 1-junction auch nur ein Bond für den Fluss verantwortlich ist, sprich diesen berechnen muss. Es kann nicht mehr als eine Stromquelle geben, welche den Fluss berechnet, d.h. die Schaltung mit diesem versorgt. Fügt man gedanklich eine zweite Stromquelle hinzu, so stellt man schnell fest, dass es einen Konflikt gibt. Die Bedingung der 1-Junction, welche besagt, dass $(n - 1)$ Kausalitätsbalken innerhalb der Verzweigung sein müssen, ist somit nicht mehr erfüllbar.

0-Junction

Anders ist dies bei einer 0-Junction. Diese weist genau eine Flussgleichung auf ($f_1 - f_2 - f_3 = 0$), welche ebenfalls genau einen Output liefert. Nachdem in einer 0-Junction

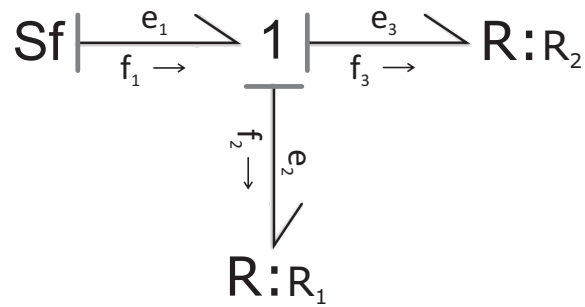


Abb. 5.44.: Kausaler Bondgraph der Stromquelle mit zwei Serienwiderständen

alle Potentiale identisch sind ($e_1 = e_2 = e_3$) kann dieses ebenfalls nur an einer Stelle berechnet (geliefert) werden. Daher besitzen diese Verzweigungen auch nur einen Kausalitätsbalken (siehe [Abbildung 5.45](#)). Die 0-Junction muss also genau einen Kausalitätsbalken besitzen.

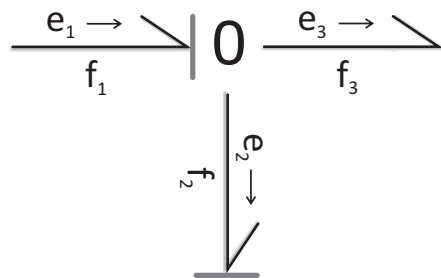


Abb. 5.45.: Kausalisierte 0-Junction. Das Potential e_1 stellt den Input dar.

Aus [Abbildung 5.45](#) ist sofort ersichtlich, dass der Fluss f_1 den Output darstellt und sich zu

$$f_1 := f_2 + f_3$$

ergibt.

Nun sind wir bereits in der Lage, das zuvor modellierte (akausale) System zu kausalisieren. Nachdem wir diesen Schritt durchgeführt haben, können wir die Gleichungen, welche den jeweiligen Elementen zugrunde liegen, aus dem Bondgraphen extrahieren, um so das äquivalente Zustandsraummodell bzw. Blockschaltbild zu erhalten.

5.2.4. Modellierung der passiven elektrischen Schaltung mit kausalen Bondgraphen

Anleitung zur Kausalisierung von Bondgraphen

- (1) Kausalität der Quellen bestimmen. Dieser Schritt wird immer als erstes durchgeführt, da diese Kausalitäten „fix“ vorgegeben sind (engl. fixed causality).
- (2.1) Als nächstes wird immer die Kausalität der energiespeichernden Elemente bestimmt. Diese Kausalitäten ergeben sich nach dem Wunsch Integratoren anstatt Differentiatoren zu verwenden. Man spricht von *bevorzugter Kausalität* (engl. preferred causality).
- (2.2) Kausalitätsbalken der Verzweigungen einzeichnen. Nachdem die Kausalität der Quellen bzw. der energiespeichernden Elemente eingezeichnet wurde, wird die Kausalität der Verzweigungen bestimmt. Hier gilt, dass 1-Junctions ($n - 1$)-Kausalitätsbalken und 0-Junctions einen Kausalitätsbalken besitzen. Da Verzweigungen die Topologie des Systems repräsentieren wird die Kausalität auch durch diese bestimmt. Man spricht von einer *erzwungenen Kausalität* (engl. constrained causality).
- (3) Kausalität der Widerstände einzeichnen. Diese sollte sich durch die Kausalisierung der Verzweigungen automatisch ergeben. Es kann dennoch vorkommen, dass der Kausalitätsbalken entweder direkt beim Element oder gegenüber des Elements eingezeichnet werden kann. In solch einem Fall ist die Kausalität des Widerstandes nicht durch die Topologie des Systems (Verzweigungen) bestimmt. Es tritt eine algebraische Schleife auf. Man spricht hier von einer *freien Kausalität* (engl. indifferent causality).

Kausalisierung des Bondgraphen der elektrischen Schaltung

Im Folgenden wird der akausale Bondgraph der elektrischen Schaltung schrittweise kausalisiert. Wir werden uns dabei strikt an die Anleitung zur Kausalisierung von Bondgraphen halten und mit der Kausalisierung der Quelle beginnen (siehe [Abbildung 5.46](#)).

[Abbildung 5.47](#) beinhaltet den zweiten Schritt, die Kausalisierung der energiespeichernden Elemente.

Nun sind nur noch die zwei Widerstände R_1 und R_2 übrig. Wir werden also im nächsten Schritt die Verzweigungen kausalisieren. Der Kausalitätsbalken der Induktivität welche sich an der 1-Junction befindet ist bereits außerhalb der Verzweigung. Deshalb müssen alle anderen Kausalitätsbalken innerhalb dieser Verzweigung sein. Damit ist die Kausalität des Widerstandes R_1 auch schon definiert worden. Die 0-Junction hat bereits

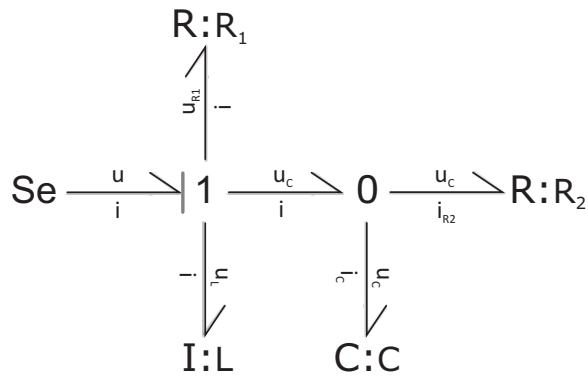


Abb. 5.46.: Kausalisieren des Bondgraphen der elektrischen Schaltung: (1) Kausalität der Quelle bestimmen.

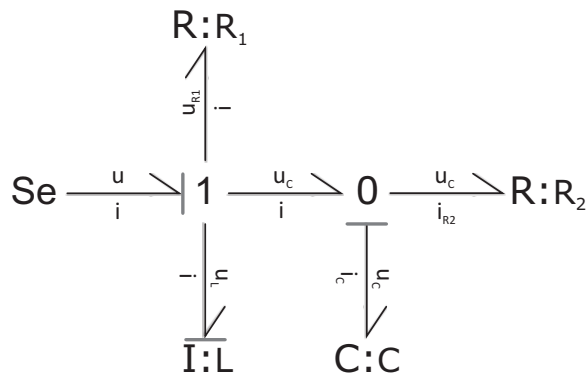


Abb. 5.47.: Kausalisieren des Bondgraphen der elektrischen Schaltung: (2.1) Kausalisierung der energiespeichernden Elemente

einen Kausalitätsbalken innerhalb der Verzweigung. D.h., dass sich alle anderen Kausalitätsbalken außerhalb der Verzweigung befinden müssen. Damit ist auch die Kausalität des zweiten Widerstandes R_2 definiert. Wir erhalten den kausalen Bondgraphen in [Abbildung 5.48](#).

Nachdem Schritt (3) nicht durchgeführt werden musste, wissen wir sofort, dass keine algebraischen Schleifen auftreten. Somit kann die Zustandsraumdarstellung ohne Probleme erstellt werden.

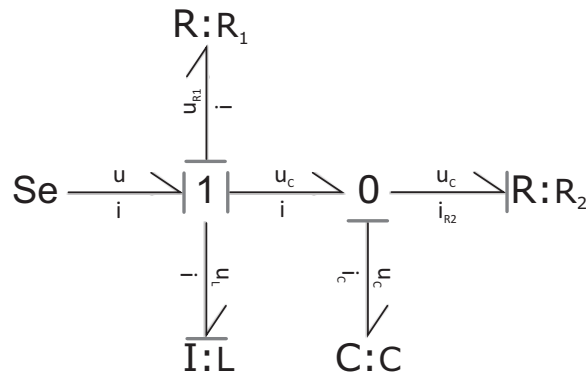


Abb. 5.48.: Kausalisieren des Bondgraphen der elektrischen Schaltung: (2.2) Kausalisierung der Verzweigungen

5.3. Aufstellen der Gleichungen anhand des Bondgraphen

Folgende Gleichungen lassen sich anhand des Bondgraphen sofort ermitteln (Tabelle 5.1):

Element	ohne Kausalität	mit Kausalität
1 :	$u - u_{R_1} - u_C - u_L = 0$	$u_L := u - u_{R_1} - u_C$
0 :	$i - i_C - i_{R_2} = 0$	$i_C := i - i_{R_2}$
$R : R_1$	$u_{R_1} = i \cdot R_1$	$u_{R_1} := i \cdot R_1$
$C : C$	$u_C = \frac{1}{C} \int i_c \cdot dt$	$u_C := \frac{1}{C} \int i_c \cdot dt = x_1$
$R : R_2$	$u_C = i_{R_2} \cdot R_2$	$i_{R_2} := \frac{1}{R_2} \cdot u_C$
$I : L$	$i = \frac{1}{L} \int u_L \cdot dt$	$i := \frac{1}{L} \int u_L \cdot dt = x_2$

Tab. 5.1.: Gleichungen ermittelt anhand des Bondgraphen.

5.3.1. Zustandsraumdarstellung

Wie bereits im Abschnitt zuvor, wird wiederum die Zustandsraumdarstellung des Systems erstellt. Diesmal wird die Zustandsraumdarstellung direkt aus den kausalen Gleichungen, die aus dem Bondgraph ermittelt wurden, erstellt. Die Zustandsvariablen sind wiederum die Spannung am Kondensator C ($u_C = x_1$) und der Strom durch die

Induktivität L ($i = x_2$).

Erste Zustandsgleichung

Hierfür wird die Gleichung in der vierten Zeile und zweiten Spalte von Tabelle 5.1 abgeleitet und wir erhalten:

$$\dot{x}_1 = \frac{1}{C} i_C \quad (5.11)$$

Nun wird i_C mit der dazugehörigen kausalen Gleichung beschrieben, welche in diesem Fall in Zeile 2, Spalte 2 von Tabelle 5.1 steht. Die Variablen werden nun solange substituiert bis nur noch Eingangsgrößen und Zustandsvariablen vorhanden sind.

$$\begin{aligned} \dot{x}_1 &= \frac{1}{C} (i - i_{R_2}) = \frac{1}{C} (x_2 - i_{R_2}) = \\ &= \frac{1}{C} \left(x_2 - \frac{1}{R_2} \cdot u_C \right) = \frac{1}{C} \left(x_2 - \frac{1}{R_2} \cdot x_1 \right) = \\ &= -\frac{1}{C} \cdot \frac{1}{R_2} \cdot x_1 + \frac{1}{C} \cdot x_2 \end{aligned} \quad (5.12)$$

Zweite Zustandsgleichung

Hierfür wird die Gleichung in der letzten Zeile und zweiten Spalte von Tabelle 5.1 abgeleitet:

$$\dot{x}_2 = \frac{1}{L} u_L \quad (5.13)$$

Auch hier wird wiederum solange substituiert bis nur noch bekannte Größen (Eingangsgrößen und Zustandsgrößen) vorhanden sind.

$$\begin{aligned} \dot{x}_2 &= \frac{1}{L} (u - u_{R_1} - u_C) = \frac{1}{L} (u - i \cdot R_1 - x_1) = \\ &= \frac{1}{L} (u - R_1 \cdot x_2 - x_1) = \\ &= -\frac{1}{L} \cdot x_1 - \frac{R_1}{L} \cdot x_2 + \frac{1}{L} \cdot u \end{aligned} \quad (5.14)$$

Bilden der Zustandsraumdarstellung

Der Vollständigkeit halber werden nochmals beide Zustandsgleichungen angeführt.

$$\begin{aligned}\dot{x}_1 &= -\frac{1}{C} \cdot \frac{1}{R_2} \cdot x_1 + \frac{1}{C} \cdot x_2 \\ \dot{x}_2 &= -\frac{1}{L} \cdot x_1 - \frac{R_1}{L} \cdot x_2 + \frac{1}{L} \cdot u\end{aligned}$$

In Matrixschreibweise erhalten wir (wie bereits im Abschnitt zuvor):

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -\frac{1}{C \cdot R_2} & \frac{1}{C} \\ -\frac{1}{L} & -\frac{R_1}{L} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{L} \end{pmatrix} \cdot u \quad (5.15)$$

$$y = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (5.16)$$

5.3.2. Blockschaltbild

Das Blockschaltbild wird direkt aus den kausalen Gleichungen (Zuweisungen) gebildet. Das Ergebnis ist in [Abbildung 3.4](#) dargestellt.

5.4. Weitere Bondgraph-Elemente: Energieumwandlung

Bisher haben wir das Speichern und das Transportieren von Energie behandelt. Ersteres wird durch C- und I-Elemente beschrieben. Letzteres durch den Bond selbst. Das R-Element wurde bis jetzt als „verlustbehaftetes“ Element behandelt, was vollkommen legitim ist, solange wir die Thermodynamik außer acht lassen.

Im Folgenden werden drei zusätzliche Elemente, welche sich mit dem Umwandeln von Energie beschäftigen, eingeführt. Während Widerstandsquellen die irreversible Umwandlung freier Energie in Wärme beschreiben, werden Transformatoren und Gyratoren verwendet, um reversible Energieumwandlungsvorgänge zwischen gleichartigen oder verschiedenartigen Energieformen zu beschreiben.

5.4.1. Widerstandsquellen

Bei der Widerstandsquelle wird die Energie (z.B. elektrische Leistung) irreversibel in Wärme umgewandelt (siehe [Abbildung 5.49](#)).

Die Widerstandsquelle ist demnach eine Quelle von Entropie was durch den zusätzlichen Buchstaben S (engl. Source) gekennzeichnet wird. Der Wärmefluss $\dot{Q} = \frac{dQ}{dt}$ (thermi-



Abb. 5.49.: Widerstandsquelle: Die elektrische Leistung $P = u \cdot i$ wird irreversibel in Wärme umgewandelt

sche Leistung) setzt sich aus dem Produkt Temperatur T und Entropiefluss $\dot{S} = \frac{dS}{dt}$ zusammen. Die Kausalität des Bonds auf der thermischen Seite ist immer so, dass der Widerstand dort als Quelle von Entropie gesehen wird, nie aber als Quelle von Temperatur. Temperaturquellen gibt es physikalisch betrachtet nicht.

5.4.2. Transformatoren

Bei Transformatoren werden Potential- und Flussvariablen über ein Übersetzungsverhältnis m folgendermaßen umgewandelt (siehe [Abbildung 5.50](#)).



Abb. 5.50.: Bondgraph eines Transformators

Die Potentialvariablen sind über m folgendermaßen verknüpft:

$$e_1 = m \cdot e_2 \quad (5.17)$$

Da ein Bond einen „verlustlosen“ Energietransport darstellt, gilt aufgrund der Energieerhaltung:

$$e_1 \cdot f_1 = e_2 \cdot f_2 \quad (5.18)$$

setzt man Gleichung (5.17) in (5.18) ein so erhält man die Verknüpfung der Flussvariablen:

$$f_2 = m \cdot f_1 \quad (5.19)$$

Für eine Übersetzungsverhältnis $m > 1$ gilt daher, dass die Potentialvariable e_1 größer als e_2 wird. Daraus folgt unmittelbar dass die Flussvariable f_1 kleiner als f_2 wird.

Beispiele für Transformatoren sind elektrische Transformatoren, mechanische Getriebe oder Kolben in einem hydraulischen System (siehe [Abbildung 5.51](#)).

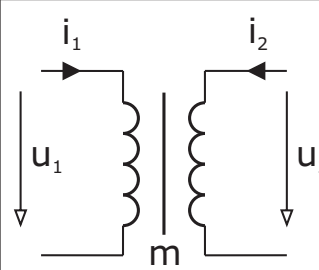
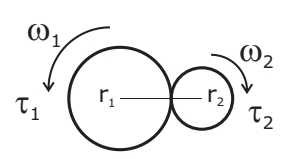
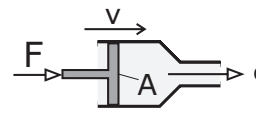
elektrisch	mechanisch	hydraulisch
		
$u_2 = m \cdot u_1$ $i_1 = m \cdot i_2$	$\tau_2 = (r_2/r_1) \cdot \tau_1$ $\omega_1 = (r_2/r_1) \cdot \omega_2$	$F = A \cdot p$ $q = A \cdot v$

Abb. 5.51.: Beispiele für Transformatoren

Kausalisierung des Transformators

Da wir jeweils eine Gleichung haben welche zwei Potential- oder Flussvariablen verknüpft, muss beim Transformator eine Potentialvariable und eine Flussvariable berechnet werden. Es ergeben sich daher, in Abhängigkeit der Topologie, folgende zwei Möglichkeiten (Abbildung 5.52 und 5.53).



Abb. 5.52.: Kausalisierung des Transformators

Aus Abbildung 5.52 erhält man folgende Gleichungen:

$$e_1 := m \cdot e_2 \tag{5.20}$$

$$f_2 := m \cdot f_1 \tag{5.21}$$



Abb. 5.53.: Kausalisierung des Transformators

Aus [Abbildung 5.53](#) erhalten wir:

$$e_2 := \frac{1}{m} \cdot e_1 \tag{5.22}$$

$$f_1 := \frac{1}{m} \cdot f_2 \tag{5.23}$$

5.4.3. Gyratoren

Ein Gyrtator verknüpft Potentialvariablen mit Flussvariablen über das Übersetzungsverhältnis r (siehe [Abbildung 5.54](#)).



Abb. 5.54.: Bondgraph eines Gyrtators

$$e_1 = r \cdot f_2 \tag{5.24}$$

Nachdem auch hier die aufgenommene Leistung der abgegebenen Leistung entspricht (siehe Gleichung (5.18)), ergibt sich die zweite dazugehörige Gleichung durch

$$e_2 = r \cdot f_1 \tag{5.25}$$

Ein Beispiel für einen Gyrtator ist ein DC-Motor. Hier erzeugt der Ankerstrom i_a ein Drehmoment τ_m (siehe [Abbildung 5.55](#)).

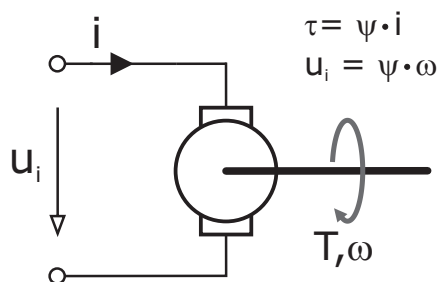


Abb. 5.55.: Beispiel für einen Gyrtator: DC-Motor

Kausalisierung des Gyrotors

Da wir die eine Gleichung links und die andere rechts vom Gyrotor berechnen müssen, können wir die Gleichungen entweder nach den beiden Potentialvariablen oder aber nach den beiden Flussvariablen auflösen.

In anderen Worten: Der Fluss f_2 ergibt sich direkt aus dem Übersetzungsverhältnis und e_1 . Daher muss für den Fall, dass e_1 der Input ist, f_2 der Output sein, da dieser über Gleichung (5.24) festgelegt ist.

Es ergeben sich wiederum zwei Möglichkeiten:



Abb. 5.56.: Kausalisierung des Gyrotors

Aus [Abbildung 5.56](#) erhält man folgende Gleichungen:

$$e_1 := r \cdot f_2 \quad (5.26)$$

$$e_2 := r \cdot f_1 \quad (5.27)$$



Abb. 5.57.: Kausalisierung des Gyrotors

Aus [Abbildung 5.57](#) erhalten wir:

$$f_2 := \frac{1}{r} \cdot e_1 \quad (5.28)$$

$$f_1 := \frac{1}{r} \cdot e_2 \quad (5.29)$$

5.5. Kausale Pfade

Kausale Pfade stellen eine alternative Möglichkeit dar, um sehr einfach und strukturiert zur Zustandsraumdarstellung zu gelangen. Bisher wurden immer alle akausalen Gleichungen aus dem akausalen Bondgraphen extrahiert und anhand des kausalen Bondgraphen im nächsten Schritt kausalisiert. Anhand der kausalen Gleichungen wurde dann systematisch die Zustandsraumdarstellung erstellt (siehe [Abschnitt 5.3](#)). Um nicht alle

akausalen Gleichungen aus dem Bondgraphen extrahieren zu müssen, kann man sogenannten kausalen Pfaden folgen. Durch diesen Schritt werden die unbekanntenen Größen sukzessive durch bekannte Größen ersetzt. Wird dies für jede Gleichung welche ein energiespeicherndes Element beschreibt durchgeführt, gelangt man so sehr strukturiert zur Zustandsraumdarstellung. Dies wird anhand eines DC-Motors demonstriert, welcher in [Abbildung 5.58](#) dargestellt ist. Der kausale Bondgraph ist in [Abbildung 5.59](#) veran-

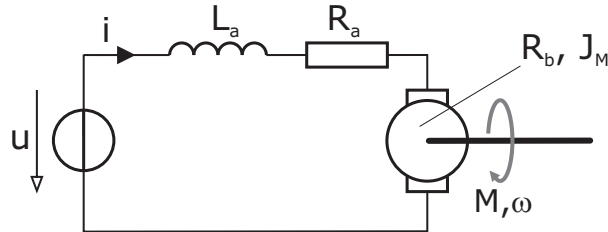


Abb. 5.58.: DC-Motor mit Ankerinduktivität L_a Kupferverlusten R_a , Trägheit des Rotors J_M und Lagerreibung R_b

schaulich. Um nun die dazugehörige Zustandsraumdarstellung zu erhalten benötigen

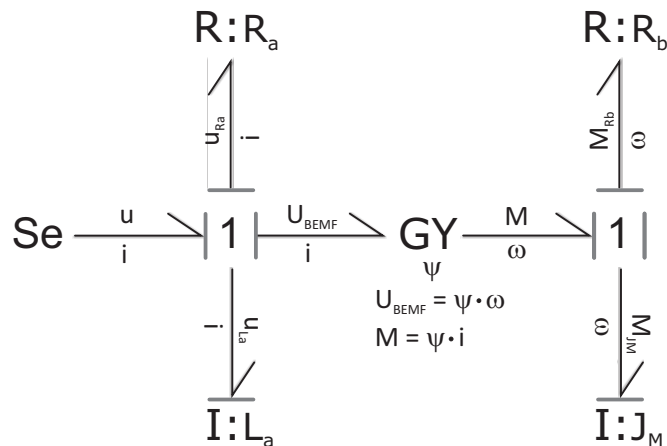


Abb. 5.59.: Kausaler Bondgraph des DC-Motors mit Ankerinduktivität L_a , Kupferverlusten R_a , Trägheit des Rotors J_M und Lagerreibung R_b

wir die Gleichungen welche die energiespeichernden Elemente beschreiben.

$$i = \frac{1}{L_a} \int u_{L_a} \cdot dt = x_1 \quad (5.30)$$

$$\omega = \frac{1}{J_M} \int M_{J_M} \cdot dt = x_2 \quad (5.31)$$

Starten wir mit Gleichung (5.30) welche die Ankerinduktivität beschreibt. Durch Bilden der zeitlichen Ableitung erhalten wir:

$$\dot{x}_1 = \frac{1}{L_a} \cdot u_{L_a} \tag{5.32}$$

Nun werden die Unbekannten auf der rechten Seite des Gleichheitszeichens so lange substituiert, bis nur noch bekannte Größen, sprich Zustandsvariablen und Eingangsgrößen vorhanden sind. Es werden demnach genau dieselben Schritte durchgeführt wie bisher. Prinzipiell ist das richtig, nur das diesmal ein etwas anderer Lösungsweg gezeigt wird. Analysieren wir die Erstellung der ersten Zustandsgleichung, welche aus Gleichung (5.32) - durch Substitution der Unbekannten - erstellt wird, nun anhand des Bondgraphen. Dafür zeichnen wir im ersten Schritt für jeden Bond explizit dessen Ein- und Ausgangsgröße ein (siehe Abbildung 5.60). Im nächsten Schritt gilt es u_{L_a}

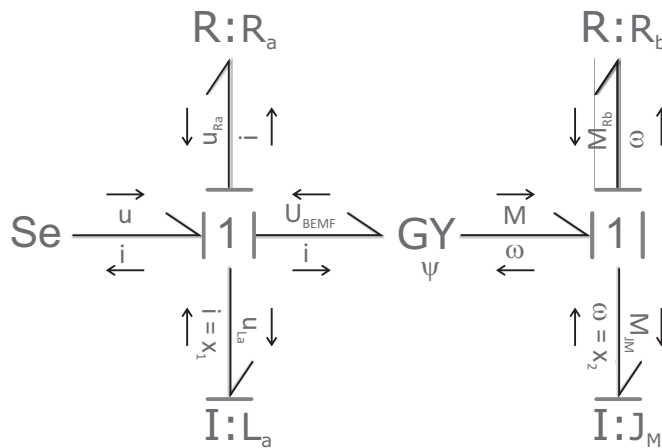


Abb. 5.60.: Kausaler Bondgraph des DC-Motor: Signalfussrichtungen der Variablen sind durch Pfeile dargestellt

zu substituieren. Nachdem u_{L_a} die Ausgangsspannung der 1-Junction darstellt (Signalfussrichtung wird durch Pfeile visualisiert), also die unbekannte Größe ist, welche aus der akausalen Gleichung ($u - u_{R_a} - u_{BEMF} - u_{L_a} = 0$) berechnet wird, folgt

$$\dot{x}_1 = \frac{1}{L_a} \cdot (u - u_{R_a} - u_{BEMF}) \tag{5.33}$$

Dieser Schritt ist noch gut nachvollziehbar und konnte zudem ohne kausale Pfade gelöst werden. Zur Erinnerung: Die 1-Junction ist eine graphische Darstellung der Kirchhoff'schen Knotenregel.

Im nächsten Schritt ersetzen wir die Spannung u_{R_a} durch den algebraische Ausdruck $i \cdot R_a$. Für die induzierte Spannung u_{BEMF} ist dieser Schritt allerdings nicht mehr so

trivial. Unter Betrachtung des in [Abbildung 5.61](#) eingezeichneten kausalen Pfades wird jedoch sehr gut ersichtlich wie u_{BEMF} substituiert werden muss, um die vollständige Zustandsgleichung zu erhalten. Folgt man dem kausalen Pfad so wird u_{BEMF} zuerst

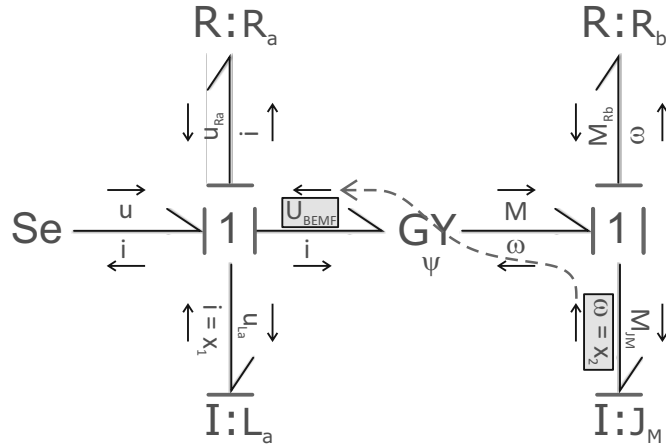


Abb. 5.61.: Kausaler Bondgraph des DC-Motor: Visualisierung des kausalen Pfades, um die induzierte Spannung durch eine Zustandsvariable auszudrücken.

durch $\psi \cdot \omega$ substituiert. Da die Variable ω zum Einen eine Flussvariable einer 1-Junctin und zum Anderen eine Zustandsvariable ist, sind wir am Ende des kausalen Pfades angelangt und erhalten folgende Zustandsgleichung:

$$\dot{x}_1 = \frac{1}{L_a} \cdot (u - i \cdot R_a - \psi \cdot \omega) = \frac{1}{L_a} \cdot (u - x_1 \cdot R_a - \psi \cdot x_2) \quad (5.34)$$

Nun kann die zweite Zustandsgleichung ermittelt werden. Dazu bilden wir die zeitliche Ableitung von Gleichung (5.31).

$$\dot{x}_2 = \frac{1}{J_M} \cdot M_{JM} = \frac{1}{J_M} \cdot (M - M_{R_b}) \quad (5.35)$$

Der zweite Term auf der rechten Seite des Gleichheitszeichens wird durch den algebraischen Ausdruck $R_b \cdot \omega$ ersetzt. Für den ersten Term müssen wir wiederum dem kausalen Pfad im Bondgraph folgen (siehe [Abbildung 5.62](#)).

Es ist sofort ersichtlich, dass das Moment M durch den Ausdruck $\psi \cdot i$ zu ersetzen ist. Wir erhalten:

$$\dot{x}_2 = \frac{1}{J_M} \cdot (\psi \cdot i - R_b \cdot \omega) \quad (5.36)$$

Um die zweite Zustandsgleichung zu erhalten, werden die Zustandsgrößen i und ω durch

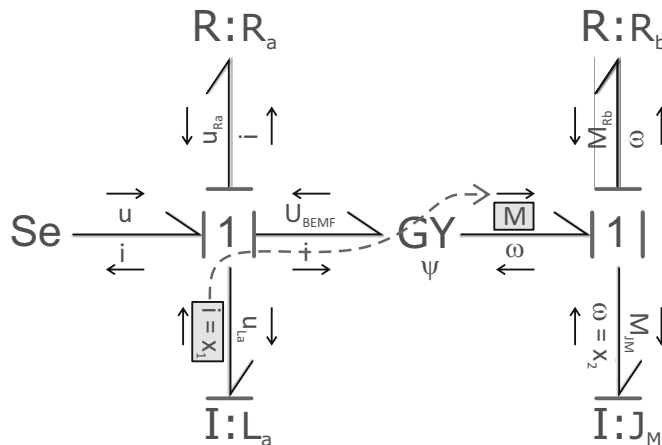


Abb. 5.62.: Kausaler Bondgraph des DC-Motor: Visualisierung des kausalen Pfades, um das Moment welches auf den Rotor wirkt durch eine Zustandsvariable auszudrücken.

x_1 und x_2 ersetzt.

$$\dot{x}_2 = \frac{1}{J_M} \cdot (\psi \cdot x_1 - R_b \cdot x_2) \quad (5.37)$$

5.6. Das Dualitätsprinzip - Duale Bondgraphen

Es existiert zu jedem Bondgraphen ein sogenannter *dualer* Bondgraph. Es ist immer möglich einen Bondgraphen in sein duales Äquivalent umzuwandeln indem alle Potential- und Flussvariablen vertauscht werden. Wenn ein Bondgraph in sein duales Äquivalent umgewandelt wird, ist weiters Folgendes zu beachten:

- Flussquellen werden in Potentialquellen umgewandelt, Kapazitäten werden zu Induktivitäten, Widerstände zu Leitwerten und umgekehrt.
- Transformatoren und Gyrotoren ändern sich nicht aber deren Übersetzungsverhältnisse werden invertiert.
- Ein 1-Junction wird zu einer 0-Junction und umgekehrt.
- Die Kausalitätsbalken wechseln die Seite des Bonds

Im Folgenden werden zwei Beispiele vorgestellt die das Prinzip dualer Bondgraphen erläutern. Das erste Beispiel ist die Dualisierung einer kompletten elektronischen Schaltung. Im darauf folgenden Beispiel soll lediglich die Induktivität der Schaltung dualisiert werden.

Abbildung 5.63 veranschaulicht den kausalen Bondgraphen einer elektrischen Schaltung sowie den dazugehörigen dualen Bondgraphen.

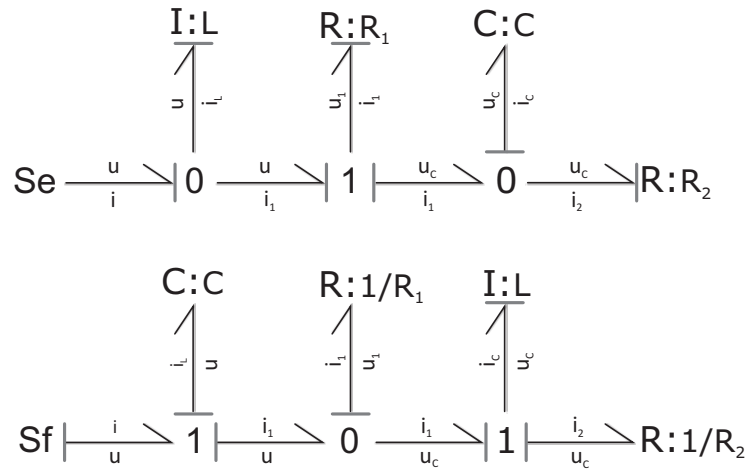


Abb. 5.63.: Duale Bondgraphen. Oben: Kausaler Bondgraph einer elektrischen Schaltung. Unten: Dazugehöriger dualer Bondgraph

Teilweise ist es erforderlich nur bestimmter Teile eines Modells zu dualisieren. Im Folgenden wird die Kapazität C durch ihr duales Äquivalent ersetzt. Nachdem jetzt nur für dieses Element die Potential- und die Flussvariable vertauscht werden müssen benötigen wir ein weiteres Element, welches diesen Schritt ermöglicht. Hierfür eignet sich ein Gyrtor mit einem Übersetzungsverhältnis von $r = 1$. Der daraus resultierende Bondgraph ist in [Abbildung 5.64](#) dargestellt.

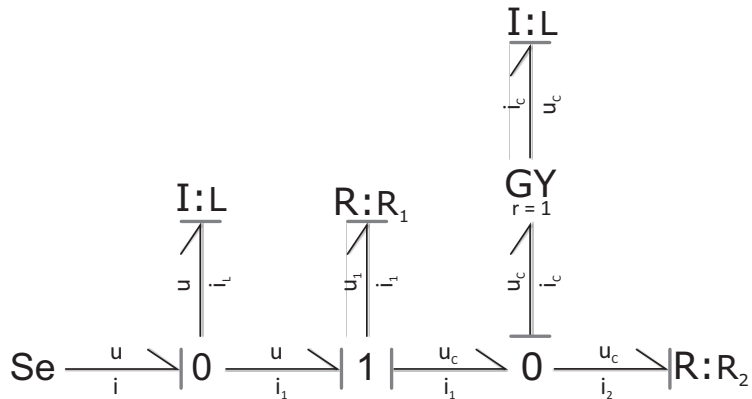


Abb. 5.64.: Duale Bondgraphen. Bondgraph einer elektrischen Schaltung in der lediglich die Kapazität C durch ihr duales Pendant ersetzt wurde.

5.7. Kausalitätskonflikte in Bondgraphen - Kausale Analyse

Nicht immer verläuft der Prozess der Kausalisierung von Bondgraphen so reibungslos wie bisher. Oft kommt es vor, dass sogenannte Kausalitätskonflikte auftreten, welche eine korrekte Platzierung des Kausalitätsbalkens nicht gestatten.

5.7.1. Einblicke in das Modell mittels kausaler Analyse

Bevor wir mit einem konkreten Beispiel beginnen, sollten wir uns jedoch noch ein paar Gedanken über die kausale Analyse mathematischer Modelle physikalischer Systeme machen. Dazu wird die Anleitung zum kausalisieren von Bondgraphen, welche in Abschnitt 5.2.4 vorgestellt wurde, herangezogen.

Es soll im Folgenden diskutiert werden, welche Art von Konflikten beim Kausalisieren von Bondgraphen auftreten können.

- Ist der Bondgraph nach Schritt (1) der Anleitung zur Kausalisierung von Bondgraphen kausalisiert, so handelt es sich um ein statisches System, d.h. es treten keinerlei dynamische Vorgänge auf. Das System hat also keine energiespeichernden Elemente welche via Differentialgleichungen beschrieben werden und besteht daher ausschließlich aus algebraischen Gleichungen. Tritt hier ein Konflikt beim Kausalisieren auf, so wurden zu viele aktive Elemente verwendet. Es werden z.B. zwei Spannungsquellen parallel geschaltet, sprich zwei Potentialquellen an eine 0-Junction angeschlossen. Da an einer 0-Junction nur ein Element für das Potential verantwortlich sein kann, sprich dieses produziert, tritt ein Konflikt auf. Das Modell ist nicht korrekt und muss geändert (modifiziert) werden.
- Tritt ein Konflikt beim Kausalisieren der energiespeichernden Elemente, also nach Schritt (2.1) auf, so spricht man von einer *strukturellen Singularität*. Es handelt sich um ein sogenanntes abhängiges Element, bei welchem die Anfangsbedingung daher nicht unabhängig gewählt werden kann, d.h. zusätzlich von einer anderen Anfangsbedingung abhängt. Dieses Thema wird ausführlich in [Unterabschnitt 5.7.3](#) behandelt. Oft treten solche Kausalitätskonflikte auf wenn Vereinfachungen im Modell getroffen werden. Seile werden beispielsweise als steif modelliert. Dadurch reduziert sich die Ordnung des Systems, d.h. es wird auf eine Differentialgleichung verzichtet, welche die Elastizität des Seils beschreibt.
- Die Kausalität der Widerstände ergibt sich in den meisten Fällen automatisch aufgrund der Topologie (siehe Schritt (2.2)). Sobald Schritt (3) erforderlich ist, die Kausalität des R-Elements also frei wählbar ist, handelt es sich um eine algebraische Schleife (siehe [Unterabschnitt 5.7.2](#)).

Im den folgenden Abschnitten wird gezeigt, wie Kausalitätskonflikte gehandhabt werden. Was hier wiederum sehr gut ersichtlich wird, ist die Berechtigung der Bondgra-

phenmodellierung. Durch das Kausalisieren des Bondgraphen können bereits zu einem frühen Zeitpunkt Fehler im erstellten Modell identifiziert werden.

Natürlich funktioniert dies auch ohne Bondgraphen anhand der Gleichungen welche das System beschreiben. Dieses Thema wird in [Abschnitt 6.9](#) ausführlich behandelt. Hier werden Algorithmen vorgestellt, um die Gleichungen zu kausalisieren, sodass die Zustandsraumdarstellung erstellt werden kann. Ferner wird gezeigt, wie algebraische Schleifen und strukturelle Singularitäten anhand diverser Algorithmen eliminiert werden. Sobald algebraische Schleifen oder strukturelle Singularitäten jedoch von Hand eliminiert werden, ist dieser Schritt sehr aufwändig und fehlerbehaftet.

Nichtsdestotrotz bieten Bondgraphen eine sehr bequeme Art der Modellierung und der damit verbundenen Optimierung des Modells anhand der kausalen Analyse.

5.7.2. Algebraische Schleifen in Bondgraphen

[Abbildung 5.65](#) zeigt eine elektrische Schaltung welche eine algebraische Schleife beinhaltet. Was das genau bedeutet, wird anhand des dazugehörigen Bondgraphen verdeut-

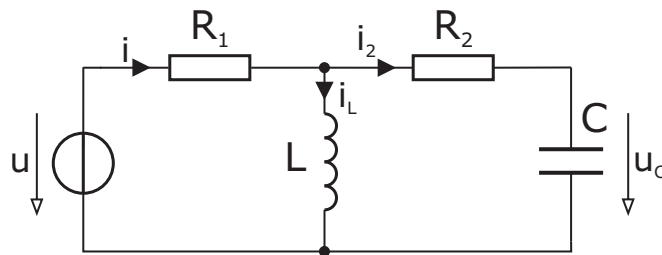


Abb. 5.65.: Elektrische Schaltung welche eine algebraische Schleife aufweist.

licht, welcher in [Abbildung 5.66](#) dargestellt ist. Kausalisieren wir den Bondgraphen nun

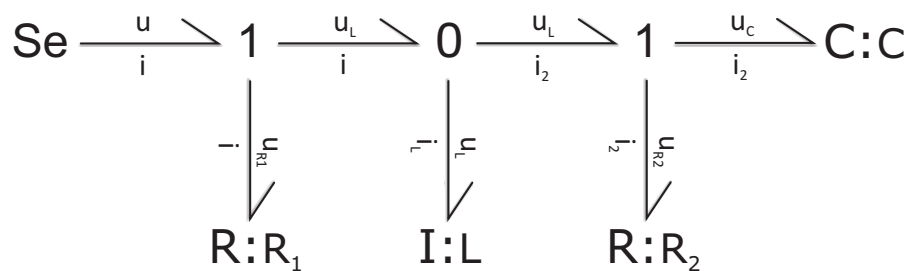


Abb. 5.66.: Akausaler Bondgraph der elektrischen Schaltung welche eine algebraische Schleife aufweist.

wiederum Schritt für Schritt. Wir starten mit der Kausalisierung der Spannungsquelle.

Anschließend kausalisieren wir die energiespeichernden Elemente. Nach den ersten zwei Schritten erhalten wir den in [Abbildung 5.67](#) dargestellten Bondgraphen.

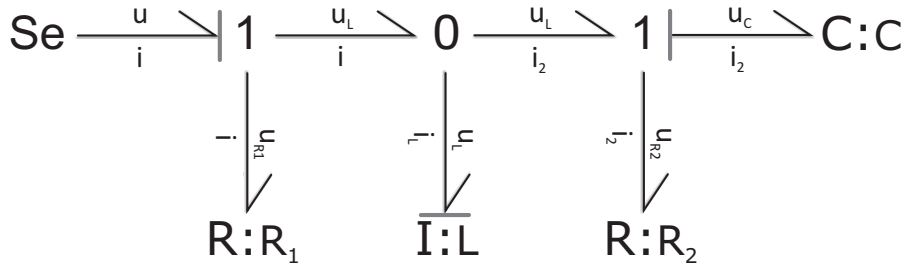


Abb. 5.67.: Teilweise kausalisierter Bondgraph der elektrischen Schaltung welche eine algebraische Schleife aufweist.

Bisher gab es keinerlei Probleme bei der Kausalisierung des Bondgraphen. Im nächsten Schritt, werden die Kausalitätsbalken der Verzweigungen eingezeichnet. Dazu bedarf es einer kurzen Diskussion: Beide 1-Junctions besitzen schon einen Kausalitätsbalken. Da jede der beiden 1-Junctions drei Bonds miteinander verknüpft, muss jeweils noch ein weiterer Kausalitätsbalken innerhalb der Verzweigung sein. Da aber auch an beiden 1-Junctions ein Widerstand ist, kann dort der Kausalitätsbalken jeweils an beiden Seiten eingezeichnet werden. Wir haben demnach freie Wahl. Sobald wir an einem Widerstand (R-Element) frei wählen können wo sich der Kausalitätsbalken befindet, wissen wir sofort, dass es sich um eine algebraische Schleife handelt. In unserem Fall ist die Lösung dieses Problems recht einfach in den Griff zu bekommen. Da wir frei wählen können wo sich der Kausalitätsbalken der Widerstände befinden soll, starten wir entweder mit der Kausalisierung von R_1 oder aber mit R_2 . In anderen Worten: Wir zeichnen den Kausalitätsbalken des Widerstandes R_1 beliebig ein. Je nachdem wo wir den Kausalitätsbalken einzeichnen, erhalten wir unterschiedlich kausalisierte Bondgraphen. Wir zeichnen den Kausalitätsbalken im ersten Schritt innerhalb der linken 1-Junction ein und erhalten folgenden Bondgraphen ([Abbildung 5.68](#)).

Was bedeutet dieser Schritt mathematisch?

Die implizite Gleichung

$$u_{R_1} - i \cdot R_1 = 0 \tag{5.38}$$

wurde damit kausalisiert. In anderen Worten: Wir haben angenommen, dass u_{R_1} zu berechnen ist, sprich der Strom i bekannt ist. Daher wird die Gleichung (5.38) zu

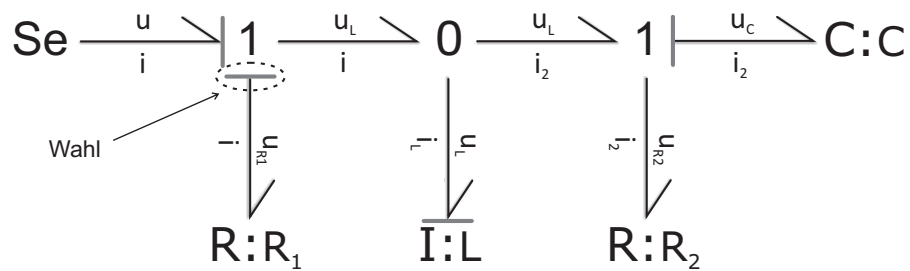


Abb. 5.68.: Teilweise kausalisierter Bondgraph der elektrischen Schaltung welche eine algebraische Schleife aufweist. Zur Lösung der algebraischen Schleife wurde der Kausalitätsbalken des Widerstandes R_1 , welcher frei wählbar ist, innerhalb der linken 1-Junction eingezeichnet.

folgender Zuweisung.

$$u_{R_1} = i \cdot R_1 \tag{5.39}$$

In Kapitel 7, Unterabschnitt 6.9.2 wird dieser Schritt als „tearing“ bezeichnet.

Die restlichen Kausalitätsbalken ergeben sich nun automatisch. Da die linke 1-Junction jetzt 2 Kausalitätsbalken besitzt ($n - 1 = 3 - 1 = 2$), muss der Kausalitätsbalken des letzten Bonds auf der anderen Seite angebracht werden. Damit ist die Kausalität der 0-Junctions ebenfalls definiert. Im selben Zug ist nun auch klar, dass der Bond, welcher sich zwischen der 0-Junction und der rechten 1-Junction befindet, den Kausalitätsbalken auf der rechten Seite haben muss. Da die rechte 1-Junction keine Kausalitätsbalken mehr benötigt, wird dieser auf der unteren Seite des Widerstandes R_2 eingezeichnet. Der vollständig kausalisierte Bondgraph ist in (Abbildung 5.69) dargestellt.

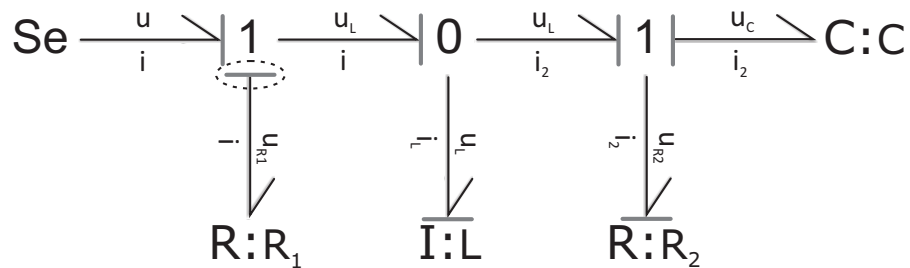


Abb. 5.69.: Kausalisierter Bondgraph der elektrischen Schaltung welche eine algebraische Schleife aufweist. Zur Lösung der algebraischen Schleife wurde der Kausalitätsbalken des Widerstandes R_1 , welcher frei wählbar ist, innerhalb der linken 1-Junction eingezeichnet.

Wie schon erwähnt, gibt es eine weitere Möglichkeit den Bondgraphen zu kausalisieren indem der Kausalitätsbalken des Widerstandes R_1 nicht innerhalb der linken 1-Junction sondern auf der Seite des Widerstands selbst, angebracht wird (siehe [Abbildung 5.70](#)).

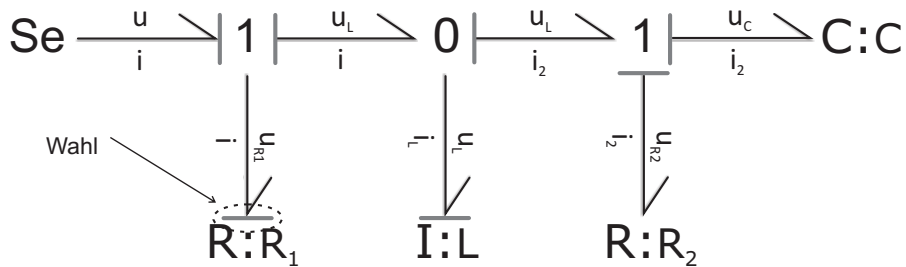


Abb. 5.70.: Kausalisierter Bondgraph der elektrischen Schaltung welche eine algebraische Schleife aufweist. Zur Lösung der algebraischen Schleife wurde der Kausalitätsbalken des Widerstandes R_1 , welcher frei wählbar ist, außerhalb der linken 1-Junction eingezeichnet.

Um zur Zustandsraumdarstellung zu gelangen gibt es demnach zwei verschiedene Möglichkeiten. Für die Erstellung des Zustandsraums wird daher der Bondgraph aus [Abbildung 5.70](#) herangezogen. Im nächsten Schritt extrahieren wir die Gleichungen aus dem Bondgraphen (siehe [Tabelle 5.2](#)).

Element	ohne Kausalität	mit Kausalität
$Se :$	$u = 30V$	$u = 30V$
$1 : links$	$u - u_{R_1} - u_L = 0$	$u_{R_1} = u - u_L$
$R : R_1$	$u_{R_1} = R_1 \cdot i$	$i = \frac{1}{R_1} \cdot u_{R_1}$
$0 :$	$i - i_L - i_2 = 0$	$i_2 = i - i_L$
$I : L$	$i_L = \frac{1}{L} \int u_L \cdot dt$	$i_L = \frac{1}{L} \int u_L \cdot dt = x_1$
$1 : rechts$	$u_L - u_{R_2} - u_C = 0$	$u_L = u_{R_2} + u_C$
$C : C$	$u_C = \frac{1}{C} \int i_2 \cdot dt$	$u_C = \frac{1}{C} \int i_2 \cdot dt = x_2$
$R : R_2$	$u_{R_2} = i_2 \cdot R_2$	$u_{R_2} = i_2 \cdot R_2$

Tab. 5.2.: Gleichungen ermittelt anhand des Bondgraphen aus [Abbildung 5.69](#)

Jetzt kann die Zustandsraumdarstellung erstellt werden. Die Zustandsvariablen sind:

$$x_1 = i_L \quad (5.40)$$

$$x_2 = u_C \quad (5.41)$$

Wir beginnen mit der ersten Zustandsgleichung:

$$\begin{aligned} \dot{x}_1 &= \frac{1}{L} \cdot u_L = \frac{1}{L}(u_{R_2} + u_C) \\ \dot{x}_1 &= \frac{1}{L}(u_{R_2} + x_2) \end{aligned} \quad (5.42)$$

Die zweite Zustandsgleichung lautet:

$$\begin{aligned} \dot{x}_2 &= \frac{1}{C} \cdot i_2 = \frac{1}{C}(i - i_L) \\ \dot{x}_2 &= \frac{1}{C}(i - x_1) \end{aligned} \quad (5.43)$$

Leider ist es nicht möglich die Variablen u_{R_2} und i durch Zustandsvariablen oder Eingangsgrößen auszudrücken. Wenn wir die Variablen durch andere Größen ausdrücken werden die Ausdrücke immer größer und wir gelangen zu keinem Ende. In solchen Situationen sprechen wir von algebraischen Schleifen. Nachdem wir bereits im Bondgraphen bemerkt haben, dass algebraische Schleifen dann auftreten wenn wir an einem R-Element dessen Kausalität frei wählen können, ist auch klar, dass die algebraische Schleife dort ihren Ursprung hat. Im nächsten Schritt untersuchen wir deshalb die kausalen Gleichungen an den Widerständen R_1 und R_2 .

$$\begin{aligned} i &= \frac{1}{R_1} \cdot u_{R_1} \\ i &= \frac{1}{R_1}(u - u_L) = \frac{1}{R_1}(u - u_{R_2} - u_C) \\ i &= \frac{1}{R_1}(u - u_{R_2} - x_2) \end{aligned} \quad (5.44)$$

Problem: Gleichung (5.44) kann nicht weiter vereinfacht werden. Wir können den Strom i nicht berechnen, da wir die Spannung u_{R_2} nicht kennen.

$$\begin{aligned} u_{R_2} &= R_2 \cdot i_2 \\ u_{R_2} &= R_2(i - i_L) = R_2(i - x_1) \end{aligned} \quad (5.45)$$

Auf der anderen Seite können wir u_{R_2} in Gleichung (5.45) nicht berechnen, da wir den Strom i nicht kennen. Dies wird im Bondgraphen durch Einzeichnen der in [Abschnitt 5.5](#) beschriebenen *kausalen Pfade* ersichtlich ([Abbildung 5.71](#)).

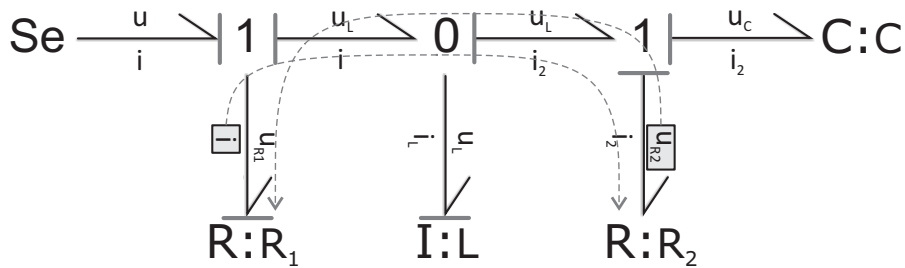


Abb. 5.71.: Kausale Pfade im Bondgraphen. Durch Einzeichnen der kausalen Pfade werden algebraische Schleifen ersichtlich

Um dieses Problem zu lösen, müssen wir obiges Gleichungssystem, bestehend aus 2 Gleichungen (5.44), (5.45) und 2 Unbekannten (u_{R_2}, i) lösen. Dazu wird Gleichung (5.45) in Gleichung (5.44) eingesetzt.

$$\begin{aligned}
 i &= \frac{1}{R_1}(u - u_{R_2} - x_2) \\
 i &= \frac{1}{R_1}(u - R_2(i - x_1) - x_2) = \frac{1}{R_1}(u - R_2 \cdot i + R_2 \cdot x_1 - x_2) \\
 i &= \dots = \frac{1}{R_1 + R_2}(u + R_2 \cdot x_1 - x_2)
 \end{aligned} \tag{5.46}$$

Im nächsten Schritt können die Gleichungen (5.42), (5.43) in die Zustandsraumdarstellung übergeführt werden. Dazu dürfen in beiden Gleichungen nur noch Zustandsvariablen, Eingangsgrößen und die Variable i vorkommen. In Gleichung (5.42) muss die Spannung u_{R_2} daher noch mit Gleichung (5.45) substituiert werden.

$$\begin{aligned}
 \dot{x}_1 &= \frac{1}{L}(R_2 \cdot i - R_2 \cdot x_1 + x_2) \\
 \dot{x}_1 &= \frac{1}{L} \left(R_2 \cdot \frac{1}{R_1 + R_2} (u + R_2 \cdot x_1 - x_2) - R_2 \cdot x_1 + x_2 \right) \\
 \dot{x}_1 &= \frac{1}{L} \cdot \frac{R_2}{R_1 + R_2} \cdot u - \frac{1}{L} \cdot \frac{R_1 \cdot R_2}{R_1 + R_2} \cdot x_1 + \frac{1}{L} \cdot \frac{R_1}{R_1 + R_2} \cdot x_2
 \end{aligned} \tag{5.47}$$

In Gleichung (5.43) wird einfach Gleichung (5.46) eingesetzt.

$$\begin{aligned}
 \dot{x}_2 &= \frac{1}{C}(i - x_1) \\
 \dot{x}_2 &= \frac{1}{C} \left(\frac{1}{R_1 + R_2} (u + R_2 \cdot x_1 - x_2) - x_1 \right) \\
 \dot{x}_2 &= \frac{1}{C} \cdot \frac{1}{R_1 + R_2} \cdot u - \frac{1}{C} \cdot \frac{R_1}{R_1 + R_2} \cdot x_1 - \frac{1}{C} \cdot \frac{1}{R_1 + R_2} \cdot x_2
 \end{aligned} \tag{5.48}$$

In *Vektor-Matrix Form* (Matrizendarstellung) erhalten wir folgende Zustandsgleichung:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -\frac{1}{L} \cdot \frac{R_1 \cdot R_2}{R_1 + R_2} & \frac{1}{L} \cdot \frac{R_1}{R_1 + R_2} \\ -\frac{1}{C} \cdot \frac{R_1}{R_1 + R_2} & -\frac{1}{C} \cdot \frac{1}{R_1 + R_2} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{1}{L} \cdot \frac{R_2}{R_1 + R_2} \\ \frac{1}{C} \cdot \frac{1}{R_1 + R_2} \end{pmatrix} \cdot u \quad (5.49)$$

Um das System simulieren zu können, müssen wir noch definieren welche Variablen als Ausgangsvariablen verwendet werden. Als Ausgangsvariable wird die Spannung am Kondensator u_C verwendet.

$$y = \begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (5.50)$$

5.7.3. Strukturelle Singularitäten

Sobald eine strukturelle Singularität auftritt, können die Kausalitätsbalken der energiespeichernden Elemente nicht mehr dort platziert werden wo sie vorgesehen sind. Aus der *integralen* Kausalität wird also zwangsweise eine *differentielle* Kausalität.

Beispiel für eine strukturelle Singularität

Abbildung 5.72 veranschaulicht eine Spannungsquelle mit zwei Induktivitäten, welche in Serie geschaltet sind. [Abbildung 5.73](#) veranschaulicht den akausalen sowie den kau-

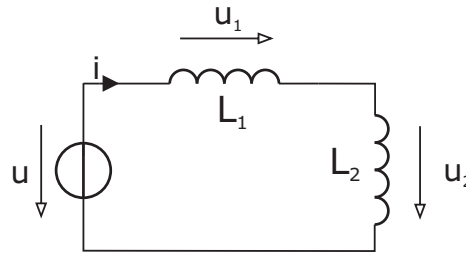


Abb. 5.72.: Spannungsquelle mit zwei in Serie geschalteten Induktivitäten

salisierten Bondgraphen der Schaltung.

Kausalisiert man den Bondgraphen, so stellt man schnell fest, dass bei einer der beiden Induktivitäten der Kausalitätsbalken nicht an der gewünschten Stelle angebracht werden kann (siehe [Abbildung 5.73](#), rechtes Bild).

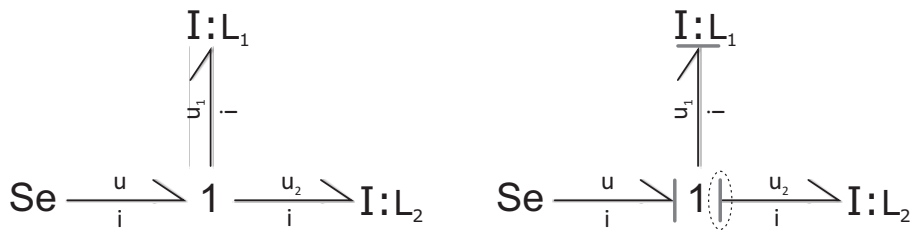


Abb. 5.73.: Links: Akausaler Bondgraph der Spannungsquelle mit zwei in Serie geschalteten Induktivitäten. Rechts: Kausalitätskonflikt - Der umrahmte Kausalitätsbalken befindet sich an der falschen Seite des Bonds

Hintergründe

Um die Ursache einer strukturellen Singularität zu verstehen, betrachten wir die Komponentengleichungen beider Induktivitäten:

$$i = i(0) + \frac{1}{L_1} \int_0^t u_1 \cdot dt \tag{5.51}$$

$$i = i(0) + \frac{1}{L_2} \int_0^t u_2 \cdot dt \tag{5.52}$$

In diesen *kausalen Gleichungen* (Zuweisungen) stellt i den Output dar. Dieser Output ist gleichzeitig der Ausgang eines Integrators und damit eine Zustandsvariable. Die Schaltung aus Abbildung 5.72 besteht aus zwei Induktivitäten von denen jede eine Zustandsvariable aufweist. Aufgrund der zwei energiespeichernden Elemente könnte man glauben, es handle sich um ein System 2. Ordnung. Da die Induktivitäten in Serie geschaltet sind, gibt es aber lediglich eine Zustandsgröße und zwar den Strom i . In solch einem Fall sind die Anfangsbedingungen $i(0)$ direkt voneinander abhängig und können aus diesem Grund nicht unabhängig (frei) bestimmt werden. Da man diese im vorliegenden Beispiel jedoch unabhängig voneinander bestimmen kann, tritt eine so genannte *strukturelle Singularität* auf.

Solange ein Programm verwendet wird, welches in der Lage ist Bondgraphen zu implementieren, muss sich der Benutzer nicht um Kausalitätskonflikte kümmern. Dies wird im Normalfall durch geeignete Algorithmen und/oder Heuristiken, welche vom Programm zur Verfügung gestellt werden, erledigt. Das Programm Dymola, welches in [Unterabschnitt 6.3.2](#) beschrieben wird, als Beispiel, verwendet zur Eliminierung struktureller Singularitäten den Algorithmus nach *Pantelides* (siehe [Unterabschnitt 6.9.3](#)). Beabsichtigt man aus dem Bondgraphen eine Zustandsraumdarstellung zu erstellen so funktioniert dies beim Auftreten eines Kausalitätskonflikts nicht mehr.

5.7.4. Eliminieren von strukturellen Singularitäten

Hier gibt es mehrere Möglichkeiten, um die strukturelle Singularität in den Griff zu bekommen. Man sollte daher, das erstellte mathematische Modell immer kritisch hinterfragen.

Modifikation des physikalischen Systems

Der Kausalitätskonflikt welcher in [Abbildung 5.73](#) auftritt, kann sehr einfach behoben werden. Die beiden Induktivitäten können beispielsweise zu einer Induktivität zusammengefasst werden ([Abbildung 5.74](#)).

$$\text{Se} \begin{array}{c} \xrightarrow{u} \\ \xrightarrow{i} \end{array} \mid 1 \begin{array}{c} \xrightarrow{u_2} \\ \xrightarrow{i} \end{array} \mid \text{I}: L_1 + L_2$$

Abb. 5.74.: Behebung des Kausalitätskonflikts. Die Induktivitäten L_1 und L_2 werden zu einer Induktivität zusammengefasst.

Durch Zusammenschalten der einzelnen Induktivitäten zu einer Gesamtinduktivität wird sichergestellt, dass es nur eine Anfangsbedingung $i(0)$ geben kann. Somit verschwindet der Kausalitätskonflikt.

Des Weiteren kann man einen (parasitären) Widerstand R parallel zu einer der beiden Induktivitäten schalten ([Abbildung 5.75](#)). Der dazugehörige Bondgraph ist in [Abbil-](#)

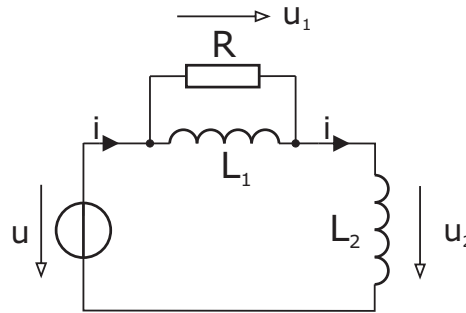
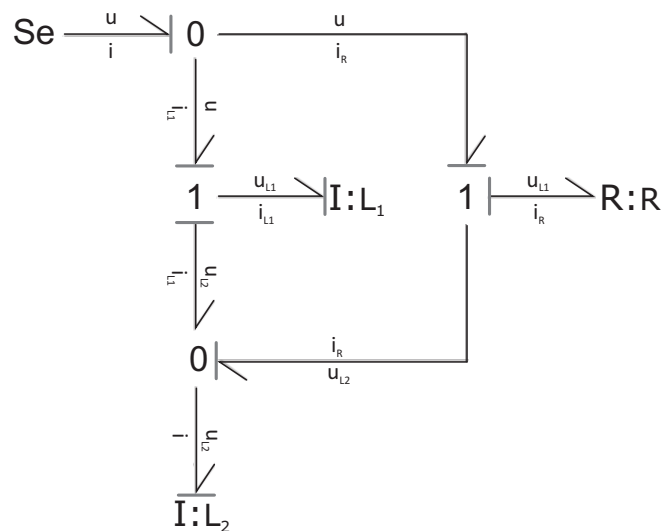


Abb. 5.75.: Behebung des Kausalitätskonflikts durch „parallel schalten“ eines Widerstandes R zur Induktivität L_1 , wobei für den Widerstand R sehr große Werte (im Bereich einiger $M\Omega$) zu wählen sind.

[dung 5.76](#) veranschaulicht. Durch das zusätzliche Element verschwindet der Kausalitätskonflikt.

Abb. 5.76.: Kausaler Bondgraph der Schaltung aus [Abbildung 5.75](#)

Probleme bei parasitären Elementen

Durch das Hinzufügen parasitärer Elemente können dem System zusätzliche Pole hinzugefügt werden. Diese Pole können dabei um mehrere Zehnerpotenzen von den systemeigenen Polen entfernt sein. D.h., dass das System dadurch sowohl schnelle als auch langsame Zeitkonstanten beinhaltet. Man spricht dann von sogenannten *steifen Systemen*. Steife Systeme stellen höhere Anforderungen an den Solver, sofern die Simulation in einer vernünftigen Zeit ausgeführt werden soll (mehr dazu in [Kapitel 7](#)).

Man erkennt also schon bei sehr einfachen Beispielen wie wichtig die Grundlagen der jeweiligen Fachgebiete sind. Es nützt der beste Modellierer nichts, wenn er nicht mit den entsprechenden Gebieten vertraut ist.

In vielen Fällen ist der Grund für eine strukturelle Singularität jedoch nicht so gut ersichtlich. Im Folgenden wird gezeigt wie eine strukturelle Singularität innerhalb des Modells behoben werden kann. D.h. das Modell wird nicht zuerst angepasst, um die strukturelle Singularität zu eliminieren.

Anleitung (Algorithmus) zur Eliminierung struktureller Singularitäten

Um strukturelle Singularitäten zu eliminieren, ohne dabei das physikalische System zu modifizieren, bedarf einiger Punkte, welche beachtet werden müssen. Ausgangspunkt ist ein mathematisches Modell unseres Systems mit dazugehörigem akausalen Bondgraphen. Beim Kausalisieren tritt dann an einem energiespeichernden Element, dessen

Anfangsbedingung nicht frei gewählt werden kann, eine strukturelle Singularität auf. Nun muss folgendes getan werden:

- (1) Nachdem an einem energiespeichernden Element ein Konflikt auftritt, verringert sich die Anzahl der Zustandsvariablen um eins. D.h., wenn wir n energiespeichernde Elemente haben, muss die Anzahl der Elemente abgezogen werden an denen ein Konflikt auftritt, um eine korrekte Ordnung des Systems und damit die Anzahl der Zustandsgleichungen zu erhalten. Die Anzahl der konfliktbehafteten Elemente kann auch größer eins sein. Wenn beispielsweise drei Massen starr miteinander verbunden werden, dann kann für jede Masse eine separate Anfangsbedingung (Anfangsgeschwindigkeit) gewählt werden, obwohl alle drei Massen dieselbe Geschwindigkeit besitzen müssen (siehe [Abbildung 5.77](#)).
- (2) Nun werden die Zustandsgleichungen für die energiespeichernden Elemente ohne Konflikt erstellt. Hier gilt wiederum, dass alle Variablen durch bekannte Größen, also durch Zustandsgrößen und Eingangsgrößen ausgedrückt werden müssen. Jedoch kommt es aufgrund der strukturellen Singularität innerhalb des Systems zu Problemen. Es können nicht alle unbekanntes Größen durch bekannte Größen ausgedrückt werden. Dazu folgendes Beispiel:

$$\dot{x}_1 = \boxed{\frac{dk}{dt}} + x_1 - 2 \cdot x_2 \quad (5.53)$$

Wobei der eingerahmte Term unbekannt ist.

- (3) Jedoch sind die Unbekannten „immer“ in abgeleiteter Form vorhanden. Dies muss so sein, da die *integrale Kausalität* am energiespeichernden Element durch den Konflikt zu einer *differentiellen Kausalität* wird. Nun gilt es für die unbekanntes Größe, welche in abgeleiteter Form vorkommt (im Beispiel dk/dt) eine zusätzliche algebraische Gleichung zu finden, welche die Unbekanntes Größe in nicht abgeleiteter Form, also k , ermittelt. Die zusätzlich ermittelte algebraische Gleichung könnte folgendermaßen aussehen:

$$k = 3 \cdot x_1 \quad (5.54)$$

- (4) Sobald diese Gleichung gefunden wurde (hier Gleichung (5.54)) und nur noch bekannte Größen enthält, wird diese abgeleitet

$$\boxed{\frac{dk}{dt}} = 3 \cdot \dot{x}_1 \quad (5.55)$$

und in Gleichung (5.53) mit der unbekanntes Größe der Zustandsgleichung substituiert.

$$\dot{x}_1 = 3 \cdot \dot{x}_1 + x_1 - 2 \cdot x_2 \quad (5.56)$$

- (5) Schlussendlich erhalten wir unsere gesuchte Zustandsgleichung, welche nur noch Zustandsvariablen enthält.

$$\dot{x}_1 = \frac{1}{4}x_1 - \frac{1}{2}x_2 \quad (5.57)$$

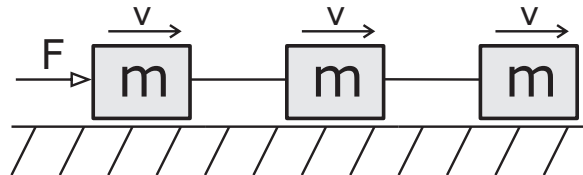


Abb. 5.77.: Drei Massen welche starr miteinander verbunden sind. Es treten an zwei Massen strukturelle Singularitäten auf, da alle Anfangsbedingungen unabhängig voneinander bestimmt werden können.

Dieser Algorithmus wird nun anhand eines Beispiels demonstriert.

Mechatronisches System - DC Motor mit Last

Im Folgenden wird gezeigt, wie strukturelle Singularitäten anhand des in [Abschnitt 5.7.4](#) vorgestellten Algorithmus direkt im Modell eliminiert werden, ohne dieses erst anzupassen. Dazu wird das Modell eines belasteten DC-Motors herangezogen. Die mechanische Seite des Motors besteht aus einer rotierenden Last welche durch deren Trägheit J modelliert wird. Durch Kontakt zum Boden tritt zusätzlich eine Reibung b_τ auf. Die Last selbst ist über eine flexible Achse, beschrieben durch deren Torsionssteifigkeit k_τ , am Motor befestigt. Die elektrische Seite des Motors beinhaltet die Induktivität L der Windungen und deren Kupferverluste R_W . Die Trägheit des Motors wird vernachlässigt, da diese im Vergleich zur Last sehr klein ist.

[Abbildung 5.78](#) veranschaulicht den DC-Motor mit Last. Der dazugehörige Bondgraph

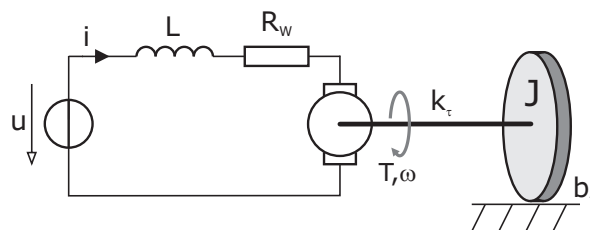


Abb. 5.78.: DC-Motor mit rotativer Last

ist in [Abbildung 5.79](#) dargestellt. Sobald dieser Bondgraph kausalisiert wird, tritt ein

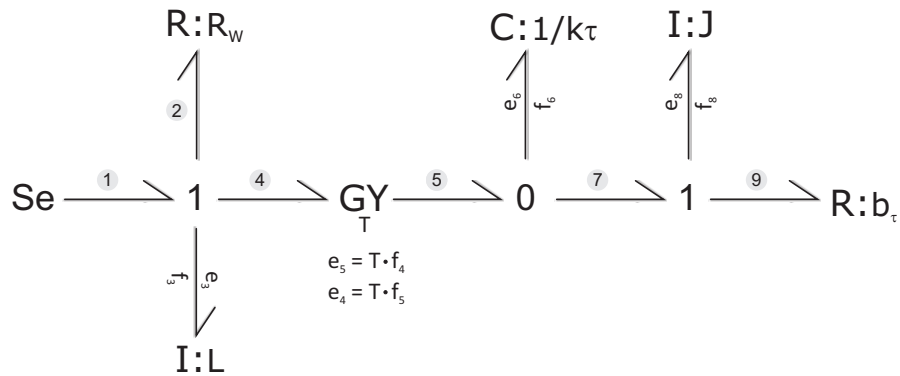


Abb. 5.79.: Akausaler Bondgraph des DC Motors mit Last

Kausalitätskonflikt am C-Element auf. Nun werden wir, ohne das Modell in Frage zu

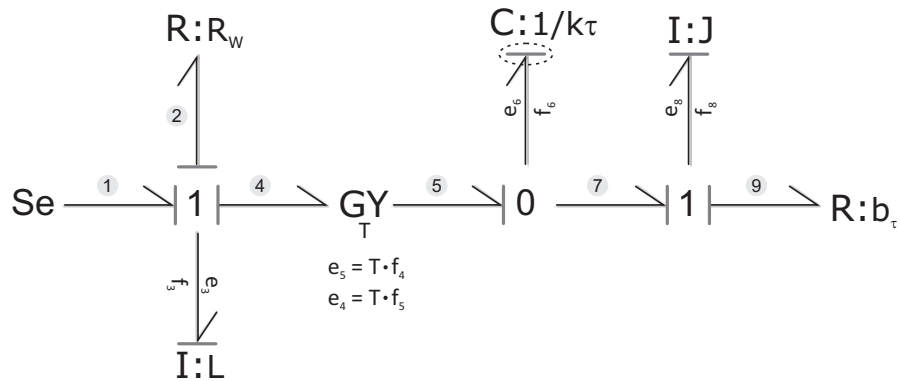


Abb. 5.80.: Kausaler Bondgraph des DC Motors mit Last. Der Kausalitätsbalken am C-Element befindet sich auf der falschen Seite des Bonds

stellen, den Bondgraphen systematisch analysieren. Wir werden also versuchen für jedes energiespeichernde Element eine Zustandsgleichung aufzustellen. Da unser System zwei I-Elemente und ein C-Element besitzt, würde es sich, sofern kein Kausalitätskonflikt auftritt, um ein System 3. Ordnung handeln welches drei Zustandsvariablen besitzt und daher mit drei Zustandsgleichungen beschrieben werden kann. Aufgrund des Konflikts welcher am C-Element auftritt, kann dort keine Zustandsvariable existieren. Betrachten wir zu diesem Zweck die 0-Junction. Hier ist das Potential identisch und daher die Summe der Flussvariablen gleich Null.

$$e_5 = e_6 = e_7$$

$$f_5 - f_6 - f_7 = 0$$

Betrachten wir nun den kausalen Bondgraphen aus [Abbildung 5.80](#) so erkennen wir, dass f_5 die Ausgangsgröße der 0-Junction ist.

$$f_5 = f_6 + f_7 = 0 \quad (5.58)$$

Als nächstes betrachten wir die Komponentengleichung des C-Elements in integraler Form:

$$e_6 = k_\tau \int f_6 \cdot dt \quad (5.59)$$

Die Zustandsvariable, welche den Ausgang des Integrators darstellt, ist die Potentialvariable e_6 . Aufgrund des Kausalitätskonflikts welcher am C-Element auftritt, ist nicht das Potential, sondern der Fluss f_6 der Ausgang des Elements. Wir erhalten:

$$f_6 = \frac{1}{k_\tau} \cdot \frac{de_6}{dt} \quad (5.60)$$

Nachdem f_6 den Ausgang des C-Elements darstellt, kann e_6 keine Zustandsvariable sein. Daraus folgt, dass auch keine Zustandsgleichung erstellt werden kann. Welche Auswirkungen das auf die verbleibenden zwei Zustandsgleichungen hat, wird im Folgenden ersichtlich.

Die linke 1-Junction besitzt denselben Fluss ($f_1 = f_2 = f_3 = f_4$), daher muss die Summe der Spannungen gleich Null sein.

$$e_1 - e_2 - e_3 - e_4 = 0$$

Betrachten wir nun den kausalen Bondgraphen aus [Abbildung 5.80](#) so erkennen wir, dass e_3 die Ausgangsgröße der 1-Junction ist.

$$e_3 = e_1 - e_2 - e_4 \quad (5.61)$$

Diesmal ist der Ausgang des energiespeichernden Elements (I-Element) so, dass es sich auch um den Ausgang eines Integrators handelt.

$$f_3 = \frac{1}{L} \int e_3 \cdot dt = x_1 \quad (5.62)$$

Die zeitliche Ableitung dieser Gleichung liefert die erste Zustandsgleichung.

$$\dot{x}_1 = \frac{1}{L} \cdot e_3 \quad (5.63)$$

Nachdem e_3 den Eingang in das I-Element und daher den Ausgang der 0-Junction darstellt, kann nun Gleichung (5.61) in Gleichung (5.63) eingesetzt werden.

$$\dot{x}_1 = \frac{1}{L} (e_1 - e_2 - e_4) \quad (5.64)$$

Durch Substitution der einzelnen Potentialvariablen mit den dazugehörigen Komponentengleichungen erhalten wir folgende Gleichung.

$$\dot{x}_1 = \frac{1}{L} (u - R_W \cdot f_2 - T \cdot f_5) \quad (5.65)$$

Für die Flussvariable f_2 können wir genauso gut f_3 verwenden, da es sich um denselben Fluss handelt.

$$\dot{x}_1 = \frac{1}{L} (u - R_W \cdot f_3 - T \cdot f_5) \quad (5.66)$$

Der Fluss f_5 kann mit Gleichung (5.58) beschrieben werden.

$$\dot{x}_1 = \frac{1}{L} (u - R_W \cdot x_1 - T \cdot (f_6 + f_7)) \quad (5.67)$$

Nun müssen wir noch f_6 und f_7 durch Zustandsvariablen ausdrücken. Für die Flussvariable f_7 wird die rechte 1-Junction betrachtet. Hier ist gut ersichtlich, dass:

$$f_7 = f_8 = f_9 \quad (5.68)$$

Wobei f_8 den Ausgang des I-Elements darstellt und damit eine Zustandsvariable ist, welche fortan mit x_2 bezeichnet wird.

$$\dot{x}_1 = \frac{1}{L} (u - R_W \cdot x_1 - T \cdot (f_6 + x_2)) \quad (5.69)$$

Für f_6 kann nur noch Gleichung (5.60) eingesetzt werden.

$$\dot{x}_1 = \frac{1}{L} \left(u - R_W \cdot x_1 - T \cdot \left(\frac{1}{k_\tau} \cdot \boxed{\frac{de_6}{dt}} + x_2 \right) \right) \quad (5.70)$$

Aufgrund des Kausalitätskonflikts hat diese Gleichung eine Unbekannte de_6/dt zu viel. Um dieses Problem in den Griff zu bekommen, muss eine zusätzliche „algebraische“ Gleichung gefunden werden (siehe Punkt (3) aus [Abschnitt 5.7.4](#)). An dieser Stelle muss folgendes getan werden: Man suche eine Gleichung welche die Unbekannte de_6/dt in „nicht“ abgeleiteter Form beschreibt und leite diese anschließend ab. D.h., wir suchen

eine algebraische Gleichung welche uns die Potentialvariable e_6 beschreibt.

$$\begin{aligned} e_6 &= e_5 = T \cdot f_4 = T \cdot f_3 \\ e_6 &= T \cdot x_1 \end{aligned} \quad (5.71)$$

Durch Bilden der zeitlichen Ableitung (siehe Punkt (4) aus [Abschnitt 5.7.4](#)) dieser Gleichung erhalten wir:

$$\boxed{\frac{de_6}{dt}} = T \cdot \dot{x}_1 \quad (5.72)$$

Setzt man diese Gleichung nun in Gleichung (5.70) ein, so sind nur noch bekannte Größen vorhanden.

$$\dot{x}_1 = \frac{1}{L} \left(u - R_W \cdot x_1 - T \cdot \left(\frac{1}{k_\tau} \cdot T \cdot \dot{x}_1 + x_2 \right) \right) \quad (5.73)$$

Jetzt muss diese Gleichung noch nach \dot{x}_1 aufgelöst werden, sodass wir eine korrekte Zustandsgleichung erhalten:

$$\dot{x}_1 = \underbrace{\left(\frac{k_\tau}{L \cdot k_\tau + T^2} \right)}_Q (u - R_W \cdot x_1 - T \cdot x_2) \quad (5.74)$$

Nun fehlt uns noch die zweite Zustandsgleichung.

$$f_8 = \frac{1}{J} \int e_8 \cdot dt = x_2 \quad (5.75)$$

Die zeitliche Ableitung dieser Gleichung liefert die zweite Zustandsgleichung.

$$\dot{x}_2 = \frac{1}{J} \cdot e_8 \quad (5.76)$$

Die rechte 1-Junction besitzt denselben Fluss weshalb die Summe der Potentialvariablen wiederum gleich Null sein muss.

$$e_7 - e_8 - e_9 = 0$$

Ausgehend vom kausalen Bondgraphen aus [Abbildung 5.80](#) erkennen wir sofort, dass die Potentialvariable e_8 den Ausgang darstellt.

$$e_8 = e_7 - e_9 \quad (5.77)$$

Diese Gleichung wird in die zweite Zustandsgleichung (5.76) eingesetzt.

$$\dot{x}_2 = \frac{1}{J}(e_7 - e_9) \quad (5.78)$$

Es wird wiederum solange substituiert bis nur noch Eingangs- und Zustandsgrößen vorhanden sind.

$$\begin{aligned} \dot{x}_2 &= \frac{1}{J}(e_5 - b_\tau \cdot f_9) = \frac{1}{J}(T \cdot f_4 - b_\tau \cdot f_8) \\ \dot{x}_2 &= \frac{1}{J}(T \cdot f_3 - b_\tau \cdot x_2) = \frac{1}{J}(T \cdot x_1 - b_\tau \cdot x_2) \end{aligned} \quad (5.79)$$

Aufstellen der Zustandsraumdarstellung

Aus den Zustandsgleichungen (5.70) und (5.79) folgt unmittelbar die Zustandsgleichung.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -Q \cdot R_W & -Q \cdot T \\ \frac{T}{J} & -\frac{b_\tau}{J} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} Q \\ 0 \end{pmatrix} \cdot u(t) \quad (5.80)$$

Nun fehlt uns nur noch die Ausgangsgleichung. Wir interessieren uns für die Flussvariable f_8 welche der Zustandsvariable x_2 entspricht und nichts anderes ist als die Winkelgeschwindigkeit der Masse welche durch die Trägheit J beschrieben wird.

$$y = \begin{pmatrix} 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (5.81)$$

Ist die strukturelle Singularität im Bondgraphen ersichtlich?

Im vorliegenden Beispiel tritt eine strukturelle Singularität auf die im Bondgraphen nicht ohne weiteres ersichtlich ist. Ist man aber mit den in [Abschnitt 5.6](#) vorgestellten dualen Bondgraphen vertraut so ist die strukturelle Singularität sehr gut ersichtlich. [Abbildung 5.81](#) veranschaulicht die strukturelle Singularität.

Das duale Element einer Kapazität ist eine Induktivität. Beim Betrachten des Bondgraphen in [Abbildung 5.81](#) ist gut ersichtlich, dass die Kapazität in Kombination des Gyrtors nichts anderes ist wie eine Induktivität, welche zur Induktivität L in Serie geschaltet ist weshalb auch eine strukturelle Singularität zustande kommt.

Kann die strukturelle Singularität vermieden werden?

Zu Beginn des Beispiels wurde erwähnt dass die Trägheit des Motors vernachlässigt wird, da diese viel kleiner ist als die der Last. Erstellen wir also im Folgenden den Bond-

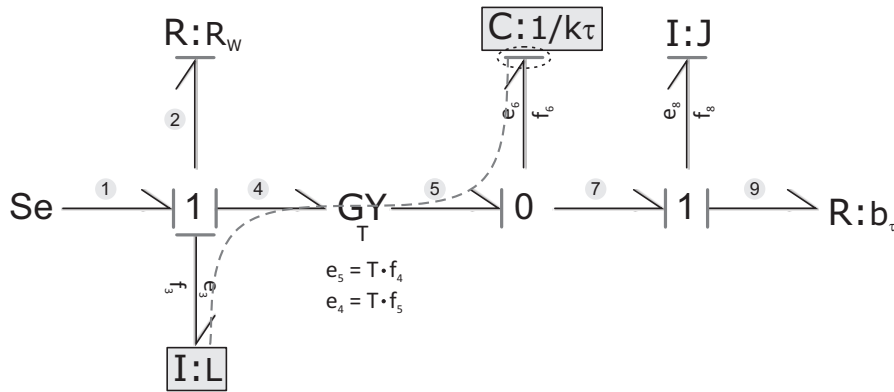


Abb. 5.81.: Veranschaulichung des Kausalitätskonflikts des (vereinfachten) mechatronischen Systems.

graphen des Systems mit Berücksichtigung der Trägheit des Motors J_M inklusive der dazugehörigen Lagerreibung R_b (siehe [Abbildung 5.82](#)). Wenn wir diesen Bondgraphen

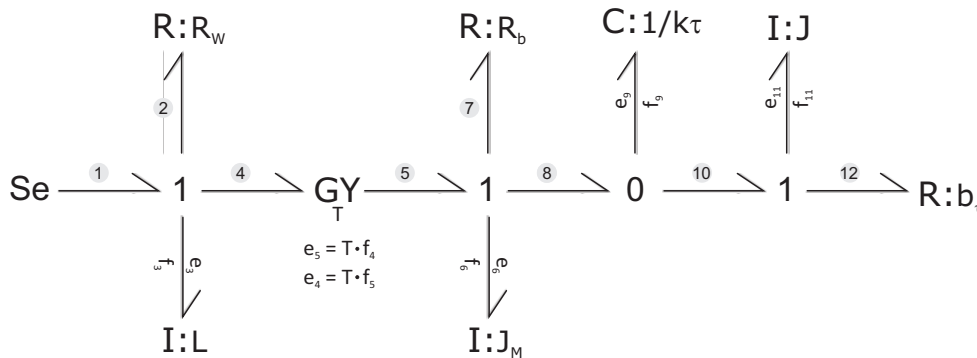


Abb. 5.82.: Akausaler Bondgraph des mechatronischen Systems mit Berücksichtigung der Trägheit des Motors.

jetzt Kausalisieren, verschwindet die strukturelle Singularität (siehe [Abbildung 5.83](#)).

5.8. Die vier Grundvariablen der Bondgraphen-Modellierung

Nehmen wir an, wir haben einen Feder-Masse Schwinger und wollen die Position der Masse bestimmen. Mit den bisher vorgestellten Bondgraphen ist das unmöglich. Bisher wurden zwei Variablen vorgestellt, welche als Potentialvariable $e(t)$ und Flussvariable $f(t)$ bezeichnet wurden. D.h., dass wir im Falle unseres Feder-Masse Schwingers nur

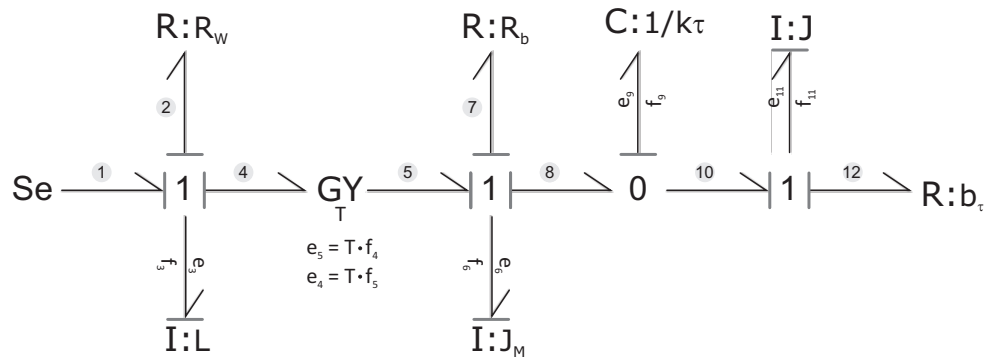


Abb. 5.83.: Kausaler Bondgraph des mechatronischen Systems mit Berücksichtigung der Trägheit des Motors. Der Kausalitätskonflikt ist dadurch nicht mehr vorhanden.

eine Anfangsgeschwindigkeit v_0 bzw. eine Anfangskraft F_0 vorgeben können.

In physikalischen Systemen gibt es jedoch mehr als nur zwei (systembeschreibende) Variablen. In der Mechanik spricht man beispielsweise neben Geschwindigkeiten (Flussvariable) und Kräften (Potentialvariable) des Öfteren von Positionen und Impulsen. In der Elektrotechnik hingegen nicht nur von Strömen und Spannungen sondern auch von Ladungen und magnetischen Flüssen.

In diesem Abschnitt werden zwei weitere Variablen eingeführt. Sodann wird im [Abschnitt 5.9](#) gezeigt, wie diese Variablen in der Bondgraphen-Modellierung eingesetzt werden.

In der Bondgraphen-Modellierung werden diese Variablen wiederum etwas allgemeiner formuliert. Dies soll anhand der allgemeingültigen Gleichungen aus [Tabelle 4.2](#) erläutert werden. Aus der Gleichung zur Speicherung der Flussvariablen

$$f(t) = \frac{1}{\alpha} \int e(t) \cdot dt$$

ergibt sich:

$$p(t) \equiv \int e(t) \cdot dt \tag{5.82}$$

Wobei $p(t)$ als *generalisierter Impuls* bezeichnet wird. Angewendet auf (mechanisch) translatorische Systeme erhalten wir:

$$\begin{aligned} v(t) &= \frac{1}{m} \int F(t) \cdot dt \\ p(t) &\equiv \int F(t) \cdot dt \end{aligned} \tag{5.83}$$

Oder anders formuliert:

$$\frac{dp(t)}{dt} = F(t) \quad (5.84)$$

beziehungsweise

$$p(t) = m \cdot v(t) \quad (5.85)$$

Diese Beziehung (Axiom) stammt aus der Newton'schen Mechanik und wird als Impulserhaltungssatz bezeichnet.

Aus der Gleichung zur Speicherung der Potentialvariable

$$e(t) = \frac{1}{\beta} \int f(t) \cdot dt$$

ergibt sich folgendes:

$$q(t) = \int f(t) \cdot dt \quad (5.86)$$

Wobei $q(t)$ als *generalisierte Position* bezeichnet wird. Für translatorische Systeme ergibt sich wiederum folgendes:

$$F(t) = k \underbrace{\int v(t) \cdot dt}_{x(t)} = k \cdot x(t)$$

$$x(t) = \int v(t) \cdot dt \quad (5.87)$$

Der Zusammenhang der vier Variablen kann auch graphisch veranschaulicht werden (siehe [Abbildung 5.84](#)).

In [Tabelle 5.3](#) wird gezeigt was die Variablen $p(t)$ und $q(t)$ für eine Bedeutung in den verschiedenen Domänen haben.

5.9. Modulierte Elemente

Bisher wurden Quellen vorgestellt, welche entweder einen konstanten Fluss oder ein konstantes Potential zur Verfügung stellen. Des Weiteren wurden Transformatoren und Gyrotoren mit einem konstanten Übersetzungsverhältnis eingeführt. Im Folgenden wird gezeigt wie konstante Werte bzw. Übersetzungsverhältnisse durch variable (modulierte) Übersetzungen ersetzt werden können. Dazu wird dieser Abschnitt in zwei Teile gegliedert. Zum einen werden Sensoren vorgestellt und zum anderen modulierte Elemente.

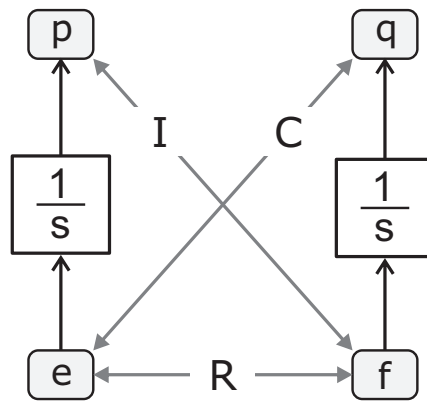


Abb. 5.84.: Zusammenhang der Variablen $e(t)$, $f(t)$, $p(t)$ und $q(t)$

	$e(t)$	$f(t)$	$p(t)$	$q(t)$
<i>Elektrische Systeme</i>	Spannung u (V)	Strom i (A)	Magn. Fluss Φ (Vs)	Ladung q (As)
<i>Translatorische Systeme</i>	Kraft F (N)	Geschwindigkeit v (m/s)	Impuls p (Ns)	Position x (m)
<i>Rotatorische Systeme</i>	Drehmoment T (Nm)	Winkelgeschw. ω (rad/s)	Drehimpuls L (Nms)	Winkel φ (rad)
<i>Hydraulische Systeme</i>	Druck p (N/m ²)	Volumenstrom q (m ³ /s)	Druckimpuls Γ (Ns/m ²)	Volumen V (m ³)

Tab. 5.3.: Zusammenhang der allgemeingültigen Variablen und den Variablen der verschiedenen Domäne

5.9.1. Sensoren

Sensoren werden verwendet, um das Potential oder den Fluss an einer Verzweigung zu messen. Zum Messen dieser Signale werden keine Bonds (Energieverbindungen) verwendet, sondern Signalverbindungen. Bei einer Signalverbindung findet kein Energiefluss statt. Das Produkt aus Potential- und Flussvariable ist gleich Null. Sensoren können ausschließlich an Verzweigungen angeschlossen werden, da hier klar definiert ist, dass es sich entweder um denselben Fluss oder um dasselbe Potential handelt. Wollen wir den Fluss ermitteln, so müssen wir einen Fluss-Sensor an eine 1-Junction anschließen (siehe [Abbildung 5.85](#)).

Im umgekehrten Fall wird ein Potential-Sensor an eine 0-Junction angeschlossen. Sind wir nun beispielsweise an der Position eines bestimmten Teils (Masse) eines mechanischen Systems interessiert, so muss die Flussvariable integriert werden (siehe [Abbil-](#)

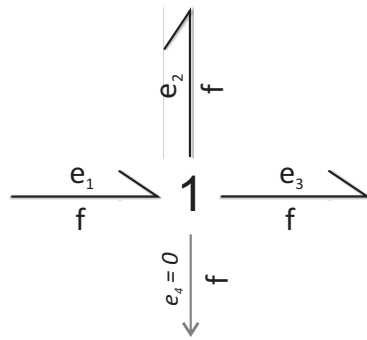


Abb. 5.85.: Sensor zur Ermittlung des Flusses.

dung 5.86).

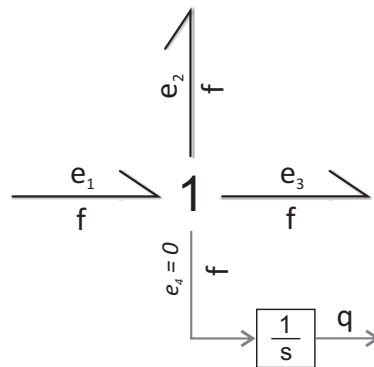


Abb. 5.86.: Sensor zur Ermittlung der (generalisierten) Position. Dazu wird die Flussvariable in einen Integrator geführt. Die Anfangsbedingung kann direkt im Integrator vergeben werden.

Wird hingegen die Potentialvariable in einen Integrator geführt, so erhalten wir den (generalisierten) Impuls (siehe [Abbildung 5.87](#)).

5.9.2. Modulierte Elemente zur reversiblen Energieumwandlung

Zu den am Häufigsten modulierten Elementen zählen Transformatoren und Gyratoren. Wird beispielsweise bei einem Transformator das Übersetzungsverhältnis kontinuierlich verändert, so lässt sich damit bereits ein ideales stufenloses Getriebe realisieren.

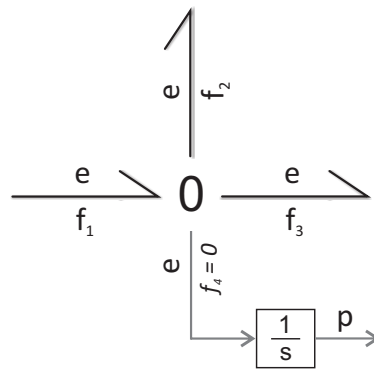


Abb. 5.87.: Sensor zur Ermittlung des (generalisierten) Impulses. Dazu wird die Potentialvariable in einen Integrator geführt.

Modulierte Transformatoren

Der Bondgraph des modulierten Transformators ist in [Abbildung 5.88](#) veranschaulicht.

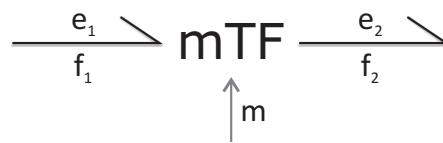


Abb. 5.88.: Modulierter Transformator, wobei m eine beliebige kontinuierliche Funktion sein kann $m = f(t)$.

Modulierte Gyratoren

Der Bondgraph des modulierten Gyratoren ist in [Abbildung 5.89](#) veranschaulicht.

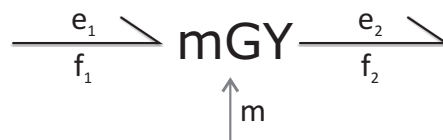


Abb. 5.89.: Modulierter Gyrtator, wobei m eine beliebige kontinuierliche Funktion sein kann $m = f(t)$.

5.9.3. Modulierte Quellen

Bei modulierten Quellen werden anstatt konstanter Werte bzw. vorgegebener Werte, externe Parameter verwendet. Sobald es sich um eine modulierte Quelle handelt wird ein zusätzliches Kürzel hinzugefügt.

- modulierte Potentialquelle: $Se \rightarrow mSe$
- modulierte Flussquelle: $Sf \rightarrow mSf$

Die modulierten Quellen sind in [Abbildung 5.90](#) dargestellt.

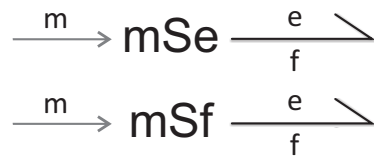


Abb. 5.90.: Modulierte Quellen. Oben: modulierte Potentialquelle. Unten: modulierte Flussquelle. Wobei m eine beliebige kontinuierliche Funktion sein kann $m = f(t)$.

5.9.4. Beispiele

Im folgenden Abschnitt wird der Umgang mit Sensoren und modulierten Elementen anhand zweier Beispiele gelernt. Zu Beginn wird eine elektrische Schaltung mit einer modulierten Stromquelle, welche ihr Eingangssignal von einer abgegriffenen Spannung erhält, modelliert. Anschließend wird ein mechanischer Schockabsorber modelliert. Dieser wird nicht fremderregt sondern über Anfangspositionen ausgelenkt.

Elektrische Schaltung mit modellierter Stromquelle

Es soll der Bondgraph der elektrischen Schaltung aus [Abbildung 5.91](#) erstellt werden.

Mittlerweile sollte die Erstellung von Bondgraphen elektrischer Schaltung kein allzu großes Problem mehr darstellen. Der Einfachheit halber nehmen wir zu Beginn an, es handle sich um eine gewöhnliche Stromquelle. Der Bondgraph ist in [Abbildung 5.92](#) veranschaulicht.

Nun soll die gewöhnliche Stromquelle durch eine modulierte ersetzt werden. Hierzu muss die Spannung u_C mit einem Sensor abgegriffen werden, dann mit dem Faktor 4 verstärkt werden und schlussendlich der Stromquelle zugeführt werden. Nachdem der Strom i_4 aber in die Stromquelle hineinfließt, muss entsprechend ein Vorzeichen ange-

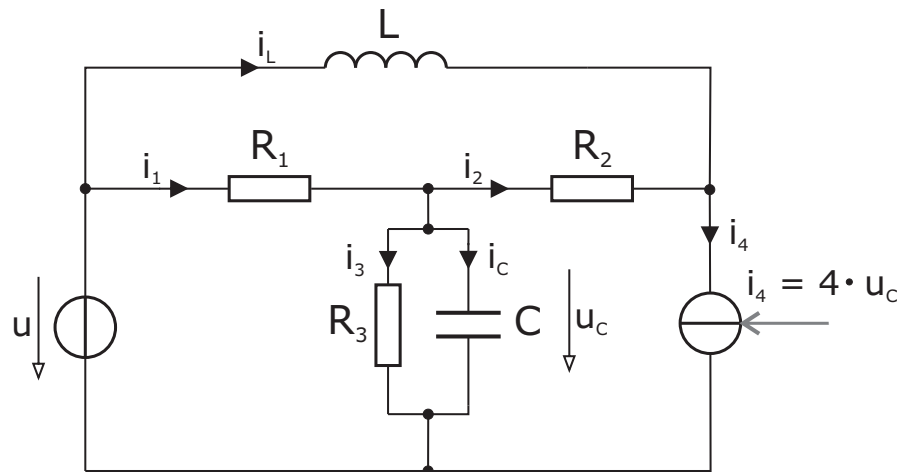


Abb. 5.91.: Elektrische Schaltung mit modulierter Stromquelle.

passt werden. Dazu wird der Faktor von 4 auf -4 geändert. Der vollständige Bondgraph ist in [Abbildung 5.93](#) dargestellt.

Mechanischer Schockabsorber

Als nächstes wird ein mechanischer Schockabsorber modelliert welcher in [Abbildung 5.94](#) dargestellt ist.

Anfangsbedingungen:

- Länge der ersten Feder: $x_{0,k1} = 2m$
- Anfangsposition der Masse m_1 : $x_{0,m1} = 2.5m$, d.h. dass die Feder um 0.5m ausgelenkt wird.
- Länge der zweiten Feder: $x_{0,k1} = 0.5m$
- Anfangsposition der Masse m_2 : $x_{0,m2} = 3.0m$, d.h. dass die Feder nicht ausgelenkt wird.

Erstellen wir auch hier wiederum den Bondgraphen ohne modellierte Elemente (siehe [Abbildung 5.95](#)).

Wie können im Bondgraphen aus [Abbildung 5.95](#) die Anfangsbedingungen vorgegeben werden?

Mit den bisherigen Elementen ist es nicht möglich der Feder eine Anfangslänge vorzugeben. An dieser Stelle sei nochmals explizit erwähnt, dass Bondgraphen, so wie wir sie bis jetzt kennengelernt haben, ausschließlich zur Beschreibung physikalischer

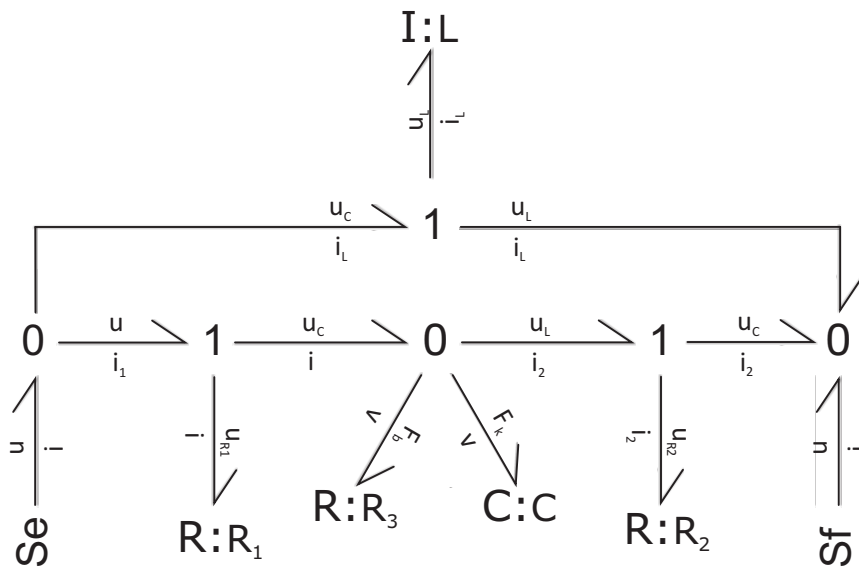


Abb. 5.92.: Bondgraph der elektrischen Schaltung wobei die modulierte Stromquelle als gewöhnliche Stromquelle angenommen wurde.

Vorgänge geeignet sind. Um eine geometrische Beschreibung zu ermöglichen, müssen die Elemente modifiziert werden. Wir können zwar der Masse m_1 eine Anfangsposition vorgeben, indem wir an der 1-Junction den Fluss messen und diesen anschließend in einen Integrator führen in dem die Anfangsbedingungen vorgegeben werden können. Dadurch wird die Feder jedoch nicht ausgelenkt sondern es wird definiert, dass die Länge der ungedehnten Feder eben genau der Anfangsposition entspricht. Natürlich können wir dem System, um es zu testen eine Kraft vorgeben. Mit den in [Unterabschnitt 5.9.3](#) eingeführten modellierten Quellen können wir mittlerweile sogar einen Kraftpuls aufschalten, was einem Stoß entspricht. Würden wir eine Kraft mit konstantem Wert anlegen so würde sich unser System für alle Zeiten in eine Richtung bewegen was eher unrealistisch ist. Aus diesem Grund ist es für mechanische Systeme durchaus sinnvoll diese so zu gestalten, dass auch geometrische Beschreibungen möglich sind. Dazu muss das C-Element modifiziert werden. Die Gleichung zur Potentialspeicherung aus [Tabelle 4.2](#), welche folgendermaßen umgeformt werden kann

$$f(t) = \beta \cdot \frac{de(t)}{dt} \quad (5.88)$$

muss durch eine lineare Gleichung ersetzt werden. Wird diese Gleichung nun integriert

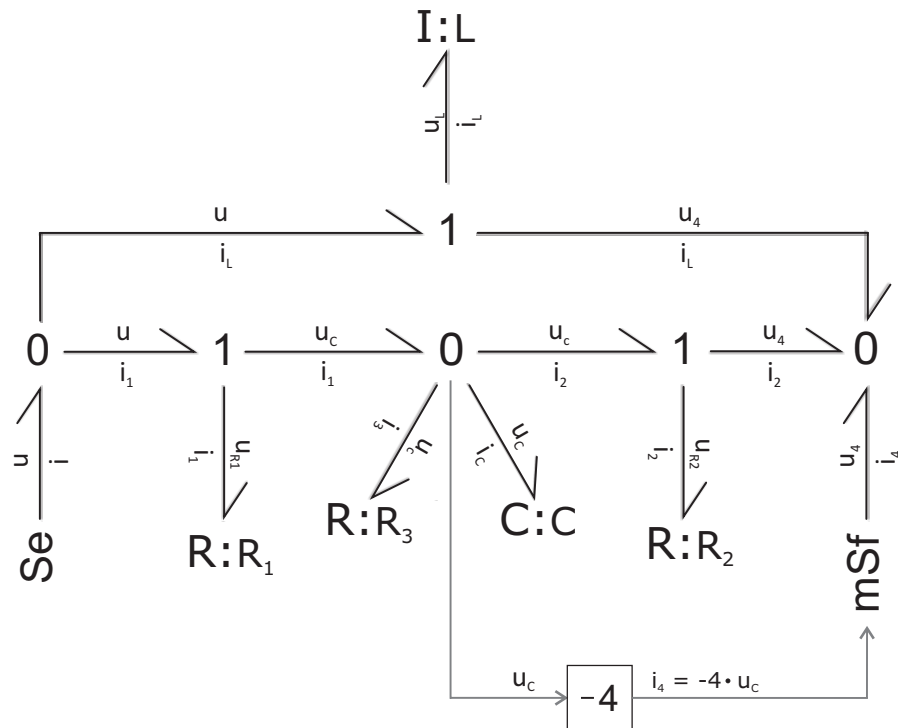


Abb. 5.93.: Vollständiger Bondgraph der elektrischen Schaltung.

so erhalten wir eine lineare Gleichung.

$$\int f(t) \cdot dt = \beta \cdot e(t)$$

$$q(t) + q_0 = \beta \cdot e(t) \tag{5.89}$$

Diese Gleichung entspricht bei mechanischen Systemen der Gleichung einer Feder ($F = k \cdot x$). Die Linke Feder des Bondgraphen aus [Abbildung 5.95](#) muss daher wie folgt modifiziert werden (siehe [Abbildung 5.96](#)).

Zuerst wird die Geschwindigkeit v_1 der Masse m_1 ermittelt. Diese wird dann in einen Integrator geführt welcher die Position ($q(t) = x_1(t)$) der Masse berechnet. Im Integrator selbst wird die Anfangsposition $x_{0,m1}$ der Masse definiert was der Auslenkung der Feder entspricht. Mit $x_{0,k1}$ kann zusätzlich die Länge der ungedehnten Feder spezifiziert werden. Die Differenz Δx beider Werte wird mit der Federkonstante k_1 multipliziert was die Federkraft F_{k1} ergibt. Da diese Kraft von der aktuellen Position $x_1(t)$ der Masse m_1 abhängt muss diese nun mittels einer modellierten Potentialquelle dem System hinzugefügt werden.

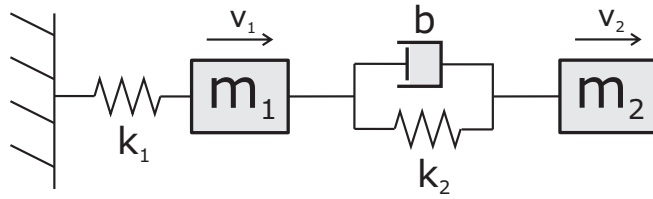


Abb. 5.94.: Mechanischer Schockabsorber.

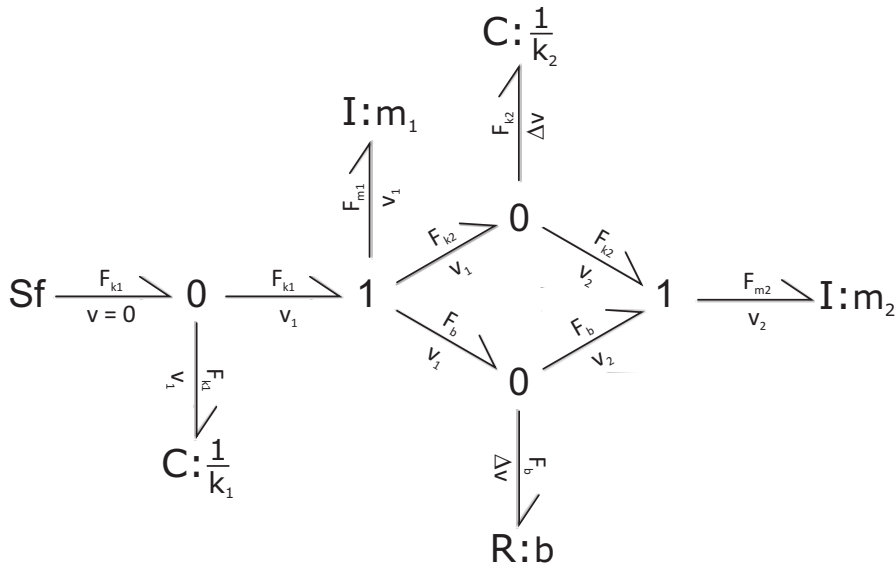


Abb. 5.95.: Bondgraph des Schockabsorbers ohne modellierte Elemente. Die Wand wurde mit einer Flussquelle $S_f = 0$ modelliert.

Werden nun dieselben Schritte für die zweite Feder durchgeführt, so erhalten wir den vollständigen Bondgraphen des Schockabsorbers welcher in [Abbildung 5.97](#) abgebildet ist.

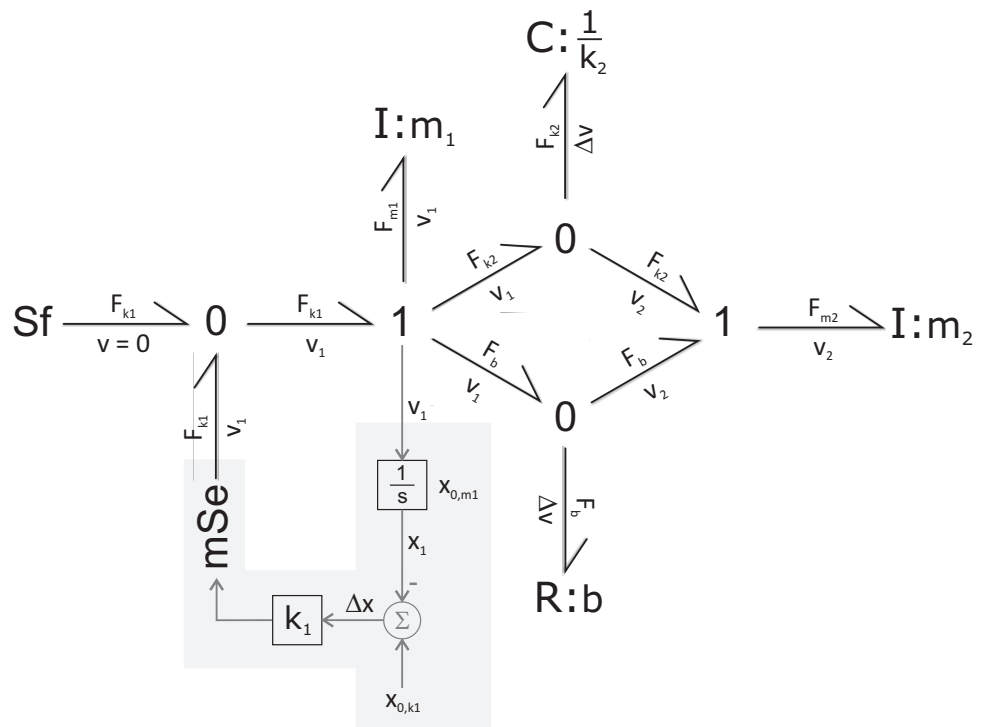


Abb. 5.96.: Bondgraph des Schockabsorbers mit modifizierter Feder (grau hinterlegt).

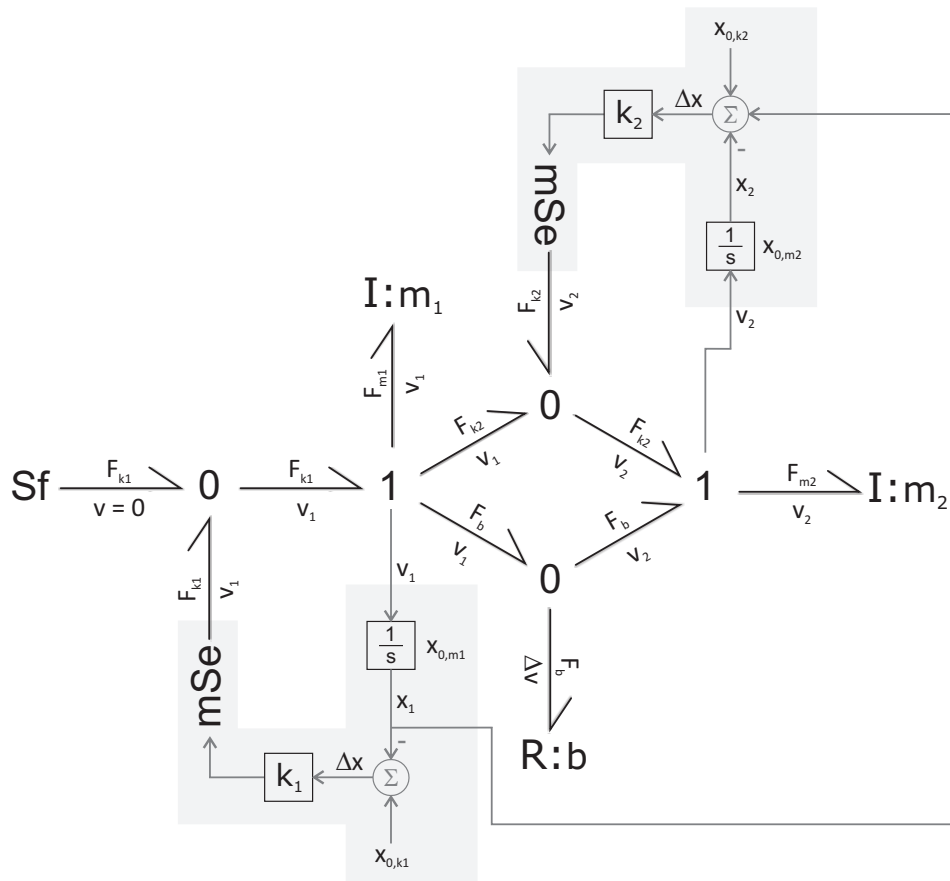


Abb. 5.97.: Vollständig modifizierter Bondgraph des Schockabsorbers.

6. Objektorientierte Modellierung

Dieses Kapitel beleuchtet kurz die geschichtliche Entwicklung von Simulationsprogrammen und den Weg bis zum heutigen Stand der Technik - der objektorientierten Modellierung. Dabei wird auf die Eigenschaften und Vorteile dieser, sowie deren „Verwandtschaft“ zu der objektorientierten Programmierung eingegangen. Weiters werden die Voraussetzungen für die komponentenbasierte und domänenübergreifende Modellierung sowie deren Vorteile erläutert. Beispiele werden anhand der open-source Sprache Modelica diskutiert, auf deren Fähigkeiten später überblickshaft eingegangen wird. Dabei soll der Leser dieses Kapitel weniger als Nachschlagewerk verwenden, um etwa Details des Syntax nachzulesen, sondern eher, um einen Überblick über die Fähigkeiten und somit die sinnvollen Einsatzgebiete der objektorientierten Modellierung zu gewinnen. Abschließend wird eine der grundlegenden Eigenheiten der gleichungsbasierten Modellierung genauer erläutert: die symbolische Vorverarbeitung.

6.1. Zur Geschichte der Modellierung

Die ersten verfügbaren Simulationsprogramme haben versucht die bis dahin in der Modellierung und Simulation gängigen Methoden widerzuspiegeln. Sie waren also mehr oder weniger ein digitales Abbild der analogen Simulationstechnik, in welcher Modelle aus Addierern, Integratoren, Multiplikatoren und Potentiometern zusammengesetzt wurden. Es hat einige Jahre gedauert, bis festgestellt wurde, dass die Erstellung analoger Programme nicht sehr komfortabel ist und somit auch die Abbildung der analogen Methoden in die digitale Welt eine unnötige Erschwerung der Modellierung mit sich brachte. Es wurde nun begonnen die weitaus größeren Möglichkeiten der digitalen Technik einzusetzen, um die Modellierung zu erleichtern [CE93].

Die nächste Generation von Simulatoren verwendete die Gegebenheiten, welche sich aus den numerischen Lösungsverfahren ergeben, als Startpunkt. Wie sich herausstellt, sind die meisten dieser Lösungsverfahren dafür erstellt Zustandsraummodelle zu lösen. Diese liegen dann allgemein in der nichtlinearen Form

$$\dot{x} = f(x, u, t) \tag{6.1}$$

vor. Um die Simulation effizienter zu machen führen verschiedene Simulationstools Hilfsvariablen z ein, welche z.B. mehrfach verwendete Zwischenergebnisse speichern.

Daraus ergibt sich die Form

$$\dot{x} = f(x, z, u, t) \tag{6.2}$$

$$z = g(x, z, u, t) \tag{6.3}$$

wobei die Hilfsvariablen z nicht in einer gegenseitigen Wechselwirkung stehen dürfen, sprich nicht Teil einer algebraischen Schleife sein dürfen¹. Zusätzlich wurden Algorithmen inkludiert, welche eine Sortierung der Zuweisungen² vornahmen. Diese Algorithmen konnten im Allgemeinen nicht mit algebraischen Schleifen umgehen. Trotzdem erlaubt dies dem Modellierer die Zuweisungen in einer beliebigen Reihenfolge zu spezifizieren, da diese nicht mehr sequenziell abgearbeitet wurden. Damit wurde auch die Verwendung von Makros ermöglicht, was die Definition von Subsystemen in einer kompakten Form möglich machte [CE93]. Eine ausführlichere Übersicht über die Entwicklung der zeitkontinuierlichen Simulation wird in [sEM98] gegeben.

6.2. Motivation

Dies stellt im Groben auch den heute aktuellen Stand der signalflussbasierten Simulationstools wie z.B. Simulink dar. Diese wurden um einige Fähigkeiten erweitert wie z.B. das Lösen von algebraischen Schleifen mit einigen Einschränkungen³. Eine grundsätzliche Eigenschaft der signalflussbasierenden Modellierung ist es, dass Signale gerichtet sind. Simulink Blöcke etwa haben fix vorgegebenen Ein- und Ausgänge. Sie entsprechen also genau genommen eher einer aus der Programmierung bekannten Zuweisung als einer mathematischen Gleichung. Zur Simulation eines Modells müssen daher immer die sich aus der Mathematik und Physik ergebenden Gleichungssysteme in einzelene Zuweisungen überführt werden⁴. Dieser Schritt muss bei der Verwendung eines signalflussbasierten Simulators manuell durchgeführt werden, was aufwändig und fehleranfällig ist.

Anhand eines Beispiels sollen die erwähnten Einschränkungen dieser Art der Modellierung aufgezeigt werden. Dazu wird im Folgenden ein einfaches mechanisches Beispiel als Blockschaltbild dargestellt, und anschließend um einen zuerst vernachlässigten Teil erweitert. Dabei soll klar werden, dass einige auf den ersten Blick schwer ersichtliche

¹Mehr zu algebraischen Schleifen in [Unterabschnitt 6.9.2](#) auf Seite 204

²Zuweisungen sind aus der Programmierung bekannt. Dabei ist fix festgelegt, welche Variable aus den anderen in dieser Programmzeile errechnet wird. Üblicherweise ist es die Variable links vom Gleichheitszeichen. Dies ist aus den gängigen Programmiersprachen bekannt, unterscheidet sich aber grundsätzlich von einer mathematischen Gleichung.

³In der Schleife enthaltene Blöcke dürfen laut der aktuellen (August 2011) Mathworks Dokumentation keine der folgenden Elemente enthalten: Blöcke mit diskreten Ausgängen, Blöcke mit komplexen oder nicht-double Werten, Unstetigkeiten, Stateflow-Charts

⁴Dieser Vorgang wird in der Modellierung als horizontale Sortierung des Gleichungssystems bezeichnet.

Schritte nötig sind, um zu einem korrekten Ergebnis zu kommen. [Abbildung 6.1](#) zeigt das anfänglich modellierte System.

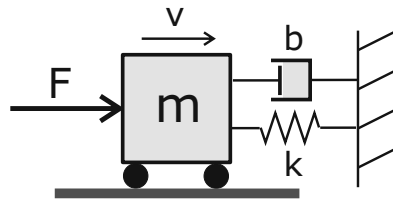


Abb. 6.1.: Mechanisches System das zu modellieren ist.

Über das Aufstellen von Komponentengleichungen für Masse, Feder und Dämpfer sowie das D’Lambert Prinzip ergeben sich die folgenden Gleichungen als mathematische Beschreibung für das System.

$$F = f(t) = u(t) \quad (6.4)$$

$$v = \frac{1}{m} \int F_m \cdot dt \quad (6.5)$$

$$F_k = k \cdot x = k \cdot \int v \cdot dt \quad (6.6)$$

$$F_b = b \cdot v \quad (6.7)$$

$$0 = F - F_b - F_k - F_m \quad (6.8)$$

Auf die Überführung des Systems in ein Blockschaltbild wird hier nicht weiter eingegangen, da diese in [Abschnitt 3.5](#) ausführlich erläutert wurde⁵.

Erkennt man während dem Modellierungsprozess oder während dem Vergleich mit Messergebnissen, dass das System zu stark vereinfacht wurde, muss das Modell erweitert werden. In unserem Fall, soll links von der Masse m eine zweite Masse m_2 angefügt werden, welche über eine Feder mit der Federkonstante k_2 an die Masse m gekoppelt wird. Diese wird wiederum in ein Blockschaltbild überführt (siehe [Abbildung 6.5](#)).

Die beiden Teilsysteme aus [Abbildung 6.1](#) und [Abb. 6.3\(a\)](#) sollen nun zu einem Gesamtsystem zusammengefügt werden. Dieses ist in [Abbildung 6.5](#) dargestellt.

Es ist daher naheliegend die Federkraft F_{k2} ([Abb. 6.3\(a\)](#)) gleich der Eingangskraft F in das ursprüngliche System zu setzen ([Abbildung 6.1](#)). Das daraus resultierende Blockschaltbild ist in [Abbildung 6.6](#) dargestellt.

⁵Genau genommen handelt es sich bei dieser Tätigkeit nicht um die Modellierung des Systems. Die Modellierung ist nach dem aufstellen des Gleichungssystems (6.4) - (6.8) abgeschlossen. Mit der Umformung in etwa ein Blockschaltbild wird dieses Gleichungssystem „nur“ noch in ein für einen signalflussbasierten Simulator verständlich formuliert.

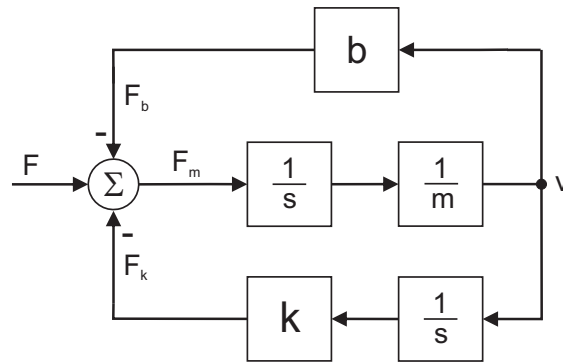


Abb. 6.2.: Blockschaltbild des Feder-Masse-Schwingers.

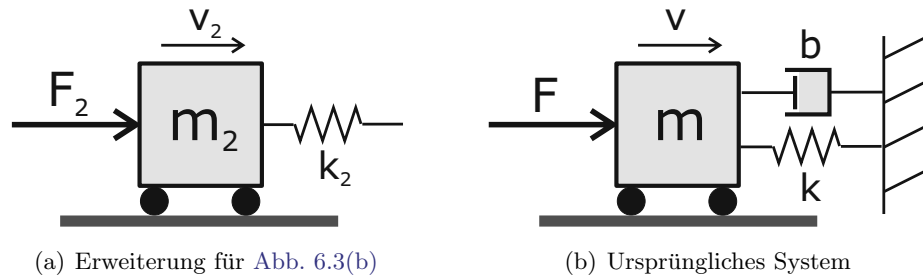


Abb. 6.3.: Modell der Erweiterung des Feder-Masse-Schwingers.

Es ergibt sich eine klare Abtrennung zwischen den beiden Modellen über die Kraft. Diese ist allerdings nicht die einzige physikalische Größe, welche bei der Zusammenschaltung der beiden Teile die physikalische Wechselwirkung beschreibt. Man hat also in der Modellierung einen Fehler begangen. Dieser Fehler besteht darin, dass auch die Position bzw. die Geschwindigkeit der Masse m einen Einfluss auf die Kraft F_{k_2} hat. Im Modell in [Abbildung 6.3](#), wird die für die Kraft verantwortliche Position x_2 nur durch die Masse m_2 bestimmt. Wir haben die Feder k_2 also ohne uns klar zu sein an einer „Wand“ fixiert und lassen die Kraft, welche auf die (unbewegliche) Wand wirkt nun auf die Masse m wirken. Dabei wird die Länge der Feder und somit die resultierende Kraft F_{k_2} falsch berechnet, wodurch ein falsches Ergebnis berechnet wird. Dieser Tatsache wurde in [Abbildung 6.7](#) Rechnung getragen.

Um den Fehler zu finden, bieten sich zwei Möglichkeiten an. Entweder kann das Gesamtsystem noch einmal modelliert werden, was für große Systeme äußerst aufwändig sein kann. Die andere Möglichkeit ist, den Fehler im Blockschaltbild aufgrund von physikalischen Überlegungen zu suchen. Im vorliegenden Beispiel ist die Korrektur des Blockschaltbildes relativ einfach. Es ist sehr gut ersichtlich, dass die Feder ei-

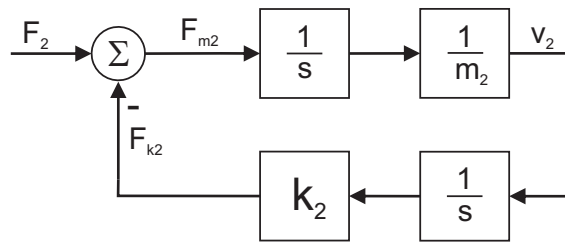


Abb. 6.4.: Blockschaltbild der Erweiterung.

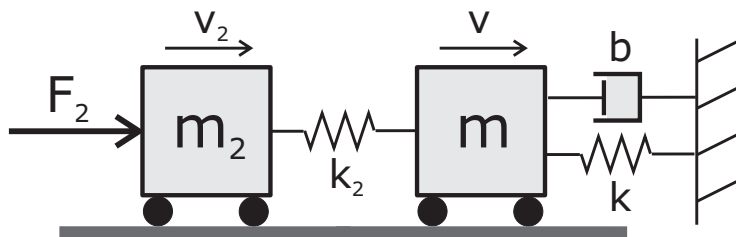


Abb. 6.5.: Zu modellierendes Gesamtsystem.

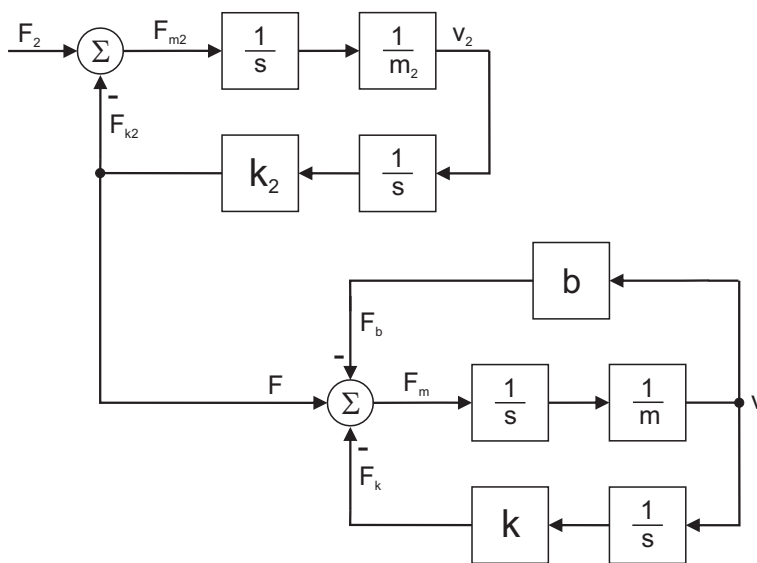


Abb. 6.6.: Falsch zusammengesetztes Blockschaltbild.

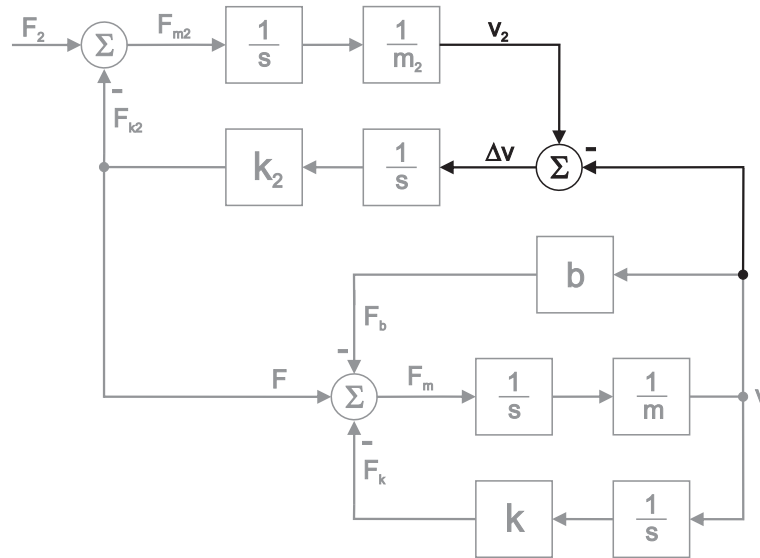


Abb. 6.7.: Korrektes Blockschaltbild.

ne Geschwindigkeitsdifferenz erfährt, welche sich aus $\Delta v = v_2 - v_1$ ergibt. Dadurch ändert sich auch die Federkraft F_{k2} , da die Feder nicht die Länge x_2 , sondern Δx als Längenänderung erfährt.

Dieses Beispiel soll aufzeigen, dass es zwar möglich ist, physikalische Systeme in signalflussbasierte Simulatoren zu berechnen, aber durch die Einschränkungen, welche sie prinzipbedingt mit sich bringen, die Modellierung oft unnatürlich ist. Die Kombination zweier Subsysteme sollte nicht die Modifikation eines der beiden Systeme erfordern. Sind die Systeme jedoch komplexer, passiert es oft, dass solche Änderungen und die dadurch entstehenden Rückwirkungen vergessen werden. Auch die dadurch entstehende Fehler sind in den Simulationsergebnissen nicht immer ohne weiteres erkennbar.

6.3. Modelica Grundlagen

Es gibt verschiedene Programme bzw. Modellierungssprachen, welche eine direkte Modellierung verschiedener technischer Domänen erlauben. Dazu zählen unter anderem die folgenden:

- Saber von Synopsys
- Simplorer von Ansoft
- SimScape von Mathworks

- 20sim von Controllab Products B.V.
- Auf Modelica basierende Programme
 - Dymola von Dynasim (mittlerweile Dassault Systems)
 - MapleSim von Maplesoft
 - MathModelica von Mathcore
 - SimulationX von ITI
 - OpenModelica von der LIU (Linköping University)

Da Modelica der de-facto Standard der objektorientierten Modellierung ist und die Sprachdefinition unter einer open-source Lizenz zur Verfügung steht, wurde diese Sprache als Basis für die Umsetzung der Beispiele gewählt. Wie aus der oben angeführten Aufzählung zu erkennen ist, sind mehrere Simulationsumgebungen verfügbar, welche die Modelica Sprache umsetzen, was eine gewisse Unabhängigkeit von dem Softwareanbieter ermöglicht.

6.3.1. Was ist Modelica eigentlich?

Modelica ist der Versuch eine gemeinsame und öffentlich verfügbare Sprachdefinition für die objektorientierte Modellierung zu schaffen, auf deren Basis verschiedene Simulationsumgebungen arbeiten können. Die Modelica Sprachdefinition wird durch die im Jahr 2000 gegründete Modelica Association gewartet und stetig erweitert. Sie baut auf den Erfahrungen vieler Versuche der Definition einer objektorientierten Modellierungssprache auf (Allan, Dymola, NMF, ObjectMath, Omola, SIDOPS+, Smile) und wird von Computerspezialisten sowie Modellierungsspezialisten aus den verschiedensten technischen Domänen weiterentwickelt. Zusätzlich kümmert sich die Modelica Association um die Erstellung der Modelica Standard Library, welche viele der grundlegenden Elemente für die Modellierung technischer Systeme zur Verfügung stellt.

Die Erstellung aller weiterer notwendigen Teile, welche eine komfortable Simulation möglich machen, bleibt den Erstellern der jeweiligen Simulationsumgebung überlassen. Dazu gehören die Erstellung einer GUI, möglicher Visualisierungstools, die Umwandlung von der Modelica Hochsprache in einen ausführbaren Code, die Implementierung verschiedener Lösungsalgorithmen usw. Einige der verfügbaren Simulationsumgebungen sind im vorherigen Abschnitt aufgelistet. Dabei sind alle der erwähnten Programme bis auf OpenModelica kostenpflichtig. Da es sich bei Modelica „nur“ um eine Sprachdefinition handelt, muss eines der erwähnten Programme verwendet werden, um Simulieren zu können. Dabei werden von der Simulationsumgebung etwa GUI, Integrationsalgorithmen, Anzeige von Simulationsergebnissen, 3D Visualisierung etc. übernommen. Am 02. April 2010 wurde die Modelica 3.2 Spezifikation publiziert, worauf auch die hier verwendeten Code-Segmente basieren.

6.3.2. Modelica und Dymola

Die derzeit am weitesten fortgeschrittene Entwicklungsumgebung welche auf Modelica basiert ist Dymola. Dabei bezieht sich „am weitesten fortgeschritten“ vor allem auf eine der Hauptaufgaben einer Modelica Modellierungsumgebung: der symbolischen Vorverarbeitung. Diese unterstützt im Unterschied zu manchen anderen Modellierungsumgebungen den vollen Modelica Sprachschatz und auch die komplette Standard Library. Da die symbolische Vorverarbeitung großen Einfluss auf die Simulationsgeschwindigkeit haben kann, und die von Dymola eingesetzte Version im Normalfall äußerst effiziente Ergebnisse liefert, werden vor allem komplexe Modelle in Dymola oft deutlich schneller als in anderen Modellierungsumgebungen simuliert. Dadurch ist Dymola auch die in der Industrie am weitesten verbreitetste Software. Andere Programme wie SimulationX bieten Vorteile in der Auswertung der Simulationsdaten und einen erweiterten Umfang der standardmäßig verfügbaren Modelle.

Die Entwicklung von Modelica geht relativ schnell voran und daher werden auch regelmäßig neue Versionen von Dymola vorgestellt. Obwohl sich dabei an dem grundlegenden Eigenschaften der Software selten etwas ändert, macht es wenig Sinn in diesem Dokument ein weiteres Einstiegsdokument für Dymola zu verfassen. Es wird daher auf ein Dokument des Herstellers verwiesen welches einen relativ schnellen Einstieg in die Welt von Modelica ermöglicht. Dieses trägt den Titel „Getting started with Dymola“ und ist aktuell⁶ unter folgendem Link zu finden. <http://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/Getting-Started.pdf>

Ein Umstieg auf eine Modelica-Umgebung anderer Hersteller sollte prinzipiell keine großen Schwierigkeiten bereiten, da viele grundsätzlichen Funktionen bekannt sein sollten. Ein zusätzlicher Aspekt für den Einsatz von Dymola ist eine aktuelle Entwicklung, welche das sehr verbreitete 3D-CAD System CATIA mit den Fähigkeiten von Dymola verbinden soll. Dadurch wird eine leichtere Vernetzung von Konstruktion und Modellierung und dynamischer Simulation ermöglicht. Das Produkt dazu wird ebenfalls vom neuen Hersteller von Dymola und CATIA (Dassault Systems) vertrieben und nennt sich CATIA Systems.

6.3.3. Umsetzung in Modelica

Hier soll das in [Abschnitt 6.2](#) aufgezeigte Beispiel in Modelica umgesetzt werden. Dazu müssen zuerst einige Grundlagen zu Modlica erläutert werden. Die grundlegende Struktur jedes Modelica Modells ist die im folgenden Codesegment beschriebene.

```
1 | model name  
2 |
```

⁶erstellt am 28. September 2011

```

3 equation
4
5 end name;
```

Code 6.1: Grundlegende Struktur eines Modells in Modelica

Ein Modell besteht aus Unbekannten und Gleichungen. Unbekannte werden direkt nach dem Modellnamen deklariert. Das in der Folge verwendete Keyword **Parameter** signalisiert dabei, dass es sich bei diesem Wert um eine Variable handelt, welche über die gesamte Simulationsdauer konstant bleibt und auf welche der Benutzer direkt Einfluss nehmen kann. Diesem Parameter kann auch direkt ein Wert zugewiesen werden, wie es in [Code 6.2](#) zu sehen ist. Gleichungen, welche das Verhalten des zu modellierenden Systems beschreiben, werden im Abschnitt **equation** definiert. Dabei ist wichtig zu erkennen, dass es sich in diesem Abschnitt wirklich um Gleichungen handelt⁷. Diese werden vor der Simulation automatisch über aufwändige Algorithmen in Zuweisungen überführt. Mehr zu diesem Vorgang, welcher symbolische Vorverarbeitung genannt wird, wird in [Abschnitt 6.9](#) erläutert. Ein wichtiger Punkt dabei ist, dass die Anzahl der Unbekannten und der Gleichungen identisch sein muss. Ansonsten kann kein lösbares Gleichungssystem erstellt werden. Variablen welchen direkt ein Wert zugewiesen wird, sind dabei in dieser Rechnung als neutral anzusehen, da zu der Unbekannten (die Variable) direkt eine Gleichung definiert wird. Ein typisches Beispiel dafür sind Parameter, welchen oft in der Zeile ihrer Definition schon ein Wert zugewiesen wird.

Beginnen wir nun mit der Modellierung des Feder-Masse-Schwingers aus [Abbildung 6.1](#) von Seite 139. In Modelica erhalten wir die in [Code 6.2](#) dargestellten Parameter und Unbekannten. Bei der Implementierung der Gleichungen (6.4) bis (6.8) in Modelica stellt sich allerdings heraus, dass wir sie in einer differentiellen Form verwenden werden müssen, da Modelica keinen Operator für eine Integration zur Verfügung stellt. Für die zeitliche Ableitung hingegen wird der Operator `der()` zur Verfügung gestellt.

Die Gleichungen welche ein Integral enthalten werden also über einfaches Differenzieren umgeformt und wir erhalten den folgenden Satz an Gleichungen.

$$F_m = m \cdot \frac{dv}{dt} \quad (6.9)$$

$$v = \frac{1}{k} \cdot \frac{dF_k}{dt} \quad (6.10)$$

$$F_b = b \cdot v \quad (6.11)$$

$$0 = F - F_b - F_k - F_m \quad (6.12)$$

Die komplette Implementierung in Modelica ergibt sich also zu [Code 6.2](#). Dabei gibt `Real` an, dass es sich bei der Variable um eine Gleitkommazahl handelt. Es wären hier

⁷Es wird also nicht zwangsläufig die auf der linken Seite des Gleichheitszeichens stehende Variable berechnet.

auch `Int` und `Bool` verfügbar, welche aber in diesem Modell nicht sinnvoll anwendbar sind, da es sich um kontinuierliche Größen handelt.

```
1 model MechanicalSystem
2   // Parameters
3   parameter Real F = 10;
4   parameter Real m = 1;
5   parameter Real b = 2;
6   parameter Real k = 100;
7   // Unknowns
8   Real Fm;
9   Real Fb;
10  Real Fk;
11  Real v;
12  equation
13    Fm = m * der(v);
14    v = 1/k * der(Fk);
15    Fb = b * v;
16    F - Fm - Fk - Fb = 0;
17 end MechanicalSystem;
```

Code 6.2: Einfacher Feder-Masse-Schwinger in Modelica

Wollen wir nun die in [Abbildung 6.3](#) auf Seite 140 dargestellte Änderung in Modelica durchführen, können wir zu dem bestehenden Modell die folgenden Gleichungen hinzufügen, ohne das ursprüngliche Modell anpassen zu müssen.

$$F_{m2} = m_2 \cdot \frac{dv_2}{dt} \quad (6.13)$$

$$\Delta v = \frac{1}{k_2} \cdot \frac{dF_{k2}}{dt} \quad (6.14)$$

$$F = F_{k2} \quad (6.15)$$

$$\Delta v = v_2 - v \quad (6.16)$$

Hier wird erstmals eindeutig klar, dass es sich hierbei um Gleichungen handeln muss, da nicht zweimal Δv berechnet werden kann. In diesem Fall wird aus [Gleichung 6.14](#) nicht Δv berechnet, sondern die Gleichung wird von der Simulationsumgebung nach dF_{k2}/dt umgeformt.

Das gesamte Modell für den Zwei-Massen-Schwinger ergibt sich also zu [Code 6.3](#) und beschreibt das in [Abbildung 6.5](#) auf Seite 141 dargestellte Modell. Dabei ist wichtig zu erkennen, dass die Variable F , welche im ursprünglichen Modell der Eingang in das System war und daher als Parameter angegeben war zu einer internen Variable wird. Der neue Eingang heißt F_2 . Weiters wird ersichtlich, dass das ursprüngliche Modell nicht angepasst werden musste.


```
1 model MechanicalSystem2
2   // Parameters
3   parameter Real F2 = 10;
4   parameter Real m = 1;
5   parameter Real m2 = 0.2;
6   parameter Real b = 2;
7   parameter Real k = 100;
8   parameter Real k2 = 500;
9   // Unknowns
10  Real F;
11  Real Fm;
12  Real Fm2;
13  Real Fb;
14  Real Fk;
15  Real Fk2;
16  Real v;
17  Real v2;
18  Real delta_v;
19  equation
20    Fm = m * der(v);
21    v = 1/k * der(Fk);
22    Fb = b * v;
23    F - Fm - Fk - Fb = 0;
24    Fm2 = m2 * der(v2);
25    delta_v = 1/k2 * der(Fk2);
26    F = Fk2;
27    delta_v = v2-v;
28    F2 - Fm2 - Fk2 = 0;
29  end MechanicalSystem2;
```

Code 6.3: Zwei-Massen-Schwinger in Modelica

An Code 6.3 fällt auf, dass einige Gleichungen bzw. genauer gesagt deren Struktur mit anderen Variablennamen und Parametern doppelt vorhanden sind. Dies trifft für alle der Elemente zu, welche in dem erweiterten Modell mehrfach vorkommen, also die Massen und Federn. Dies entspricht nicht dem komponentenorientierten Ansatz den man mit Modelica eigentlich verfolgen sollte. Wir sollten versuchen für grundlegende und öfters verwendete Komponenten allgemeingültige Klassen zu definieren und die mehrfach vorhandenen Komponenten über Instanzierung dieser Klassen in Objekten abzubilden. Die Begriffe Klasse und Objekte werden in [Abschnitt 6.5](#) ab Seite 157 genauer diskutiert. Vorher sollen noch einige eher „kosmetische“ Anpassungen vorgestellt werden, welche Modelica zur Verfügung stellt, um die Modelle verständlicher zu gestalten und die Auswertung zu erleichtern sowie Fehler vorzubeugen.

6.3.4. Modell-Kosmetik in Modelica

In diesem Abschnitt soll gezeigt werden welche Features Modelica bietet, um dem Modellierer das Leben zu erleichtern. Es werden allerdings auch wichtige Aspekte wie die Initialisierung von Gleichungen angesprochen. Verwendet wird dafür [Code 6.2](#).

Als erstes soll gezeigt werden, wie Variablen mit Kommentaren versehen werden können. Dabei handelt es sich nicht um Kommentare wie man sie aus der Programmierung kennt. Sie werden hinter die Variable in Anführungszeichen als Anmerkung geschrieben und später, abhängig von der Modellierungsumgebung, an verschiedenen Stellen zur Erläuterung der Variable angezeigt. Die Implementierung dazu ist in [Code 6.4](#) zu sehen. Kommentare wie in Programmiersprachen waren in [Code 6.2](#) durch einen doppelten Slash “//“ gekennzeichnet zu sehen. Sie scheinen sonst nirgends auf.

```
1 model MechanicalSystem
2   parameter Real F = 10 "Kraft";
3   parameter Real m = 1 "Masse";
4   parameter Real b = 2 "Dämpfungskonstante";
5   parameter Real k = 100 "Federkonstante";
6   ...
7 equation
8   ...
9 end MechanicalSystem;
```

Code 6.4: Kommentieren von Variablen

Eine weitere Eigenschaft von Variablen in Modelica ist, dass sie mit Einheiten versehen werden können. Dies ist in [Code 6.5](#) dargestellt. Genau wie die Kommentare können Einheiten in der Auswertung angezeigt werden. Dabei haben diese Einheiten nicht nur kosmetischen Nutzen, sondern können von der Modellierungsumgebung auch überprüft werden. Bei konsequent mit Einheiten versehenen Modellen führen verschiedene Modellierungsumgebungen Einheitenchecks in den definierten Gleichungen durch und geben Warnungen aus, falls ein Fehler gefunden wird.

```
1 model MechanicalSystem
2   parameter Real(unit = "N") F = 10 "Kraft";
3   parameter Real(unit = "kg") m = 1 "Masse";
4   parameter Real b = 2 "Dämpfungskonstante";
5   parameter Real k = 100 "Federkonstante";
6   ...
7 equation
8   ...
9 end MechanicalSystem;
```

Code 6.5: Variablen und Einheiten

Es ist allerdings auch möglich die in der Modelica Standardbibliothek vordefinierten Typen zu verwenden, um Variablen gleichzeitig einen Typ und eine Einheit zuzuweisen. So ist z.B. die Kraft in [Code 6.6](#) definiert.

```
1 type Force = Real (final quantity="Force", final unit="N");
```

Code 6.6: Variablen und Typen

Wird einer Variablen ein Type zugewiesen, wird ihr somit automatisch eine Einheit und ein Typ zugewiesen. Die in der Modelica Standard Library vordefinierten Typen sind in der Library `SIunits` abgelegt. Wie sie verwendet werden können ist in [Code 6.7](#) aufgezeigt. Da diese Variante etwas unübersichtlich ist, hat sich eine abgekürzte Alternative eingebürgert. Diese ist in [Code 6.8](#) dargestellt. Dabei wird die Variable `SI` eingeführt, welche das `import` Statement verwendet, um die langen Typenbezeichner abzukürzen.

```
1 model MechanicalSystem
2   parameter Modelica.SIunits.Force F = 10 "Force";
3   parameter Modelica.SIunits.Mass m = 1 "Mass";
4   parameter Modelica.SIunits.TranslationalDampingConstant b = 2 "
   Damping constant";
5   parameter Modelica.SIunits.TranslationalSpringConstant k = 100
   "Spring constant";
6   ...
7 equation
8   ...
9 end MechanicalSystem;
```

Code 6.7: Typen direkt aus der Modelica `SIunits` Library

```
1 model MechanicalSystem
2   import SI = Modelica.SIunits;
3
4   parameter SI.Force F = 10 "Force";
5   parameter SI.Mass m = 1 "Mass";
6   parameter SI.TranslationalDampingConstant b = 2 "Damping
   constant";
7   parameter SI.TranslationalSpringConstant k = 100 "Spring
   constant";
8   ...
9 equation
10  ...
11 end MechanicalSystem;
```

Code 6.8: Abgekürzte Version über den `import` Befehl

In dem verwendeten Beispiel (Code 6.2) wird direkt ersichtlich werden die Integratoren nicht direkt ersichtlich definiert. Etwas versteckt wird aber durch jede der verwendeten Ableitungen (`der()` Operator) eine Zustandsvariable und somit auch ein Integrator definiert. Integratoren können Initialwerte ungleich Null haben. Mit diesen Initialwerten kann die Vergangenheit der speichernden Elemente abgebildet werden⁸. In Modelica kann dieser Initialwert über den in Code 6.9 dargestellten „initial equation“ definiert werden⁹.

```
1 model MechanicalSystem
2   // Parameters
3   ...
4   // Unknowns
5   ...
6   initial equation
7     v = 0;
8     Fk = 0;
9   equation
10    ...
11 end MechanicalSystem;
```

Code 6.9: Definition von initial equations zur Startwertvorgabe

Die Initialgleichungen aus Code 6.9 können direkt in Code 6.2 eingebaut werden. Werden keine Initialgleichungen angegeben werden die Variablen üblicherweise automatisch mit Null initialisiert.

Neben den bereits angesprochenen Kommentaren, welche unter anderem zur Erläuterung der Simulationsergebnisse eingeblendet werden, bietet das `protected` Statement eine Möglichkeit Parameter und/oder Variablen, welche für den Anwender in der Auswertung nicht von Interesse sind, zu verstecken. Code 6.10 zeigt die Anwendung dieses Statements. Diese Variablen werden in der Simulation bzw. im Simulationsergebnis nicht direkt angezeigt. Außerdem ist ein Zugriff aus anderen Modellen auf diese Variablen dann nicht mehr gestattet.

```
1 model MechanicalSystem
2   // Parameters
3   ...
4   protected
5     SI.Force Fm;
6     SI.Force Fb;
7   public
8     SI.Force Fk;
```

⁸Mit dem Initialwert wird die bis zum Startzeitpunkt der Simulation aufsummierte bzw. aufintegrierte Eingangsvariable dargestellt.

⁹Hierzu bestehen noch alternative Möglichkeiten, welche auf Seite 178 in Abschnitt 6.7.6 beschrieben werden.

```

9   SI.Velocity v;
10 initial equation
11   ...
12 equation
13   ...
14 end MechanicalSystem;

```

Code 6.10: Public and protected Variablen

Neben den Einheiten, welche mit Variablen verknüpft werden können, ist es möglich einige weitere Attribute von Variablen festzulegen. Eine Übersicht ist in [Tabelle 6.1](#) aufgeführt.

Attribut	Bedeutung
min	Der Wert der Variable kann nicht unter den vorgegebenen Wert sinken. Eine Länge kann z.B. nicht negativ werden.
max	Der Wert der Variable kann nicht über den vorgegebenen Wert steigen.
start	Eine alternative Möglichkeit den Startwert einer Zustandsvariable anzugeben (siehe Abschnitt 6.7.6 auf Seite 178).
fixed	Wenn auf <code>true</code> gesetzt, wird der Startwert als fest vorgegebener Initialwert verwendet. Auf <code>false</code> gesetzt, wird der Wert als Schätzwert behandelt. Er ist also z.B. Startwert einer Iteration.
nominal	Gibt einen Wert für die Größenordnung vor, in welcher die Variable sich üblicherweise bewegt. Dadurch können bessere numerische Eigenschaften erreicht werden.
stateSelect	Ermöglicht eine Einflussnahme auf die Wahl der Zustandsvariablen. Diese Wahl kann großen Einfluss auf die Geschwindigkeit und Genauigkeit der Simulation haben. Sie kann etwa bei dreidimensionalen Systemen entscheidend sein und bei Simulationenzeiten die um Faktoren kleiner sind genauere Ergebnisse liefern. Möglichkeiten sind hier <code>StateSelect.never</code> , <code>.avoid</code> , <code>.default</code> , <code>.prefer</code> und <code>.always</code> .

Tab. 6.1.: Übersicht der möglichen Attribute von Variablen

6.4. Voraussetzungen für die komponentenbasierte Modellierung

Aus den im vorangegangenen Abschnitt erläuterten Einschränkungen, welche durch die signalflussbasierte Modellierung entstanden sind, sollen nun grundlegende Anforderungen an die objekt- oder geräteorientierte Modellierung aufgezeigt werden. Dabei wird im Speziellen auf Schnittstellen und rechnerische Kausalität eingegangen.

6.4.1. Schnittstellen

Durch die in [Abschnitt 6.2](#) aufgezeigten Schwierigkeiten wird eine Anforderung an eine physikalische Modellierungssprache deutlich. Es muss eine Möglichkeit geben Teilsysteme gleicher technischer Domänen ohne weiteren Aufwand miteinander verbinden zu können. Diese Anforderung wird durch sinnvoll definierte Schnittstellen erfüllt. Welche Informationen in einer auf physikalischen Eigenschaften basierenden Schnittstelle genau übertragen werden müssen, soll im Folgenden durch einige nahe liegende Überlegungen verdeutlicht werden.

Betrachtet man [Abbildung 6.1](#) ist schnell klar, dass in allen Verbindungen entweder Kräfte oder Geschwindigkeiten übertragen werden. Dies sind genau die Informationen, welche auch beim in der Mechanik üblichen „freimachen“ der Systeme verwendet werden. Es ist daher naheliegend auch diese Variablen für die Definition der Schnittstelle in Betracht zu ziehen. Damit kann jede in diesem Beispiel nötige Information zwischen den einzelnen Komponenten 1D translatorischer Systeme übertragen werden. Es wird somit möglich jede Komponente komplett getrennt von den anderen zu beschreiben und zu testen.

Allerdings ist die Geschwindigkeit in diesem Fall nicht die ideale Variable. Wesentlich besser geeignet ist die Position. Obwohl diese über Integration und Differentiation zusammenhängen, ist in der Position mehr Information enthalten. Würde die Geschwindigkeit verwendet werden, wäre es z.B. nicht möglich eine Initialposition über die Schnittstelle zu übertragen. Dies ist auch aus der Mathematik bekannt und tritt dort in Form der Integrationskonstante auf. Zusätzlich ist es mit der Position in der Schnittstelle möglich die sogenannten holonomischen Zwangsbedingungen zu modellieren. Wir können also z.B. einer Masse eine Abmessung mitgeben, damit nicht nur Punktmassen definiert werden können. Zusätzlich können Abmessungen von Elementen generell so definiert werden.

Wir haben nun als Schnittstellengrößen für die 1D translatorische Schnittstelle die Position und die Kraft festgelegt. Für ein besseres Verständnis wird zusätzlich die elektrische Domäne herangezogen. Dazu kann jede elektrische Schaltung wie etwa die aus einem der folgenden Unterabschnitte ([Abbildung 6.16](#) auf Seite 197) herangezogen werden. Betrachtet man die Gleichungen welche aus der Elektrotechnik bekannt sind, wird klar

dass das Verhalten jeder Komponente beschrieben werden kann, wenn die Spannung an der Komponente¹⁰ bzw. der Strom durch die Komponente¹¹ gegeben ist. Die Spannung an der Komponente ist dabei etwas ungenau ausgedrückt. Genauer handelt es sich dabei um das Potential vor und nach dem jeweiligen Objekt. Die Differenz dieser beiden Potentiale ergibt die Spannung an der Komponente. Überlegt man nun was sinnvolle Schnittstellen in der Elektrotechnik sind, bieten sich die elektrischen Kontakte (Pins) jedes Bauelementes an. Ein Pin kann eindeutig durch die Angabe des aktuellen elektrischen Potentials und des Stromflusses durch den Pin beschrieben werden. Dadurch ist auch klar wie ein Pin in Modelica aussehen muss. Mehr zum Pin in Modelica ist auf Seite 185 in Code 6.36 zu sehen.

Es sind nun für zwei technische Domänen sinnvolle Schnittstellenvariablen definiert. Um zu klären welche Arten von Variablen hier enthalten sind wollen wir nun zwei neue Begriffe einführen. Diese sind bereits in Kapitel 5 erläutert worden¹², sollen hier aber erneut kurz behandelt werden.

Es gibt in der Natur zwei grundsätzlich unterschiedliche Arten von Größen. Diese unterscheiden sich in der Reaktion auf eine Zusammenführung mehrerer gleicher Variablen. Sogenannte *extensive* Variablen ändern sich proportional mit der Menge der betrachteten Elemente. *Intensive* Variablen hingegen ändern sich nicht mit der Menge der betrachteten Elemente. Als Beispiel soll ein fliegendes Flugzeug herangezogen werden. Hier kann sich folgende Frage gestellt werden: Was ändert sich wenn wir zwei Flugzeuge betrachten? Wenn ein Flugzeug 1000m hoch fliegt, wie hoch fliegen zwei Flugzeuge dieses Typs? Oder wenn ein Flugzeug 1000km/h schnell fliegt, wie schnell fliegen zwei gleiche Flugzeuge? Offensichtlich hat bei der Position bzw. der Geschwindigkeit eines Objektes die Anzahl der Objekte keinen Einfluss auf diese Art der physikalischen Größe. Es handelt sich hier also um eine intensive Variable. Diese intensiven Variablen werden in Modelica als „Across“ Variablen bezeichnet¹³. Der Zusammenhang ändert sich wenn man folgende Frage stellt: Wenn ein Flugzeug 100 Personen transportieren kann, wie viele Personen können dann zwei Flugzeuge transportieren? Natürlich sind nicht direkt Personen die extensive Variable, sondern die Gewichtskraft, welche sie verursachen. Hier ändert sich die Variable proportional zur Menge. Dieser Typ von Variable wird in Modelica als „Flow“ Variable^{14,15} bezeichnet.

Nun versuchen wir, durch das Betrachten der Ähnlichkeiten zwischen dem mechanisch-translatorischen Beispiel und dem elektrischen Beispiel, eine allgemeine Beschreibung für die Schnittstellenvariablen zu finden. Beim Zusammenschließen zweier Komponenten mittels einer idealen Verbindung ergeben sich in der Verbindung immer Paare von

¹⁰Voltage *across* the Component.

¹¹Current *through* the component

¹²Im Zusammenhang mit Bondgraphen spricht man von Potential- und Flussvariablen.

¹³Intensiv = Potential bei Bondgraphen = Across Variablen in Modelica = unabhängig von der Menge

¹⁴Extensiv = Flow bei Bondgraphen = Flow in Modelica = proportional zur Menge,

¹⁵In der Bondgraphentheorie ist die Definition in der Mechanik eine andere. Dieser Umstand wird allerdings durch das Dualitätsprinzip ermöglicht (siehe Abschnitt 5.6 auf Seite 103).

Variablen. Eine der beiden Variablen nimmt denselben Wert an. Elektrisch gesehen wäre diese Variable das elektrische Potential dieses elektrischen Verbindungsstücks. Die zweite elektrische Variable ist der Strom. Dieser fließt auf der einen Seite in die Verbindung hinein und auf der anderen Seite der Verbindung wieder heraus. Über die Verbindung gesehen ergibt sich die Summe der Ströme also zu Null. Verbindet man beliebig viele Bauelemente mittels einer idealen Verbindung stellt sich wieder ein Potential für alle Verbindungen ein und die Ströme ergeben sich zu Null. Im mechanischen Beispiel ergibt sich das gleiche Bild. Eine ideal starre und masselose Verbindung zweier Elemente bewegt sich immer genau mit einer Geschwindigkeit bzw. liegen an der selben Position. Die an allen Anschlüssen dieser Verbindung wirkenden Kräfte summerieren sich zu Null.

Führt man dieses Gedankenexperiment weiter, ergibt sich, dass diese Eigenschaften von Verbindungen in allen technischen Domänen zu finden sind. Der jeweiligen Summe, welche sich immer zu Null ergibt, wurde allerdings jeweils unterschiedliche Namen versehen. In der Elektrotechnik wird sie als Kirchhoff'sche Knotenregel bezeichnet. In der translatorischen Mechanik nennt man sie das D'Alembert'sche Prinzip. Wir fixieren also, dass wir die Variablen, welche in einer Verbindung immer den gleichen Wert annehmen, allgemein als *intensive* Variablen bezeichnet werden. Die anderen sind *extensive* Variablen, welche sich in der Summe immer zu Null ergeben. In [Tabelle 6.2](#) wird eine Aufstellung der intensiven bzw. extensiven Variablen verschiedener technischer Domänen gegeben¹⁶.

Domäne	intensive Variable (Potentialvariable)	extensive Variable (Flussvariable)
<i>Elektrische Systeme</i>	elektrisches Potential	Strom
<i>Translatorische Systeme</i>	Position	Kraft
<i>Rotatorische Systeme</i>	Winkel	Drehmoment
<i>Hydraulische Systeme</i>	Druck	Volumenfluss
<i>Magnetik</i>	magnetisches Potential	magnetischer Fluss
<i>Thermodynamik</i>	Temperatur	Wärmestrom

Tab. 6.2.: Intensive und extensive Variablen in den verschiedenen Domänen

Es gibt einige Grundsätze zum Design von Schnittstellen, welche in [\[Sch09b\]](#) in Kapitel 21 aufgelistet sind. Einige Kernaussagen daraus sind folgende:

- Die Schnittstelle soll in der Regel nicht von dem beabsichtigten Ein-/Ausgangsverhalten der Komponente abgeleitet werden.
- Wird eine Komponente für sich betrachtet, sollen die Schnittstellen alle Variablen

¹⁶Für die Modellierung von Fluidodynamik, bei denen sowohl Masse als auch thermische Energie transportiert wird, gibt es in Modelica zusätzlich einen Variablentypen `stream`. Dieser soll hier allerdings nicht weiter besprochen werden.

enthalten, die zur Beschreibung des Verhaltens der Komponente nötig sind.

- Es sollen nur voneinander unabhängige Variablen in einer Schnittstelle übertragen werden. Sie sollen nicht über algebraische Zusammenhänge oder Integration/Differenziation voneinander abhängen.
- Werden die Schnittstellen miteinander verbunden, müssen die Erhaltungsgleichungen erfüllt sein. Dazu zählen Energie-, Massen-, Impuls-Bilanz, bzw. Kraft- und Momentgleichgewicht. Genauso müssen Randbedingungen wie die Gleichheit von Winkel, Position, elektrischem Potential, Temperatur, Druck etc. in verbundenen Schnittstellen erfüllt sein.

In Modelica sind viele der für die Modellierung nötigen Schnittstellen schon in der Standard Library definiert. Es müssen also selten grundsätzlich neuen Schnittstellen „erfunden“ werden. Zusätzlich macht eine Wiederverwendung Sinn, da die neu erstellten Komponenten so mit den bestehenden getesteten Komponenten verwendet werden können.

6.4.2. Akausalität

Nachdem nun klar ist, dass das Modell für jede Komponente separat definiert und getestet werden kann, wird noch eine zweite essentielle Eigenschaft der komponenten- bzw. objektorientierten Modellierung beleuchtet: die Akausalität. Dabei bezieht man sich in der Modellierung nicht auf das physikalische Ursache-Wirkungs-Prinzip, sondern auf eine rechnerische Kausalität. Es geht also darum was in einer Gleichung als bekannt und unbekannt angenommen wird. Diese Eigenschaft wird anhand eines elektrischen Widerstands demonstriert.

Will man in einer signalflossorientierten Beschreibungsform (z.B. Simulink) ein allgemeingültiges Modell eines elektrischen Widerstands definieren, müssen zwei Modelle erstellt werden. Beide beschreiben das Ohm'sche Gesetz. Dies ist nötig, da kausale Formen berücksichtigt werden müssen, wie in [Gleichung 6.17](#) dargestellt. Dabei ist es wichtig zu beachten, dass das Gleichheitszeichen „=“ eine Gleichung beschreibt, welche noch umgeformt werden kann. Ein „:=“ beschreibt eine Zuweisung wie sie aus prozeduralen Programmiersprachen wie z.B. C, C++ oder Java bekannt sind. Bei Zuweisung ist festgelegt, dass die Unbekannte auf der linken Seite des Zuweisungszeichens steht und aus den Bekannten auf der rechten Seite berechnet wird. Eine Umformung ist hier nicht mehr möglich.

$$u = R \cdot i \Rightarrow \begin{cases} u := R \cdot i & i \dots \text{Input} \\ i := u/R & u \dots \text{Input} \end{cases} \quad (6.17)$$

Die Notwendigkeit dieser unterschiedlichen Modelle ist auch schon in [Abschnitt 3.5](#), genauer [Abbildung 3.4](#) auf Seite 47 aufgetreten. Das Gleiche gilt natürlich auch für sämtliche andere technische Domänen, wie z.B. bei Dämpfern in mechanischen Systemen. Generell gilt, dass eine Gleichung nach Unbekannten wie z.B. den Schnittstellenvariablen umgeformt wird, da Parameter (hier der Widerstand R) als Eingabe in die Simulation eingebracht werden und somit bekannt sind. Auch wird klar, dass für die Lösbarkeit einer solchen Gleichung genau eine der Schnittstellenvariablen bekannt sein muss, um die andere daraus berechnen zu können.

Was für einen einzelnen Widerstand leicht zu bewerkstelligen ist, kann in aufwändigeren Modellen deutlich komplizierter werden. So ist es z.B. nötig das Modell des Widerstands zu ändern, wenn eine an den Widerstand angeschlossene Spannungsquelle gegen eine Stromquelle getauscht wird oder statt einer Kapazität eine Induktivität angeschlossen wird¹⁷. Diese Notwendigkeit lässt sich auf einer rein physikalischen Basis nicht erklären, weil sich an dem Widerstand physikalisch betrachtet nichts ändert. Was sich allerdings ändert ist die Form, oder genauer die Kausalität, in welcher das Ohm'sche Gesetz verwendet wird. Speziell bei größeren Systemen kann so eine kleine Änderung wie das Austauschen einer Spannungsquelle gegen eine Stromquelle oder im mechanischen Sinne das Vorgeben einer Drehzahl anstatt eines Drehmoments dazu führen, dass praktisch das ganze Modell angepasst werden muss. Wobei sich wiederum nicht die Gleichung, sondern nur deren Kausalität ändert.

Zusätzlich wird die Verständlichkeit des Modells verschlechtert. Es wäre deutlich einfacher das ohmsche Gesetz immer in derselben Form anzugeben. Um dies zu ermöglichen müssten nicht die in Gleichung (6.17) dargestellten Zuweisungen, sondern tatsächlich die mathematische Gleichung dem Modell hinterlegt sein. Diese muss dann vom Programm abhängig von der Topologie der zu simulierenden Schaltung in die jeweils entsprechende Zuweisung umgewandelt werden. Natürlich gilt das nicht nur für das ohmsche Gesetz sondern für alle Gleichungen, die auch deutlich komplexer werden können. Dies ist allerdings mit Beschreibungsformen welche eine Kausalität vorgeben nicht sinnvoll möglich.

Man erkennt, dass es in der Modellierung nicht sinnvoll ist, wie in der Programmierung Zuweisungen für die Beschreibung von Komponenten vorzugeben. Es ist hier sinnvoller und auch physikalisch besser rechtfertigbar, echte Gleichungen zur Beschreibung von Komponenten zu definieren. Daraus entstehen allerdings einige der zentralen Herausforderungen der objektorientierten Modellierung, da diese Gleichungen in Zuweisungen überführt werden müssen, um sie auf einem Computer ausführbar zu machen. Auf diese Eigenschaft wird in [Abschnitt 6.9](#) genauer eingegangen.

¹⁷Aufgrund der unterschiedlichen Zustandsvariablen

6.5. Dekomposition technischer Systeme in Klassen

In diesem Abschnitt sollen Überlegungen angestellt werden, wie das in diesem Kapitel schon mehrmals diskutierte Beispiel aus [Abbildung 6.1](#) in sinnvolle Klassen unterteilt werden kann. Ziel ist es Klassen zu definieren, welche möglichst universell eingesetzt werden können und somit möglichst viele Anwendungszwecke abdecken können.

6.5.1. Klassen, Objekte und Schnittstellen

Bisher wurde noch nicht geklärt was eine Klasse eigentlich ist. Leser, welche mit den Konzepten der objektorientierten Programmierung vertraut sind, können die nächsten beiden Absätze überspringen, da eine Klasse im Sinne der Programmierung und der Modellierung dieselben Aufgaben erfüllt. Eine Klasse ist eine Art Bauplan für ein Objekt. Die Klasse definiert über welche Eigenschaften das Objekt verfügt. Dabei können jegliche Arten von Variablen verwendet werden, um diese Eigenschaften zu beschreiben. Will man also etwa einen Kreis als Klasse beschreiben, wäre es sinnvoll in dieser Klasse die Variablen Mittelpunkt und Radius zu speichern. In welcher Form das geschieht ist dem Programmierer überlassen. Der Programmierer kann entscheiden welcher Variablenname verwendet wird, ob vordefinierte Datentypen, oder komplexe Strukturen etc. verwendet werden, um nur einige der Freiheiten zu nennen. Diese in der Klasse definierten Variablen werden in der objektorientierten Programmierung als Attribute bezeichnet. Weiters enthält eine Klasse die Definition von Methoden. Methoden sind aus der Programmierung bekannte Funktionen. Sie werden verwendet, um das Verhalten der Klasse zu bestimmen und z.B. Einfluss auf die Attribute zu nehmen.

Das oben beschriebene Konzept der Klasse entspricht wie bereits erwähnt, nur einem Bauplan. Wird nun eine Komponente aus einer solchen Klasse erstellt, ihr also ein Name gegeben und den Attributen Werte zugewiesen, wird daraus ein Objekt. Dieser Vorgang wird **instanzieren** genannt. Das erstellte Objekt nimmt Platz im Speicher ein und kann anhand des Namens eindeutig identifiziert werden. Verschiedene Objekte einer Klasse haben grundsätzlich die gleichen Attribute. Diese können aber in jedem Objekt unterschiedliche Werte annehmen. Man kann gleichzeitig mehrere Objekte einer Klasse erzeugen. Es ist also etwa möglich zwei gleiche und zusätzlich noch einen dritten unterschiedlichen Kreis zu definieren. Identifiziert werden Objekte über deren Namen, welche sich unterscheiden müssen, oder über die Adresse des Speicherbereichs in dem sie abgelegt sind. Alle drei haben einen eindeutigen Bezeichner, die Objekte sind aber vom Typ her dieselben. Wir haben also neue Klasse Kreis geschaffen, welche Daten (Attribute) und Funktionalität (Methoden) enthalten kann. Der Datentyp Integer kann genau so gut als eine Klasse angesehen werden, welche nur ein Attribut hat, welches den Wert des Integers enthält und über einen eindeutigen Namen oder die Adresse des Speicherbereichs identifiziert wird.

Schnittstellen beschreiben in der objektorientierten Modellierung meist physikalische

Verbindungen. Ihre Definition ist daher für die Aufteilung von Systemen in Klassen von zentraler Bedeutung. Der erste und äußerst wichtige Schritt wird es also sein sinnvolle Schnittstellen zu definieren. Dazu wurde bereits in [Unterabschnitt 6.4.1](#) diskutiert welches in diesem Fall sinnvolle Variablen sind. Das Ergebnis war, dass es für eine 1D translatorische Schnittstelle sinnvoll ist, die Kraft und die Position als Schnittstellenvariablen zu wählen. Mit diesen beiden Schnittstellen-Variablen kann also jede Komponente unabhängig von ihrer Umgebung beschrieben werden.

Anders als in der objektorientierten Programmierung ist es in der Modellierung oft einfach, sinnvolle Objekte zu erkennen. Sie korrespondieren meist mit den einzelnen Bauteilen oder Komponenten aus welchen sich ein System zusammensetzt. Im Beispiel aus [Abbildung 6.1](#) kann schnell erkannt werden, aus welchen Komponenten sich das Gesamtsystem zusammensetzt. Offensichtliche Komponenten sind die Masse, die Feder und der Dämpfer. Zwar nicht ganz so offensichtlich aber trotzdem naheliegend ist die Wand, welche sozusagen als Nullpunkt (die absolute Positionierung) des Systems dient. Auch die externe Anregung des Systems sollte über eine Klasse beschrieben werden und so müssen wir auch die auf der linken Seite eingetragene Kraft als eigenständiges Objekt modellieren. Um die Komponenten nun unabhängig von ihrer Umgebung definieren zu können, müssen wir sinnvolle Stellen im Modell als Schnittstellen definieren. Es bieten sich Stellen an, an welchen die Komponenten verbunden werden können. Dies ist in [Abbildung 6.8](#) über die im Vergleich zu [Abbildung 6.1](#) hinzugefügten Quadrate dargestellt.

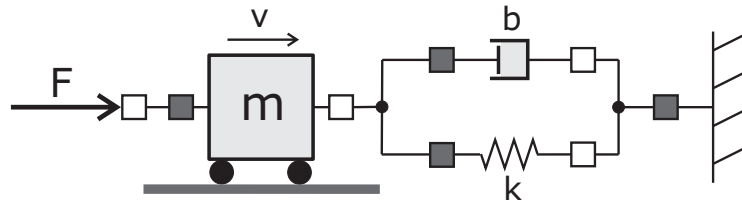


Abb. 6.8.: Sinnvolle Schnittstellen im Feder-Masse-Schwinger

Generell sind Schnittstellen vorgesehen, um Objekte mit ihrer Umgebung in Verbindung treten zu lassen. Daher soll ein Objekt - egal welcher technischen Domäne - wie in [Abbildung 6.9](#) aufgebaut sein. Dies ermöglicht in Zusammenhang mit der Simulationsumgebung eine grafische Verknüpfung von Objekten auf einem sehr intuitiven Weg. Davon abweichende Wege sind zwar möglich, sollten aber nur in Ausnahmesituationen verwendet werden.

Über die Definition der intensiven und extensiven Variablen in der Schnittstelle ist es dem Simulationsprogramm so möglich, die für das Verhalten der verknüpften Objekte nötigen Gleichungen automatisch zu erstellen. Dabei werden, wie bereits in [Unterabschnitt 6.4.1](#) erwähnt, in den verbundenen Schnittstellen die intensiven Variablen gleich gesetzt. Extensive Variablen müssen sich in jedem Fall zu null addieren. Es werden al-

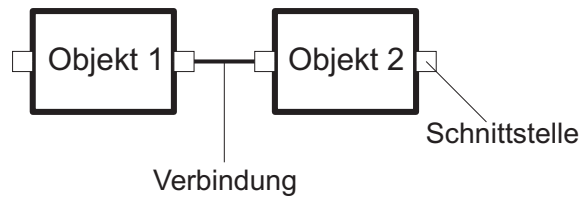


Abb. 6.9.: Schnittstellen als Verbindung zu Umgebung

so beim Verbinden zweier Schnittstellen automatisch Gleichungen generiert, welche in [Abbildung 6.10](#) dargestellt sind.

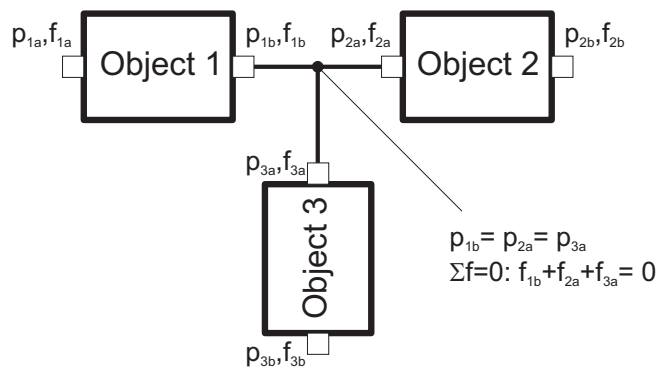


Abb. 6.10.: Schnittstellen und die daraus generierten Gleichungen

In der objektorientierten Modellierung wird die Unterscheidung zwischen Klasse und Objekt nicht sehr strikt gehandhabt. Sie ist zwar theoretisch vorhanden, praktisch spricht man aber immer von Modellen.

6.5.2. Schnittstellen in Modelica

Nun wollen wir also eine erste mechanisch-translatorische Schnittstelle in Modelica implementieren. Wichtig dabei ist, dass die Schnittstelle in Modelica nicht von der Klasse `model` abgeleitet wird, sondern von der Klasse `connector`. Dies ist nötig, da in der Model Klasse immer gleich viel Gleichungen wie Unbekannte definiert sein müssen. In einem Connector jedoch ist es nicht erlaubt Gleichungen zu definieren. Diese werden über die Unterscheidung zwischen extensiver und intensiver Variable automatisch erstellt, sobald die Objekte miteinander verbunden werden. Dies kann entweder grafisch über Verbindungslinien oder textuell über ein `connect` Statement erfolgen.

```
1 connector Flange
2   "1D translational flange"
3
4   SI.Position s "Absolute position of flange";
5   flow SI.Force f "Cut force directed into flange";
6
7 end Flange;
```

Code 6.11: Mechanisch-translatorischer 1D Flansch

Über das Verbinden der einzelnen Connectoren werden automatisch die angenommenen Bedingungen bezüglich des Verhaltens der Variablen bestimmt. Die Modellierungsumgebung weiß also, dass Potentialvariablen in einer Verbindung denselben Wert annehmen und sich Flussvariablen zu Null addieren. Daher muss auch die Schnittstelle so gestaltet sein, dass sie diese Gesetze erfüllt.

Nun sind die Voraussetzungen geschaffen, um mit der Modellierung der eigentlichen Elemente zu beginnen. Dabei ist zu erwähnen, dass es sich dabei um eine vereinfachte Version der Modelle aus der Modelica Standard Library handelt, welche aber in dieser Form nicht simulierbar sind. Sie werden hier in dieser Form verwendet, um das Verständnis zu erleichtern. Für eine detailliertere Version kann die Standard Library eingesehen werden.

6.5.3. Die Komponenten in Modelica

Im Folgenden sollen die einzelnen nötigen Komponenten des Beispiels aus [Abbildung 6.8](#) in Modelica umgesetzt werden.

Die Masse

Wir beginnen mit der Masse, wobei wir bereits eine etwas fortgeschrittene Masse modellieren, welche auch eine Länge L haben kann. Als Komponentengleichung verwenden wir die wohl-bekannte Trägheitsgleichung

$$F_m = m \cdot a = m \cdot \ddot{x} \quad (6.18)$$

Zusätzlich müssen wir den Zusammenhang zwischen den Kräften in den Schnittstellen und der Trägheitskraft F_m herstellen. Dazu verwenden wir den Zusammenhang nach D'Alembert. Dieser besagt, dass die Summe aller Kräfte null ergeben muss.

$$F_m = F_l + F_r \quad (6.19)$$

F_l beschreibt dabei die Kraft im linken Flansch. F_r dementsprechend die Kraft im rechten Flansch. Beide Kräfte zeigen zur Masse hin, was die Vorzeichen in der Gleichung

erklärt. Dies ist eine Vorzeichenwahl, welche es möglich macht, die Summe der Kräfte in den Schnittstellen auf Null zu setzen.

Wie bereits festgestellt, sind für die Klasse Masse zwei Schnittstellen nötig. Diese sind in Code 6.12 in den Zeilen 6 und 7 definiert. Die beiden Parameter sind in den Zeilen 3 und 4 definiert. Die Unbekannten sind die Position s , die Geschwindigkeit v und die Beschleunigung a . In den Gleichungen erkennt man, wie die Position von Masse und Flanschen sowie die Länge der Masse zusammenhängen. Zusätzlich müssen die über die Schnittstelle zur Verfügung gestellten Größen für die Modellierung verwendet werden, um die für die Beschreibung der Dynamik der Masse nötige Beschleunigung zu errechnen. Daher wird die Schnittstellenvariable Position zweimal abgeleitet, um die Beschleunigung zu berechnen und den Zusammenhang mit der zweiten Schnittstellenvariable, der Kraft, herstellen zu können. Die Ableitungen werden separat durchgeführt, da Modelica für die Ableitung nur den `der()` Operator zur Verfügung stellt.

```

1  model Mass "Sliding mass with inertia"
2    import SI = Modelica.SIunits;
3    parameter SI.Mass m "Mass of the sliding mass";
4    parameter SI.Length L "Length of component, from left flange to
      right flange";
5
6    Flange flange_l "Left flange"
7    Flange flange_r "Right flange"
8
9    SI.Position s "Absolute position of center of component";
10   SI.Velocity v "Absolute velocity of component";
11   SI.Acceleration a "Absolute acceleration of component";
12
13  equation
14    flange_l.s = s - L/2;
15    flange_r.s = s + L/2;
16    v = der(s);
17    a = der(v);
18    m*a = flange_l.f + flange_r.f;
19  end Mass;

```

Code 6.12: Modell einer Masse

Nun haben wir ein Masse definiert, welche in dieser Form über die in dem Objekt definierten Schnittstellen allgemein einsetzbar ist.

Eine Feder

Auch für die Beschreibung der Feder gehen wir direkt von der Beschreibungsgleichung der Komponente aus. Zusätzlich müssen wir die Länge der Feder Δx (`s_rel`) aus den

Positionen der Flansche x_l (`flange_l.s`) und x_r (`flange_r.s`) berechnen.

$$F = k \cdot \Delta x \quad (6.20)$$

$$\Delta x = x_r - x_l \quad (6.21)$$

Dies ergibt in Modelica den Code 6.13.

```
1  model Spring
2    import SI = Modelica.SIunits;
3
4    parameter SI.TranslationalSpringConstant k = 1 "spring
      constant";
5
6    SI.Force f "force between flanges";
7    SI.Distance s_rel "relative distance";
8
9    Flange flange_l;
10   Flange flange_r;
11  equation
12   s_rel = flange_r.s - flange_l.s;
13   flange_r.f = f;
14   flange_l.f = -f;
15   f = k*s_rel;
16  end Spring;
```

Code 6.13: Modell einer Feder

Ein Dämpfer

Im Unterschied zu der Federgleichung ist für den Dämpfer die Geschwindigkeit die zentrale Größe, da die Kraft welche vom Dämpfer erzeugt wird, proportional zu dieser ist. Diese muss über einmaliges Differenzieren aus der Position errechnet werden.

$$F = b \cdot \Delta v = b \cdot \Delta \dot{x} \quad (6.22)$$

$$\Delta v = v_r - v_l \quad (6.23)$$

Dies ergibt in Modelica den Code 6.14.

```
1  model Damper
2    import SI = Modelica.SIunits;
3
4    parameter SI.TranslationalDampingConstant b = 1 "damping
      constant";
5
6    SI.Force f "force between flanges";
```



```

7   SI.Distance s_rel "relative distance";
8   SI.Velocity v_rel "relative velocity";
9
10  Flange flange_l;
11  Flange flange_r;
12  equation
13    s_rel = flange_r.s - flange_l.s;
14    v_rel = der(s_rel);
15    flange_r.f = f;
16    flange_l.f = -f;
17    f = b*v_rel;
18  end Damper;

```

Code 6.14: Modell eines Dämpfers

Die Wand

Eine Wand gibt im einfachsten Fall eine Position fix vor. Dabei interessieren die entstehenden Kräfte nicht weiter, sie werden von der Wand einfach absorbiert. Daraus entsteht [Code 6.15](#).

```

1  model Wall
2    Flange flange;
3
4    equation
5      flange.s = 0;
6    end Wall;

```

Code 6.15: Modell einer Wand

Zusätzlich könnte man natürlich die Position der Wand als Parameter vorgeben und sie somit verschiebbar machen. Für unseren Fall ist das allerdings nicht nötig.

Die Kraft

Zuletzt brauchen wir noch ein Modell mit dem wir das System anregen können. Dabei wird von einer Anregung über eine konstante Kraft wie in [Code 6.16](#) ausgegangen.

```

1  model Force
2    import SI = Modelica.SIunits;
3
4    parameter SI.Force f = 1;
5
6    Flange flange;
7

```

```
8 equation  
9   flange.f = -f;  
10 end Force;
```

Code 6.16: Modell einer Kraftquelle

Das negative Vorzeichen der Kraft ergibt sich wiederum aus der immer nach innen zeigenden Kraft in den Schnittstellen.

Diese konstante Kraft könnte über verschiedene Wege auch zeitveränderlich vorgegeben werden. Man kann etwa für jede Art der Anregung (konstant, sinusförmig, sprungförmig etc.) eine eigene Quelle definieren und die genaue Signalform über Parameter vorgeben. Eine andere Möglichkeit bieten kausale Signaleingänge. Diese können über die Statements `input` bzw. `output` definiert werden, oder es werden aus der Modelica Standard Library die vorgefertigten `RealInput` bzw. `RealOutput` verwendet. So können über die ebenfalls in der Modelica Standard Library definierten `Blocks`¹⁸ zur Vorgabe von Werten verwendet werden.

6.5.4. Packages in Modelica

Modelica bietet die Möglichkeit verwandte Klassen in Packages zusammenzufassen. Diese Packages werden genau wie einzelne Modelle in einer `.mo` Datei gespeichert. Es ergibt sich der Vorteil, dass nach dem Öffnen des Packages alle darin zusammengefassten Modelle geladen werden und in allen anderen Modellen dieses Packages automatisch zur Verfügung stehen.

Um ein Package zu erstellen wird das Keyword `model` aus den bisherigen Beispielen durch `package` ersetzt. Nun können entweder weitere Unterpackages erstellt werden, oder bereits Modelle eingefügt werden. Üblicherweise stellen grafische Modellierungsumgebungen relativ komfortable Wege für die Erstellung dieser Package-Strukturen zur Verfügung, weshalb hier nicht weiter auf die Erstellung dieser eingegangen wird. Eine gute Referenz für eine sinnvolle Package-Struktur ist die Modelica Standard Library.

Hier erstellen wir das Package, um die einzelnen Modelle gut verschalten zu können und damit sie in einem File zusammengefasst werden können. Das Konzept der Packages sollte aber schon bei eher kleinen Aufgaben angewandt werden. Dies sieht in unserem Falle aus wie in [Code 6.17](#).

Werden verschiedene Modelle und Beispiele zu einem gewissen Aufgabengebiet in einem Package zusammengefasst, werden diese üblicherweise als Library bezeichnet. Neben den Klartextversionen solcher Libraries ist es auch möglich verschlüsselte Versionen zu erstellen, um Details der Implementierung vor dem User zu verstecken. Der User kann die Modelle aber trotzdem verwenden. Dies wird üblicherweise in kommerziell verkauften Libraries angewandt.

¹⁸Zu finden unter `Modelica.Blocks`

```

1  package Mechanics_1D_translational
2  model Mass
3  ...
4  end Mass;
5
6  model Spring
7  ...
8  end Spring;
9
10 model Damper
11 ...
12 end Damper;
13
14 // Force and Wall models
15 end Mechanics_1D_translational;

```

Code 6.17: Zusammenfassung der Modelle in einem Package

6.5.5. Verschalten der Komponenten

Die Verschaltung der Komponenten erfolgt über `connect` Statements. Diese erstellen im Hintergrund die für die Schnittstellen nötigen Gleichungen, setzen also in unserem Fall die Positionen der beiden Schnittstellen auf dieselben Werte und sorgen dafür, dass sich die Kräfte in den Schnittstellen zu Null aufheben. Der Code für die Verschaltung der in [Unterabschnitt 6.5.3](#) erstellten Komponenten zu dem ursprünglich zu modellierenden Beispiel ([Abbildung 6.8](#) auf Seite 158) ist in [Code 6.18](#) zu sehen.

Im ersten Teil werden dabei die einzelnen Objekte aus den Klassen im Package instanziiert. Es wird also z.B. ein Objekt mit dem Namen `f` vom Typ `Force` erstellt. Auf die Objekte in diesem Modell kann jetzt über `f.Variablenname` zugegriffen werden. Dies wird in den Connect Statements ausgenützt. Weiters können bei der Instanzierung Parameterwerte übergeben werden. Dies wird als eine *Modifier* Gleichung in den Klammern hinter dem Objektnamen getan. Dabei wird dem Parameter mit dem entsprechenden Namen links vom Gleichheitszeichen der Wert rechts vom Gleichheitszeichen zugewiesen. Die Kraft wird also in diesem Beispiel nicht ein Newton sondern, wie im Kraft-Modell definiert, zehn Newton sein. Genau so ist das mit allen Parametern möglich. So werden in dem Beispiel auch Federkonstante, Dämpfungswert und Masse angepasst. Hat ein Modell mehrere Parameter die angepasst werden müssen, können mehrere dieser Modifier Gleichungen, mit Beistrichen getrennt, verwendet werden. Werden keine Gleichungen übergeben verwenden die Modelle die per Default definierten Parameter. Gibt es keine Defaultparameter¹⁹ wird eine Fehlermeldung ausgegeben.

In den Connect-Statements wird angegeben welche der Schnittstellen verbunden sein sollen. Alle diese Vorgänge sind zwar wie hier demonstriert textuell durchführbar,

¹⁹Was seit Modelica 3.2 üblich ist.

üblicherweise werden die Verbindungen aber grafisch definiert, da dies schneller und übersichtlicher ist. Zusätzlich werden bei der grafischen Variante die zur Darstellung benötigten Daten in Form von Annotations (siehe [Unterabschnitt 6.7.14](#)) automatisch erstellt.

```
1  model FederMasseSystem
2      Force f(f=10);
3      Spring spring(k=100);
4      Damper damper(b=2);
5      Mass mass(m=1);
6      Wall wall;
7  equation
8      connect (f.flange_r, mass.flange_l);
9      connect (mass.flange_r, damper.flange_l);
10     connect (damper.flange_r, wall.flange_l);
11     connect (mass.flange_r, spring.flange_l);
12     connect (spring.flange_r, wall.flange_l);
13     connect (c.n, ground.p);
14 end Mechanics_1D_translational;
```

Code 6.18: Verschalten der einzelnen Modelle zum Beispiel

6.6. Objektorientiertheit in Programmierung und Modellierung

In diesem Abschnitt wird die Verwandtschaft der objektorientierten Modellierung mit der objektorientierten Programmierung etwas näher beleuchtet. In objektorientierten Programmiersprachen wird mittels verschiedener Techniken versucht die Erstellung komplexer Programme zu vereinfachen, sowie die Wartbarkeit, Erweiterbarkeit und Verständlichkeit des Codes zu verbessern. Einige dieser Konzepte wurden in der objektorientierten Modellierung übernommen. Auf diese wird im Folgenden eingegangen.

Die Konzepte welche von der objektorientierten Modellierung in Anlehnung an die objektorientierte Programmierung vorhanden sind, werden im Folgenden im Kontext der Programmierung beschrieben.

- **Abstraktion:** Sie versucht die Art der Implementierung von der Verwendung zu trennen. Es ist z.B. der Aufruf von Methoden eines Objektes gemeint, bei welchem der Aufrufende nicht wissen muss, wie die Methode implementiert ist. Als allgemein verständliches Beispiel sei hier ein Autofahrer angegeben. Das Objekt Fahrer bekommt den Befehl zu beschleunigen, ob es allerdings ein Modell des Fahrers enthält, welcher von Hand schaltet, oder mittels eines Automatikgetriebes, muss dem Befehlsgeber nicht bewusst sein.
- **Datenkapselung:** Dabei geht es darum die Daten oder Zustände, welche in

einer Klasse abgelegt sind, von der direkten Manipulation durch den Benutzer zu verbergen. Der Zugriff wird nur über wohl definierte Funktionen ermöglicht.

- **Komposition:** Mehrere (Sub-)Objekte werden zu einem neuen, übergeordneten Objekt zusammengefasst. Wird eines der Subobjekte angepasst, hat das direkt Auswirkungen auf das übergeordnete Objekt.
- **Vererbung:** Eine bestehende Klasse kann über die Vererbung erweitert werden. Die neue Klasse besitzt alle Attribute und Methoden der Basisklasse. Die neue Klasse „erbt“ also die Eigenschaften der Basisklasse. Diese neue Klasse kann durch neue Funktionen oder Attribute erweitert werden. Veränderungen an der Basisklasse wirken sich auf alle Klassen aus, welche von ihr erben (abgeleitet sind).
- **Polymorphie:** Ein Objekt welches nach außen hin dasselbe Aussehen hat, sich aber anders verhalten kann. Beispiele hierzu sind Operator Overloading²⁰ oder direkt in Modelica die `replacable` und `redeclare` Statements welche in [Unterabschnitt 6.7.12](#) diskutiert werden.

Die Kommunikation zwischen den einzelnen Objekten findet in der Programmierung über Nachrichten (engl. Messages) statt. In der Modellierung werden dazu - wie schon öfters besprochen - Schnittstellen verwendet. Diese Konzepte unterscheiden sich grundsätzlich, da Objekte in der Modellierung sehr eng miteinander verknüpft sind, was in der Programmierung nicht zwingend der Fall sein muss.

6.6.1. Konzepte aus der objektorientierten Programmierung

Hier soll aufgezeigt werden, wie die im vorherigen Abschnitt vorgestellten Techniken der Objektorientierung auf die Modellierung angewandt werden können. Grundsätzlich wäre es möglich die verschiedenen physikalischen Teilsysteme, aus welchen sich das zu modellierende System zusammensetzt, direkt mit den Techniken der objektorientierten Programmierung abzubilden. Also einen objektorientierten Simulationscode zu erzeugen. Dabei ergeben sich allerdings Probleme, welche aus den verschiedenen Einsatzgebieten der beiden objektorientierten Ansätze entstehen. Objekte, welche kontinuierlich veränderliche Größen berechnen und diese zu einem anderen Objekt kommunizieren, müssen dies gezwungenermaßen mit einer sehr hohen Frequenz tun. Dazu ist die in der objektorientierten Programmierung verwendete Technik des „Message Passing“ zu langsam. Es muss also aus der objektorientierten Definition der Objekte ein monolithischer²¹ Code-Block erstellt werden, um diesen effizient abarbeiten zu können [Cel96]. Zusätzlich ergibt sich die Notwendigkeit für einen monolithischen Code-Block in der Si-

²⁰Der Operator „+“ zur Addition kann für Real Werte oder komplexe Zahlen verwendet werden. In den beiden Fällen werden unterschiedliche Operationen ausgeführt, obwohl der Operator der gleiche ist.

²¹Ein eng gekoppelter, nicht in Teilsysteme oder Komponenten gegliederter Code.

mulation von gemischt kontinuierlichen und diskreten (hybriden) Systemen. Die dazu nötigen Techniken lassen sich nur in einem monolithischen Code anwenden [ECO93], [Cel79].

Aus genau dieser Überführung des objektorientierten Modells in einen monolithischen Code entstehen allerdings einige der fordernten Problemstellungen der objektorientierten Modellierung. Diese werden teilweise in [Abschnitt 6.9](#) und [Kapitel 7](#) behandelt.

Ein weiterer wichtiger Unterschied zwischen der objektorientierten Modellierung und Programmierung ist, dass es bisher keine etablierte Sprache erlaubt dynamisch, also während der Laufzeit eines Programms, Objekte zu erstellen. Arbeiten zu diesen „strukturvariablen“ Systemen, bei welchen sich die Anzahl der Zustandsvariablen während der Simulation ändern kann sind allerdings im Gange. Zu diesen Thema gibt es einen kurzen Ausblick in [Abschnitt 6.8](#).

6.6.2. Verwandtschaft der objektorientierten Modellierung zur objektorientierten Programmierung

Aus dem vorherigen Abschnitt ist klar geworden, dass die Objektorientierung in der Modellierung eher als eine Art Strukturierungskonzept Verwendung findet und sich im erstellen Simulationscode nicht mehr direkt erkennen lässt. Die Objektorientierung beschränkt sich also - wie der Name schon sagt - auf die Art der Modellierung. Es sind dabei folgende Grundprinzipien, welche die Rahmenbedingungen für diese Art der Modellierung festlegen. Einige der dafür nötigen Voraussetzungen wurden in [Abschnitt 6.4](#) anhand eines Beispiels verdeutlicht.

Es wird grundsätzlich versucht ein technisches System in Objekte zu unterteilen, welche jeweils einen möglichst abgeschlossenen, sinnvollen und wiederverwendbaren Teil des Gesamtsystems bilden. Das ist in vielen Fällen einfach. So sind etwa bei der Modellierung elektronischer Schaltkreise die Objekte durch die einzelnen elektronischen Komponenten wie Widerständen, Kapazitäten, Quellen etc. vorgegeben. In anderen Fällen ist die Aufteilung in Objekte deutlich komplizierter z.B. für Reifen [[And09](#)].

Ein grundlegender Unterschied zwischen objektorientierter Modellierung und objektorientierter Programmierung ist im Begriff der Schnittstelle (engl. Interface) zu finden. Diese beschreiben in der objektorientierten Programmierung eine Mindestmenge nötiger Methoden, welche von jeder Klasse, welche diese Interface verwendet zu implementieren sind. In der objektorientierten Modellierung hingegen sind Schnittstellen in einem physikalischen Sinne zu verstehen. Dabei handelt es sich um die z.B. mechanisch-translatorische Schnittstelle zwischen Objekten, welche physikalische Größen zwischen den Objekten austauschen.

Wiederverwendbarkeit

Dieser Punkt ist vor allem bei der Modellierung großer Systeme von zentraler Bedeutung. Oft kommt es in großen Systemen vor, dass ein einzelnes Modell (ein Widerstand, ein Motor etc.) in einem übergeordneten Modell mehrmals auftaucht. Auch wenn diese nicht exakt das gleiche Verhalten aufweisen, aber doch durch dieselben physikalischen Zusammenhänge beschrieben werden können. Es muss daher die Möglichkeit geben, die Parameter eines Modells (Widerstandswert eines elektrischen Widerstands) unabhängig von dessen Modell zu verändern. Dies entspricht in der objektorientierten Programmierung der **Instanziierung** mit verschiedenen Werten für die Attribute der Klasse. Wird ein komplexes Objekt aus mehreren einfacheren Objekten geformt, wird dies als **Komposition** bezeichnet. Ein einfaches Beispiel dafür ist das Zusammensetzen eines mechanischen Modells aus mehreren grundlegenden Elementen wie z.B. Massen und Federn. Auch dieses Modell entspricht wieder einem Objekt und kann über das anbringen geeigneter Schnittstellen in einem größeren Modell wiederverwendet werden usw.

Betrachtet man ein einfaches Beispiel aus der Elektrotechnik, so haben Widerstand, Kapazität und Induktivität die gemeinsame Eigenschaft, dass sie zwei Pins haben, und durch eine Potentialdifferenz (Spannungsabfall) sowie einen durch sie fließenden Strom charakterisiert werden. Um sich diese Ähnlichkeiten zu Nutze zu machen, wird das Konzept der **Vererbung**²² von der objektorientierten Programmierung aufgegriffen. Es wird in der objektorientierten Modellierung einmalig ein Modell erstellt, welches die gemeinsamen Eigenschaften mehrerer Objekte beschreibt. Es wird also einmalig ein Grundmodell erstellt, welches die Eigenschaften z.B. aller elektrischen Elemente mit zwei Pins modelliert. In diesem konkreten Fall wird beschrieben, dass die Summe der Ströme in den beiden Pins null sein muss und die Potentialdifferenz der am Element anliegenden Spannung entspricht. Dieses Modell muss an sich nicht funktionstüchtig sein. Später kann diese Beschreibung über eine Komponentenengleichungen (ohmsches Gesetz, Induktionsgesetz etc.) mittels des Konzepts der Vererbung erweitert werden. So wird z.B. das Verhalten von Widerstand, Kapazität etc. über wenige zusätzliche Gleichungen hinzugefügt, um eine vollständige Beschreibung dieser Elemente zu erhalten. Die Eigenschaften der Pins werden dann direkt vom Grundmodell abgeleitet, und die erweiternden Gleichungen hinzugefügt. Da das Grundmodells nur einmal vorhanden ist und die darauf aufbauenden Modelle direkt darauf zugreifen, muss dieses bei einer Anpassung nur einmal bearbeitet werden. Somit wird die Wartung komplexer Modelle deutlich vereinfacht. Dabei muss beachtet werden, dass das Grundmodell für sich genommen keine physikalische Funktion hat und in der Realität nicht existiert. Folglich sind diese Grundmodelle auch nicht simulierbar. Ein Beispiel dazu ist in [Code 6.39](#) aufgezeigt.

Ebenfalls für die Wiederverwendbarkeit wird das Konzept der **Polymorphie** zumin-

²²Entspricht im Normalfall einer Spezialisierung.

dest teilweise umgesetzt. Dabei können einzelne Teile eines Modells durch eine modifizierte Beschreibung ersetzt werden. Es können so z.B. einfache lineare Modelle direkt durch komplexere Modelle ersetzt werden. Dabei ist der große Vorteil, dass etwa ein Modell, bei welchem nur eine Gleichung verändert werden muss, eben nur diese eine Gleichung abgeändert werden kann, wobei der Rest des Modells aus der Basisklasse übernommen wird. Ganz grundlegend bedeutet Polymorphismus, dass Objekte mit identischen Methoden unterschiedliche Funktionalität haben [Zim10]. Ein Beispiel dazu ist in [Unterabschnitt 6.7.12](#) aufgezeigt.

Durch beide der hier aufgezeigten Konzepte (Vererbung und Polymorphismus) wird erreicht, dass die Anzahl der mehrfach erstellten Codezeilen minimiert werden kann. Dies erleichtert die Wartung und Erweiterung vor allem bei entsprechend großen Modellen deutlich. Allerdings wird dadurch - vor allem bei übertriebener Anwendung - auch die Lesbarkeit der Modelle etwas eingeschränkt, da die Gleichungen, welche ein Objekt beschreiben, über mehrere Klassen verteilt werden können und somit der Zusammenhang schwerer zu erfassen ist. Trotzdem ist die Vermeidung von mehrfachen Codezeilen ein zentraler Punkt für die Erstellung großer Modelle. Man sollte allerdings von Beginn der Modellierung an einige Zeit aufwenden, um sinnvolle Ansatzpunkte für die vorgestellten Konzepte zu suchen.

Nähe zur Realität

Die Nähe zur Realität zielt vor allem auf die Verständlichkeit für den Modellierer ab. Dabei werden unter Anderem generell gültige physikalische Gesetze wie z.B. die Energieerhaltung oder die Verbindungen zwischen den einzelnen Komponenten berücksichtigt. Dadurch können Fehlerquellen reduziert werden.

Innerhalb von erstellten Modellen kann entschieden werden, welche Größen aus dem Modell von der Außenwelt versteckt sind, welche einsehbar sind, und welche direkt als Schnittstellen zur Verfügung gestellt werden. Es handelt sich also dabei um die Umsetzung der **Datenkapselung** bzw. der **Abstraktion** in der Modellierung. Ein weiterer essentieller Punkt ist, dass **Schnittstellen**, welche zur Verbindung der einzelnen Submodelle verwendet werden, möglichst den in der Realität verwendeten entsprechen. So werden elektrische Leitungen, mechanische Wellen oder ähnliches verwendet, welche jeweils eine extensive und eine intensive Größe übertragen und somit dem Transport von Leistungen (oder Energieflüsse) entsprechen. Dieser Ansatz ist ähnlich zu dem in [Kapitel 5](#) vorgestellten Bond. Es ist also gewährleistet, dass diese topologischen Verbindungen ähnlich wie in Experimenten im Labor erstellt werden können. Man kann somit komplette mechanische Wellen oder elektrische Pins verbinden und muss die zusammengehörigen Potential und Flussvariablen nicht künstlich voneinander trennen. Dies ist ein weiterer Grund für die Notwendigkeit von Akausalität in der Definition der Modelle, weil somit im Vorhinein nicht vorgegeben werden kann, ob die Potential oder die Flussvariable die jeweils zu berechnende oder bekannte Größe ist.

Als weiterer Punkt soll die Möglichkeit zur **hierarchischen Strukturierung** (= Komposition) von Modellen aufgezeigt werden. Dabei ist es möglich eine Kombination verschiedener Modelle zu einem neuen Modell zusammenzufassen. Diese kann dann wieder ein Teilmodell eines neuen Modells sein usw. Dies ermöglicht einen strukturierten Aufbau komplexer Modelle.

Um in der Wahl der Modellierungsform möglichst frei zu sein, kann die eigentliche Implementierung vom Modellierer „versteckt“ werden. Diese Vorgangsweise wird als „wrappen“ bezeichnet. Dabei wird dem eigentlichen Modell, das z.B. aus Bondgraphen bestehen kann, oder direkt durch mathematische Zusammenhänge definiert ist, eine grafische Darstellung „überlagert“. Dadurch ist es möglich die standardisierten Symbole aus den verschiedenen technischen Domänen nachzubilden und so eine intuitive Nutzung für Spezialisten der verschiedenen Domänen zu ermöglichen. Es ist so möglich, jemanden mit einer Bibliothek arbeiten zu lassen, welche mit Hilfe von Bondgraphen aufgebaut ist, ohne dass der Anwender diese Technik beherrschen muss. Der Anwender muss, um ein funktionierendes Modell aus einzelnen Komponenten aufbauen zu können, nicht jede Teilkomponente verstehen. Er muss sie allerdings auf eine einfache Art korrekt miteinander verknüpfen können. Diese Forderung machen Abstraktion, Datenkapselung und Schnittstellen zu äußerst zentralen Aspekten der objektorientierten Modellierung.

Zusammenfassung

Man sieht also, dass es zwischen der objektorientierten Programmierung und der objektorientierten Modellierungstechnik einige Verwandtschaften gibt. Dabei ist allerdings auch zu beachten, dass es durch die verschiedenen Einsatzgebiete auch zu deutlichen Unterschieden in der Auffassung und Umsetzung mancher Details kommen kann.

6.6.3. Objektorientierte Modellierung und Bondgraphen

Die Methodik der Bondgraphen wurden 1959 von Paynter entwickelt und von Karnopp und Rosenberg weiterentwickelt. Das geschah während einer Zeit zu der die objektorientierte Modellierung noch nicht existierte. Trotzdem lassen sich zwischen den beiden Modellierungstechniken so viele Gemeinsamkeiten feststellen, dass Borutzky [Bor10] und Broenik [Bro99] die Ansicht vertreten, dass es sich bei der Bondgraphen Methodik um einen Spezialfall der objektorientierten Modellierung handelt.

6.7. Die Fähigkeiten von Modelica

Hier sollen fortgeschrittene Konzepte in der Modellierungssprache Modelica aufgezeigt werden, welche sich teilweise die objektorientierten Konzepte zu Nutze machen. Dabei

werden für den Großteil der Konzepte nur der Anwendungszweck erläutert und ein einfaches Beispiel gegeben. Der Leser wird dann auf einschlägige Literatur verwiesen, welche diese Konzepte im Detail erläutert. Dabei sei vor allem auf [Fri04] verwiesen, welcher eine äußerst umfangreiche Zusammenstellung zu einer etwas älteren Version von Modelica (2.2) präsentiert. In Kapitel 21 von [Sch09b] sind einige Konzepte ausgearbeitet und sehr gut aufbereitet. Bei sehr detaillierten Fragen hilft die Modelica Sprachdefinition [Ass10] weiter. Zusätzlich werden hier die Grundzüge von Modelica wiederholt, um einen direkten Einstieg in dieses Kapitel zu ermöglichen.

6.7.1. Kausale Blöcke `Blocks`

In Modelica sind nicht nur die bisher sehr ausführlich besprochenen akausalen Komponenten möglich, sondern es können auch kausale Blöcke ähnlich wie in Simulink verwendet werden. Diese Modelle²³ haben fix vorgegebene Signalein- und -ausgänge. Sie können zur Ansteuerung von gesteuerten Quellen wie etwa einer gesteuerten Spannungs- oder Momentenquelle verwendet werden. Somit können beliebige Signalformen grafisch zusammengestellt werden. Gleichmaßen verfügen Sensoren über Signalausgänge. Über die Blöcke ist hier wiederum eine Weiterverarbeitung bzw. Filterung dieser Sensorwerte möglich.

Kausale Blöcke stellen grundsätzlich für den Modelica Compiler nur Bereiche von Gleichungen dar, bei denen die Kausalität schon vorgegeben ist - also Zuweisungen. Daher sind sie als eine Art Vereinfachung zu den akausalen Elementen zu sehen. Auch die Schnittstellen zwischen den kausalen Blöcken stellen Vereinfachungen von physikalischen Schnittstellen dar. Die Variablen werden zwischen Ein- und Ausgängen gleichgesetzt, sind also intensive bzw. Potential-Variablen.

6.7.2. Komposition und `Connect Statement`

Nachdem die grundlegenden Elemente von Modelica mittlerweile klar sein sollten, werden wir das erlangte Wissen nun an einer einfachen elektrischen Schaltung anwenden. Der Modelica Code mit welchem die in [Abbildung 6.11](#) dargestellte Schaltung modelliert werden kann ist in [Code 6.19](#) aufgezeigt.

```
1 model el_circuit
2     import an = Modelica.Electrical.Analog;
3     an.Basic.Ground ground;
4     an.Basic.Capacitor C (C=22E-6);
5     an.Basic.Resistor R2 (R=1E3);
6     an.Basic.Inductor L (L=1E-3);
7     an.Basic.Resistor R1 (R=10);
```

²³Zu finden unter `Modelica.Blocks`

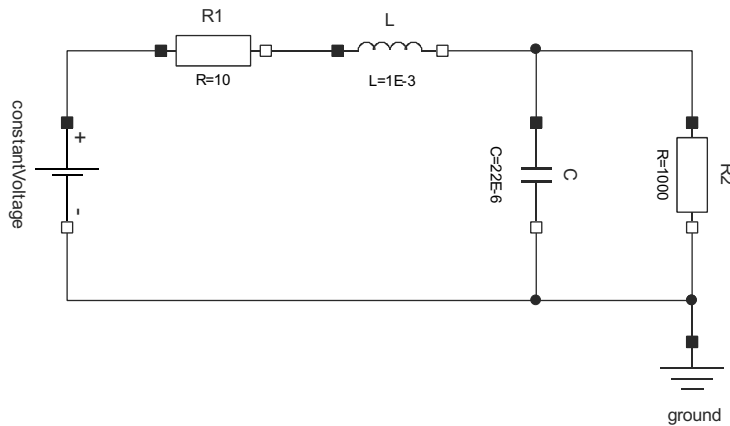


Abb. 6.11.: Einfache elektrische Schaltung erstellt in Dymola

```

8   an.Sources.ConstantVoltage constantVoltage(V=30);
9   equation
10  connect (R1.n, L.p);
11  connect (L.n, R2.p);
12  connect (C.p, L.n);
13  connect (R2.n, C.n);
14  connect (constantVoltage.p, R1.p);
15  connect (constantVoltage.n, C.n);
16 end el_circuit

```

Code 6.19: Eine einfache elektrische Schaltung in Modelica.

In Zeile 1 wird wiederum der Namen des Modells definiert. Zeile 2 importierte alle Elemente der analogen elektrischen Library. Alternativ dazu könnte man alle „an.“ in den Komponentendefinitionen durch `Modelica.Electrical.Analog` ersetzen. Wie schon angedeutet werden in den Zeilen 3 bis 8 die einzelnen Komponenten des Modells definiert. Dabei wird als erstes der Typ der Komponente (z.B. `an.Basic.Resistor` für einen Widerstand) angegeben, darauf folgt die Bezeichnung der Komponente, gefolgt von den optionalen *Modifiern* welche die Parameter der verwendeten Komponenten modifizieren können.

Hier lassen sich bereits weiter Eigenschaften der Objektorientiertheit erkennen. So werden in diesem Modell `el_circuit` andere Modelle wie die zuvor definierten Widerstände und Kapazitäten verwendet. Dies entspricht bereits dem objektorientierten Konzept der Komposition. Dies wurde auch schon bei der Modellierung von Widerstand und Kapazität verwendet in dem die Objekte `Pin` darin instanziiert wurden.

Im zweiten Teil (Zeile 10 bis 16) des Modells werden die Verbindungen des Modells über das `connect` Statement definiert. Dabei werden die einzelnen Pins der verschiedenen Elemente mit idealen Leitern verbunden, wie es aus einem Schaltplan bekannt ist. Dabei

können immer nur zwei Pins verbunden werden. Dies ist auch der Grund dafür, dass in aktuellen Modelica-Umgebungen immer zwei Connectoren verbunden werden müssen. Es kann also z.B. kein Pin mit einer Verbindung verschalten werden.

Natürlich ist es seit längerer Zeit in einer Vielzahl von Modelica Simulationsumgebungen möglich diese Modelle, wie von anderen Simulationsumgebungen bekannt, direkt grafisch zu erstellen. Dabei ist es interessant zu wissen, dass die Simulationsumgebung im Hintergrund automatisch den entsprechenden Modelica Code erstellt. Im Normalfall ist dann auch möglich diesen Code anstelle der grafischen Darstellung zu bearbeiten. Dies bietet in manchen Anwendungsfällen deutliche Vorteile. Die Änderungen werden in beide Richtungen übernommen. Zusätzlich ist seit Modelica 3.0 auch die grafische Darstellung der Icons (also deren Position, Orientierung etc.) direkt in Modelica definiert. Dazu werden Annotations verwendet, welche in [Code 6.19](#) der Übersichtlichkeit halber entfernt wurden. Mehr zu Annotations in [Unterabschnitt 6.7.14](#).

6.7.3. Vektoren { } und Matrizen []

Anders als MATLAB unterscheidet Modelica zwischen Vektoren und Matrizen. Dabei erfolgt die Unterteilung bei der Definition der Variablen wie folgt:

```
1 Real v{5};           // Vektor mit fünf Elementen
2 Real m1[3,4];       // Matrix mit drei Zeilen und fünf Spalten
3 Real m2[5,1];       // Matrix mit fünf Zeilen und einer Spalten
4 Real m3[5,3];       // Matrix mit fünf Zeilen und drei Spalten
5 Real m4[2,3,4];     // dreidimensionalen Matrix
```

Code 6.20: Definition von Vektoren und Matrizen in Modelica.

Dabei ist auf den ersten Blick nicht ersichtlich was der Unterschied zwischen einem Vektor und der Matrize mit einer Spalte ist. Diese werden aber vom Modelica Compiler unterschiedlich behandelt. Vektoren können nur mit dem Array-Operator { . . . } Werte zugewiesen werden. Diese werden immer durch Beistriche getrennt. Im Gegensatz dazu können Matrizen nur mit dem Matrizen-Operator [. . .] befüllt werden. Im Matrizen-Operator werden Elemente einer Zeile mit Beistrichen „ , “ getrennt, ein Zeilensprung erfolgt über einen Strichpunkt „ ; “, Also gleich wie in MATLAB.

Bei der Multiplikation zweier Vektoren wird automatisch ein Skalarprodukt angewandt. Über `cross(v, v)` kann ein Vektorprodukt berechnet werden. Da Vektoren als solche definiert sind, muss nicht transponiert werden. Bei der Rechnung mit Matrizen muss man sich händisch um die korrekten Dimensionen kümmern.

Es ist nicht möglich einer Matrix die Werte eines Vektors zuzuweisen. Die Gleichung `m2 = v` führt also zu Fehlern, obwohl die Dimensionen an sich korrekt wären. Vektoren können allerdings über den Matrizen-Operator in Matrizen umgewandelt werden.

Demzufolge ist $m2 = [v]$ eine gültige²⁴ Gleichung. Es können auch mehrere Vektoren zu einer Matrix zusammengefasst werden, indem sie durch Beistriche getrennt werden. Ein Beispiel dafür wäre $m3 = [v, 2*v, 3*v]$.

Das Zugreifen auf einzelne Elemente von Vektoren und Matrizen funktioniert über eckige Klammern “[]“, wie im Folgenden demonstriert. Dabei können über den “:“ Operator mehrere Indizes auf einmal angesprochen werden. Über “[:]“ greift man auf alle Elemente eines Vektors zu.

```

1  Real v{5};    // Vektor mit fünf Elementen
2  equation
3  v[1] = 1;
4  v[2] = 3;
5  v[3:5] = {1, 2, 3};

```

Code 6.21: Zugreifen auf Elemente eines Vektors

Vektoren können auch verwendet werden, um Vektoren von Komponenten zu definieren. Dabei wird genau wie bei der Definition einer Variable an die Definition der Komponente eine geschwungene Klammer { } angefügt. Somit kann eine Anzahl von Modellen auf einmal definiert werden und die Anzahl der Modelle als Parameter vorgegeben werden. Eine Verschaltung dieser wird dann üblicherweise über Schleifen erledigt (siehe [Unterabschnitt 6.7.7](#)).

6.7.4. Strukturen: record

Ein record in Modelica ist ähnlich der aus anderen Programmiersprachen bekannten Strukturen (struct). Dabei können Variablen von verschiedenem Datentyp und Namen zusammengefasst werden. Diese werden dann gemeinsam behandelt. Die folgenden Beispiele sind der Modelica Language Specification [[Ass10](#)] entnommen.

Das erste Beispiel ist ein typischer Anwendungszweck für Strukturen. Es werden zusammengehörige Variablen in einer Struktur zusammengefasst. In diesem konkreten Beispiel handelt es sich um eine komplexe Zahl, welche aus Real- und Imaginärteil zusammengesetzt wird.

```

1  record Complex "Complex number"
2    Real re "real part";
3    Real im "imaginary part";
4  end Complex;

```

Code 6.22: Strukturen zur Definition von komplexen Zahlen

²⁴Genau genommen entstehen daraus fünf Gleichungen

Eine weit verbreitete Anwendung eines `record` ist im folgenden Beispiel aufgezeigt. Dabei werden die für ein Modell nötigen Parameter in einem `record` zusammengefasst und mit dem Modell gemeinsam in einem `Package` abgelegt. Der Vorteil an dieser Vorgehensweise ist, dass in den erstellten Libraries verschiedenen Parametersätze für die Modelle abgelegt werden können. Diese können entweder grafisch oder textuell in den erstellten Modellen instanziiert werden und somit sind die Modelle parametrisiert.

```
1 package Motors
2   record MotorData "Data sheet of a motor"
3     parameter Real inertia;
4     parameter Real nominalTorque;
5     parameter Real maxTorque;
6     parameter Real maxSpeed;
7   end MotorData;
8
9   model Motor "Motor model" // using the generic MotorData
10    MotorData data;
11    ...
12  equation
13    ...
14  end Motor;
15
16  record MotorI123 = MotorData( // data of a specific motor
17    inertia      = 0.001,
18    nominalTorque = 10,
19    maxTorque    = 20,
20    maxSpeed     = 3600) "Data sheet of motor I123";
21
22  record MotorI145 = MotorData( // data of another specific
23    motor
24    inertia      = 0.0015,
25    nominalTorque = 15,
26    maxTorque    = 22,
27    maxSpeed     = 3600) "Data sheet of motor I145";
28 end Motors
```

Code 6.23: Parametrieren von Motoren über Strukturen

6.7.5. Bedingte Ausdrücke: `if/then/elseif/then/else`

In Modelica können Gleichungen, welche eine Variable errechnen, von einer Bedingung abhängig errechnet werden. Ein vereinfachtes Beispiel aus [And09] zeigt die Berechnung der Reifenumfangskraft abhängig davon ob Kontakt mit dem Untergrund besteht oder nicht. Besteht Kontakt (`Contact == true`), wird die Gleichung $f_{Tire} = -f_N \cdot \mu \cdot v_{Slip}$ verwendet. Ist kein Kontakt gegeben, der Reifen hat also vom Boden abgehoben, wird die Umfangskraft auf null gesetzt, da keine Kraft übertragen werden kann.

```

1
2  fTire = if Contact then -fN * mu * vSlip else 0;

```

Code 6.24: Bedingte Gleichung zur Berechnung einer Reifenkraft

Die Formulierung ist etwas unüblich, wenn man Programmiersprachen wie C oder Java gewohnt ist, aber durchaus eingängig. Die Umschaltung erfolgt hier basierend auf der booleschen Variable `Contact`. Dies kann auch während der Simulationszeit passieren. Da es sich hier um ein diskretes Umschalten handelt, kann über verschiedene Funktionen beeinflusst werden, wie der Solver diese Umschaltungen behandelt. Dazu gehören unter anderem die Funktionen `noEvent` und `smooth`, wobei hier nicht weiter auf deren Funktion eingegangen werden soll. Diese kann unter anderem in [Ass10] genau nachgelesen werden.

Ähnlich wie Gleichungen abhängig von einem booleschen Ausdruck umgeschaltet werden können, kann dies auch mit Modellen geschehen. Dies kann allerdings nicht mehr während der Laufzeit der Simulation passieren, sondern muss beim Übersetzen erfolgen. Daher muss die Bedingung in der Definition der Komponenten beim Zeitpunkt der Modellübersetzung auswertbar sein. Dazu ein Beispiel aus [Ass10].

```

1  parameter Integer level=1;
2  Level1 component1(J=J) if level==1 "Conditional component";
3  Level component2 if level==2, component3 if level==3;

```

Code 6.25: Bedingte Definition einer Komponente

6.7.6. Eventbasierte Gleichungen: when

Neben den bedingten Ausdrücken, gibt es auch eventbasierte Gleichungen. Diese werden über das `when` Statement definiert. Im Unterschied zum `if` Statement, kann das `when` Statement nur im Gleichungsabschnitt verwendet werden. Der grundlegende Unterschied zwischen `if` und `when` Statement ist, dass eine der Gleichungen im `if` Statement in jedem Simulationsschritt ausgeführt werden. Beim `when` werden die Gleichungen genau dann einmalig ausgeführt wenn die Bedingung von `false` auf `true` springt.

Das Beispiel unterhalb soll diesen Zusammenhang erläutern. Dabei wird die boolesche Variable `b` bei jedem Nulldurchgang des Sinus mit negativer Steigung `u` invertiert. Das Ergebnis dieses Modells ist in [Abbildung 6.12](#) dargestellt.

```

1  model whendemo
2    parameter Real A=1.5, w=4;
3    Real u;
4    Boolean b;

```

```

5
6 equation
7   u = A*Modelica.Math.sin(w*time);
8   when u < 0 then
9     b = not pre(b);
10  end when;
11 end whendemo;

```

Code 6.26: Beispielhafte Anwendung des when Statements

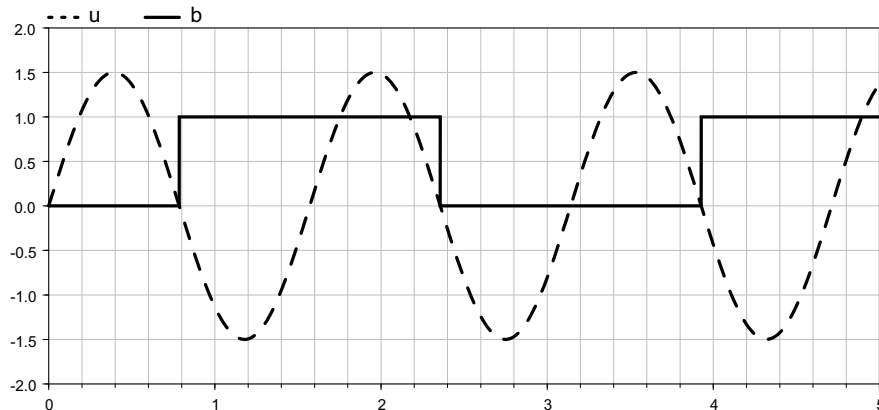


Abb. 6.12.: Ergebnis des einfachen when Beispiels

Initialisierung: initial equation oder start

In [Unterabschnitt 6.7.6](#) haben wir uns nicht darum gekümmert, wieso die Variable `b` mit `false` beginnt. Der Grund für dieses Verhalten ist einfach erklärt. Startwerte nicht explizit initialisierter Variablen werden üblicherweise automatisch auf 0 bzw. `false` gesetzt. Nun wollen wir uns verschiedene Varianten ansehen, wie wir das Startverhalten des Modelles beeinflussen können. Dazu wird die der Typ der Variable `b` von `Boolean` auf `Real` geändert, da sonst z.B. das `reinit` Statement nicht funktioniert. Zusätzlich ändern wir den `not ()` Operator in eine Multiplikation mit `-1`, da der `not ()` Operator nicht auf einen `Real` angewandt werden kann.

Im folgenden Beispiel verwenden wir wiederum ein `when` Statement, um der Variable einen definierten Startwert mitzugeben. Hier wird `initial ()` verwendet, um eine Gleichung einmalig während des Initialzeitpunktes auszuführen. Dabei könnte man annehmen, dass es hier ausreichend ist statt `reinit(b, 1)` ein einfaches `b = 1` einzufügen. Dies ist allerdings in Modelica nicht erlaubt, da es hier möglich wäre, dass mehrere `when` Statements gleichzeitig aktiviert werden und somit nicht mehr klar wäre, welche Gleichung gültig ist. Deshalb muss an dieser Stelle das `reinit` verwendet werden.


```

1 model whendemo
2   parameter Real A=1.5;
3   parameter Real w=4;
4   Real u;
5   Real b;
6
7 equation
8   u = A*Modelica.Math.sin(w*time);
9
10  when initial() then
11    reinit(b, 1);
12  end when;
13
14  when u < 0 then
15    b = -pre(b);
16  end when;
17 end whendemo;

```

Code 6.27: Initialisierung von Code 6.26 über das `reinit` Statment

Eine weitere Möglichkeit eine Variable zu initialisieren ist direkt bei deren Definition. Dies ist im folgenden Beispiel aufgezeigt. Zusätzlich kann hier definiert werden, ob es sich um einen fix vorgegebenen Wert, oder um einen Schätzwert handelt. Der Schätzwert wird bei der Initialisierung mehrerer zusammenhängender Größen verwendet, um einen Startwert für die nötige Iteration zu liefern. So kann diese schneller konvergieren oder in schwierigen Fällen überhaupt erst ein Ergebnis liefern. In diesem Fall ist das Attribut `fixed` zuvor nicht sinnvoll eingesetzt, da es keinen Zusammenhang zu anderen Variablen gibt. Es hat also auch keinen Einfluss, ob dieses Attribut auf `true` oder `false` gesetzt wird. In anderen Fällen kann es aber wichtig sein.

```

1 model whendemo
2   ...
3   Real u;
4   Real b (start = 1, fixed = true);
5
6 equation
7   ...
8 end whendemo;

```

Code 6.28: Initialisierung von Code 6.26 über das `start` Attribut

Eine alternative Möglichkeit ist es, die Variablen im `initial equation` Abschnitt zu definieren. Diese Form der Initialisierung ist im folgenden Beispiel aufgezeigt.

```

1 model whendemo
2   ...
3   Real u;

```

```

4   Real b;
5
6   initial equation
7     b = 1;
8
9   equation
10    ...
11  end whendemo;

```

Code 6.29: Initialisierung von Code 6.26 über `initial equation`

Welche der verschiedenen Formen der Initialisierung verwendet wird, hängt von den Vorlieben des Benutzers und den Möglichkeiten der einzelnen Varianten ab.

6.7.7. Schleifen: `for/in/loop`

Ähnlich wie in bekannten Programmiersprachen wie C, C++ oder Java ist es auch in Modelica möglich Schleifen zu definieren. Dabei wird die Syntax in den unterhalb aufgeführten Beispielen verwendet.

```

1   for i in 1:10 loop           // i takes the values 1,2,3,...,10
2   for r in 1.0 : 1.5 : 5.5 loop
3                                   // r takes the values 1.0, 2.5, 4.0, 5.5
4   for i in {1,3,6,7} loop      // i takes the values 1, 3, 6, 7

```

Code 6.30: Definition von `for` Schleifen

Schleifen sind z.B. nützlich, um große Anzahlen gleicher Modelle zu definieren und zu verknüpfen. Ein sehr gut ersichtliches Beispiel ist die Verknüpfung vieler einzelner Batteriezellen zu einem Batteriepack. Dabei werden oft etwa 100 gleiche Zellen verschalten. Hier jedes Element manuell einzeln zu definieren und zu verschalten ist keine sinnvolle Variante. Zusätzlich kann die Anzahl an Zellen so parametrierbar gemacht werden.

Ein fortgeschrittenes Beispiel ist in [ECK⁺11] zu finden. Hier wird ein Vektor von Modellen einer Batteriezelle definiert. Somit können die einzelnen Zellen über deren Indizes angesprochen werden. Anschließend werden die Batteriezellen verschalten. Zusätzlich ist auch noch die Möglichkeit gegeben diese seriell und parallel zu verschalten.

```

1   ...
2   equation
3     //series connection
4     for s in 1:ns-1 loop
5       connect(cell[s,1].pin_n, cell[s+1,1].pin_p);
6     end for;
7
8     //parallel connection

```

```

9   for p in 1:np-1 loop
10      for s in 1:ns loop
11          connect (cell[s,p].pin_p, cell[s,p+1].pin_p);
12          connect (cell[s,p].pin_n, cell[s,p+1].pin_n);
13      end for;
14  end for;
15
16  //connector connection
17  for s in 1:ns loop
18      connect (cell[s,1].pin_p, pin_pCell[s]);
19      connect (cell[s,1].pin_n, pin_nCell[s]);
20  end for;
21
22  //top connection
23  connect (cell[1, 1].pin_p, pin_pPackage);
24  //bottom connection
25  connect (cell[ns, np].pin_n, pin_nPackage);
26  //heatPort connection
27  connect (cell[:, :].heatPort, heatPort[:, :]);
28  ...

```

Code 6.31: Anwendung von `for` Schleifen in der Verbindung von vektorisierten Elementen

So können Batteriepacks mit unterschiedlich parametrisierten Zellen (etwa verschiedene Innenwiderstände oder Kapazitäten) definiert werden. Sind alle Zellen zumindest in der Simulation exakt gleich kann die Spannung bzw. der Strom mit der Anzahl an in serie bzw. parallel geschalteten Zellen multipliziert werden. Das beschleunigt die Simulation durch die entfallenden Gleichungen und Zustände deutlich.

Hier bleibt anzumerken, dass für diese Art der Modelldefinition kein grafisches Modell erstellt wird. Dies geschieht, weil zu dem aus dem Code erstellten Modell keine grafische Information hinterlegt wird. Diese wird bei der grafischen Modellierung von der Modellierungsumgebung automatisch erstellt und fehlt deshalb hier komplett. Etwas mehr Information dazu ist in [Unterabschnitt 6.7.14](#) einsehbar.

6.7.8. Globale Variablen: `inner` und `outer`

Globale Variablen sind in praktisch jeder Art der Programmierung eher sparsam einzusetzen, da sie oft zu einer schwer erkennbaren Fehlerquelle werden. Das gleiche gilt grundsätzlich auch in der Modellierung. Allerdings gibt es hier einige Fälle, in welchen der Einsatz der globalen Variablen nicht nur rechtfertigbar ist, sondern wirklich Sinn macht. Gute Beispiele dafür sind etwa eine Umgebungstemperatur, ein Umgebungsdruck, oder ein Vektor, welcher die Gravitation beschreibt. Diese Variablen wirken auf alle Objekte eines Modells. Daher bietet es sich an dafür globale Variablen zu verwenden.

Die Syntax einer solchen Modellstruktur ist unterhalb dargestellt und aus [Ass10] entnommen. Dabei wird die Variable `T0` definiert, welche z.B. eine Umgebungstemperatur darstellen könnte. Sie wird in der Klasse `B` als globale Variable definiert, was über das vorgestellte `inner` erreicht wird. In den Objekten `a1` und `a2` der Klasse `A` wird auf die Variable aus der Klasse `B` zugegriffen, was über das vorgestellte `outer` geschieht. Dadurch sind alle drei Variablen miteinander verknüpft und somit haben sie immer denselben Wert. Abhängig von der Definition der `inner` Variable kann diese konstant, oder über die Zeit veränderlich sein. Letzteres ist es hier der Fall.

```
1  class A
2    outer Real T0;
3    ...
4  end A;
5
6  class B
7    inner Real T0;
8    A a1, a2;    // B.T0, B.a1.T0 and B.a2.T0 is the same variable
9    ...
10 end B;
```

Code 6.32: Definition von globalen Variablen über `inner/outer` Statements

Diese Verknüpfung von Variablen über `inner/outer`, ist auch über mehrere Hierarchiestufen hinweg möglich. Eine `outer` Variable wird grundsätzlich immer mit der ihr am nächstgelegenen gleichnamigen `inner` Variable verknüpft.

6.7.9. Prozedurale Algorithmen: `algorithm`

In einer `algorithm` Sektion kann Code sehr ähnlich einer üblichen Programmiersprache eingegeben werden. Dabei werden keine Gleichheitszeichen “=” verwendet, sondern Zuweisungsoperatoren “:=“. Dadurch passieren `algorithm` Abschnitte die symbolische Vorverarbeitung (mehr dazu in [Abschnitt 6.9](#)), welche die sonst verwendeten Gleichungen in eine ausführbare Form bringt, unberührt. Sie werden zwar zwischen den anderen Gleichungen einsortiert, aber die interne Reihenfolge bleibt erhalten. Die Zuweisungen werden also exakt so ausgeführt, wie sie im Code hinterlegt sind.

Grundsätzlich sind `algorithm` Abschnitte so sparsam wie möglich einzusetzen. Die große Stärke der objektorientierten Modellierung liegt in der symbolischen Vorverarbeitung und exakt dieser Teil wird mit den `algorithm` Abschnitten größtenteils umgangen. Fälle in welchen ein `algorithm` nötig werden kann, sind folgende:

- Wenn `while` Schleifen nötig sind. Diese sind nur in `algorithm` Abschnitten möglich.
- Ein diskreter Algorithmus wie z.B. ein digitaler Regler soll implementiert werden,

bei welchem die Reihenfolge der Auswertung der Zuweisungen vom Benutzer definiert werden muss.

- Die Formulierung von verschachtelten `if` Statements in Gleichungen zu umständlich wird.
- Immer, wenn in Modelica eine Funktion `function` implementiert wird.

6.7.10. Funktionen: `function`

Funktionen sind in Modelica ganz ähnlich zu den aus Programmiersprachen wie C bekannten Funktionen. In Modelica haben sie folgende Eigenschaften.

- Variablen müssen `public` oder `protected` definiert sein.
- Jede der Variablen muss mit einem Prefix versehen sein. Möglich sind `input`, `output`, `parameter` oder `constant`.
- Es sind praktisch alle Operatoren verfügbar, welche in Modellen verwendbar sind. Ausnahmen sind der `der()` Operator und andere Operatoren, welche im Zusammenhang mit dem Eventhandling stehen, wie z.B. der `pre()` Operator.
- Modelica Funktionen sind frei von Seiteneffekten, entsprechen also mathematischen Funktionen. Sie liefern bei gleichen Eingangsvariablen unter allen Umständen gleiche Ausgangsvariablen²⁵. Es können daher in Funktionen keine Variablen gespeichert werden.
- Sie können nur einen `algorithm` und keinen `equation` Abschnitt enthalten.

Der generelle Aufbau einer Modelica Funktion ist im Folgenden Beispielcode zu sehen (aus [Ass10]).

```

1  function funktionname
2    input  TypeI1 in1;
3    input  TypeI2 in2;
4    input  TypeI3 in3 := default_expr1 "Comment" annotation(...);
5    ...
6    output TypeO1 out1;
7    output TypeO2 out2 := default_expr2;
8    ...
9  protected
10   <local variables>
11   ...
12  algorithm
13   ...
14   <statements>
15   ...

```

²⁵Ausnahme sind hier externe Funktionen, welche Seiteneffekte haben können.

```
16 end funktionname;
```

Code 6.33: Aufbau einer Funktion

Ein Beispiel zur Berechnung der Summe der Elemente eines komplexen Vektors ist unterhalb zu sehen.

```
1 function 'sum' "Return sum of complex vector"
2   input Complex v[:] "Vector";
3   output Complex result "Complex sum of vector elements";
4   algorithm
5     result:=Complex(0);
6     for i in 1:size(v,1) loop
7       result:=result + v[i];
8     end for;
9   end 'sum';
```

Code 6.34: Beispiel einer Funktion

Funktionen werden in Modelica wie in Programmiersprachen verwendet. Wird ein Codesegment mit definiertem Input und Output an mehreren Stellen benötigt, kann dies über eine Funktion einmalig realisiert werden und an den verschiedenen Stellen verwendet werden.

6.7.11. Die Vererbung: extends

In diesem Abschnitt sollen einfache passive elektrische Zweipole²⁶ in Modelica unter der Ausnutzung der objektorientierten Möglichkeiten der Sprache beschrieben werden. Es wurde dazu absichtlich auf ein sehr einfaches Beispiel zurückgegriffen, um die Konzentration auf die Sprachkonstrukte und nicht auf das eigentliche Modell zu lenken.

Das Modell eines elektrischen Widerstands (Abbildung 6.13) könnte in Modelica aussehen wie in Code 6.35 beschrieben.

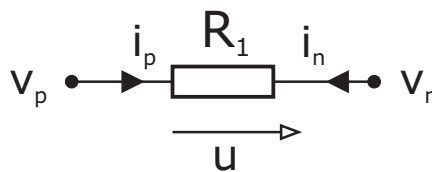


Abb. 6.13.: elektrischer Widerstand

²⁶also elektrische/elektronische Bauelemente mit zwei Pins

```

1 model Resistor "Ideal resistor"
2   Pin p, n;
3   Voltage u;
4   parameter Resistance R;
5   equation
6     u = p.v - n.v;
7     p.i + n.i = 0;
8     R*p.i = u;
9 end Resistor;

```

Code 6.35: Definition des Widerstands

Dabei wird in Zeile 1 definiert, dass es sich um ein Model mit dem Namen „Resistor“ handelt. Die unter Anführungszeichen folgende Anmerkung gibt eine minimale Beschreibung des modellierten Elementes an. In den Zeilen 2, 3 und 4 werden die einzelnen „Bestandteile“ oder Attribute des Widerstands spezifiziert. Er wird also aus den beiden Pins p und n , einer Spannung u , und einem Parameter Widerstand (engl. Resistance) R zusammengesetzt.

In Zeile 5 wird über das `equation` statement definiert, dass nun Gleichungen folgen, welche das Verhalten des Widerstands beschreiben. Um die gängige Variable für die Beschreibung eines Widerstands zur Verfügung zu stellen, und weil Modelica grundsätzlich mit Potentialvariablen arbeitet, muss der Spannungsabfall welcher in Zeile 3 als Spannung definiert wurde aus der Differenz der Potentiale berechnet werden (Zeile 6). In Zeile 7 wird angegeben, dass der in den Widerstand fließende Strom auch wieder aus dem anderen Pin fließen muss, der Widerstand also keinen Strom „konsumieren“ kann. Zeile 8 ist schließlich die Definition des ohmschen Gesetz. Noch einmal soll an dieser Stelle darauf hingewiesen werden, dass es sich hierbei um akausale Gleichungen und nicht um Zuweisungen handelt.

Es wurden in dieser Beschreibung einige Elemente verwendet, die zwar einem Ingenieur mit elektrotechnischem Hintergrund etwas sagen, die aber Modelica ohne eine gesonderte Definition unbekannt sind. Diese sind der `Pin` und die elektrische Spannung `Voltage`, in den folgenden Codesegmenten 6.36 und 6.37 definiert sind.

```

1 connector Pin
2   Voltage v;
3   flow Current i;
4 end Pin;

```

Code 6.36: Definition des Pins

Der `Pin` beschreibt, wie dessen Name schon sagt, ein Verbindungselement das durch ein elektrisches Potential (`Voltage v`) und einen Strom (`Current i`) beschrieben wird. Das Statement `flow` vor dem Strom gibt an, dass es sich dabei um eine extensive Variable handelt. Die Spannung bzw. das elektrische Potential v wird ohne weiteres Statement

angegeben und dadurch automatisch als intensive behandelt, wobei der in Modelica gängige Ausdruck eine *Accross* Variable ist. Aus der Definition von intensiven und extensiven Variablen können die Gleichungen, welche die Verbindung zweier Konnektoren (Pins) beschreiben automatisch erstellt werden (siehe dazu [Unterabschnitt 6.4.1](#)).

```

1 type ElectricPotential = Real
2   (final quantity="ElectricPotential",
3    final unit="V");
4 type Voltage = ElectricPotential;

```

Code 6.37: Definition der elektrischen Spannung

Es fehlt allerdings immer noch die Beschreibung des `Voltage` Elements. Dies ist war mehr von kosmetischer Natur als die bisherigen Elemente, kann den Modellierer aber trotzdem wesentlich bei seiner Arbeit unterstützen. Es wird [Code 6.37](#) in Zeile 1 wiederum die Bezeichnung der neuen Klasse (Keyword `type`) auf „ElectricPotential“ festgelegt. Außerdem wird definiert, dass es sich dabei um einen Datentyp `Real` handelt, was in Modelica einer kontinuierlichen Variable entspricht, die mit doppelter Präzision (wie eine Variable vom Typ `double` in MATLAB) berechnet wird. Über das Keyword `quantity` wird die Bezeichnung des Datentyps festgelegt, während über `unit` die Einheit definiert werden kann. Somit ist es bei einer konsequenten Verwendung von Datentypen möglich, Fehler über einen automatischen Vergleich von Einheiten und physikalischer Größe (`quantity`) zu finden. Dieser ist allerdings nicht von allen Simulationsumgebungen implementiert. In Zeile 4 werden der elektrischen Spannung die gleichen Eigenschaften wie dem elektrischen Potential zugewiesen.

Nun wollen wir damit fortfahren ein Modell für einen Kondensator wie er in [Abbildung 6.14](#) dargestellt ist zu definieren. Das Modell ist in [Code 6.38](#) aufgezeigt.

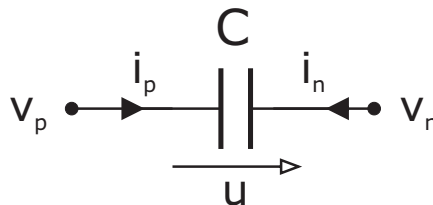


Abb. 6.14.: Kondensator

```

1 model Capacitor "Ideal capacitor"
2   Pin p, n;
3   Voltage u;
4   parameter Capacitance C;
5   equation
6     u = p.v - n.v;

```



```

7   p.i + n.i = 0;
8   C*der(u) = p.i;
9   end Capacitor;

```

Code 6.38: Definition des Kondensators

Vergleicht man nun [Code 6.35](#) und [Code 6.38](#) so erkennt man schnell, dass diese viele gemeinsame Codezeilen haben. Unterschiede können nur in den Zeilen 4 und 8 ausgemacht werden. Dies widerspricht dem Grundsatz der Objektorientiertheit, da hier mehrfach vorkommende Codesegmente vermieden werden sollen. Es kommt einem also recht schnell das Konzept der Vererbung wieder in den Sinn, womit ein Modell Eigenschaften von einem anderen Modell erben und somit übernehmen kann. Der Modellierer hat nun also die Aufgabe eine möglichst umfangreiche Zusammenfassung der gemeinsamen Eigenschaften der zu beschreibenden Modelle zu erstellen. In diesem Beispiel umfasst das alle Zweipole. Diese können durch das in [Abbildung 6.15](#) dargestellte Element beschrieben werden.

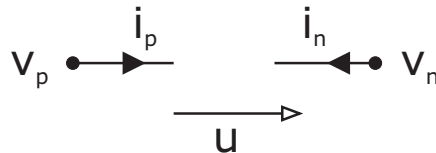


Abb. 6.15.: Zweipol

Das zu [Abbildung 6.15](#) passende Modell ist in [Code 6.39](#) aufgezeigt.

```

1   partial model OnePort
2     Pin p, n;
3     Voltage u;
4     equation
5       u = p.v - n.v;
6       p.i + n.i = 0;
7   end OnePort;

```

Code 6.39: Definition des Grundgerüsts eines Zweipols oder Eintors (Oneport)

Man kann aus dem Vergleich von [Code 6.35](#), [6.38](#) und [6.39](#) einfach erkennen, dass der Zweipol alle Elemente und Gleichungen umfasst, welche für Widerstand und Kondensator gleich sind. Wichtig zu erkennen ist, dass es sich bei diesem Modell um ein `partial` Modell handelt. Das Keyword `partial` bedeutet hier, dass es sich bei diesem Modell um ein nicht direkt simulierbares Modell handelt (siehe Abstraktion auf Seite 169). Es werden hier für die beschriebenen Variablen nicht genügend Gleichungen definiert, was eine Simulation unmöglich macht. Grundsätzlich sind für die Berechnung eines Zweipols zwei Gleichungen und zwei Unbekannte nötig. Hier wurde allerdings zusätzlich zu

den beiden Unbekannten noch der Spannungsabfall u als Variable definiert, was eine zusätzliche dritte Gleichung nötig macht.

Will man dieses partielle Modell verwenden, um einen Widerstand (Code 6.40) und einen Kondensator (Code 6.41) zu beschreiben, kann das `extends` Statement verwendet werden. Über das `extends` Statement wird angegeben, dass das partielle Modell `OnePort` erweitert werden soll. Somit erben die beiden Modelle alle Eigenschaften des `OnePort`. Zusätzlich wird der `parameter` für die beiden Modelle unterschiedlich definiert, wobei man beachten sollte, dass sich hier auch die Datentypen der Parameter unterscheiden. Zusätzlich wird die bisher fehlende Gleichung definiert. Somit handelt es sich um ein simulierbares Modell, wodurch das Keyword `partial` entfällt. Die Simulationsumgebung fügt bei der Verwendung des Widerstands oder Kondensators alle Variablen und Gleichungen zusammen und erstellt somit das ausführbare Modell.

```
1 model Resistor "Ideal resistor"
2   extends OnePort;
3   parameter Resistance R;
4   equation
5     R*p.i = u;
6 end Resistor;
```

Code 6.40: Objektorientierte Beschreibung eines Widerstands

```
1 model Capacitor "Ideal capacitor"
2   extends OnePort;
3   parameter Capacitance C;
4   equation
5     C*der(u) = p.i;
6 end Capacitor;
```

Code 6.41: Objektorientierte Beschreibung eines Kondensators

Auf diesem Weg können - wie bereits erwähnt - alle elektrischen Elemente mit zwei Pins (Induktivität, Diode, Schalter etc.) modelliert werden. Dies ist aber nicht in allen Domänen der Fall. Wenn man versucht alle Komponenten aus [Abbildung 6.8](#) von einer Basisklasse abzuleiten, wird man erkennen, dass man dabei auf unüberwindbare Hindernisse stößt. Es gibt hier zwischen der 1D translatorischen Mechanik und den elektrischen Bauelementen grundlegende Unterschiede. Die flow Variable ist im elektrischen Beispiel der Strom. Dieser bleibt auf jeden Fall vor und nach dem Element gleich. Betrachtet man das mechanische Beispiel ist die flow Variable die Kraft. Diese ist zwar bei Feder und Dämpfer vor und nach dem Element dieselbe, allerdings ist dies bei der Masse nicht der Fall, weil für die Beschleunigung Kraft nötig ist. Bei einer wirkenden Beschleunigung ist die Kraft vor und nach der Masse also unterschiedlich. Im Falle der Masse ist die Ausdehnung konstant, dies ist bei Feder und Dämpfer nicht der Fall. Daher sind auch in der Modelica Standard Library zwei unterschiedliche partielle Modelle

zur Beschreibung der 1D Translation vorgesehen (siehe [Code 6.42](#) bzw. [Code 6.43](#))

```

1 partial model PartialRigid
2   "Rigid connection of two translational 1D flanges "
3   SI.Position s
4   "Absolute position of center of component (s = flange_l.s + L
5     /2 = flange_r.s - L/2)";
6   parameter SI.Length L(start=0)
7     "Length of component, from left flange to right flange (=
8     flange_l.s - flange_r.s)";
9
10  equation
11  flange_l.s = s - L/2;
12  flange_r.s = s + L/2;
13 end PartialRigid;

```

Code 6.42: Partielle Klasse welche als Basisklasse für die Masse verwendet wird.

```

1 partial model PartialCompliant
2   "Compliant connection of two translational 1D flanges"
3   SI.Distance s_rel(start=0) "Relative distance (= flange_r.s -
4     flange_l.s)";
5   SI.Force f "Force between flanges (positive in direction of
6     flange axis R)";
7
8  equation
9  s_rel = flange_r.s - flange_l.s;
10 flange_r.f = f;
11 flange_l.f = -f;
12 end PartialCompliant;

```

Code 6.43: Partielle Klasse welche als Basisklasse für Feder und Dämpfer verwendet wird.

6.7.12. Austausch von Submodellen: `replacable/redeclare`

Über den `redeclare` Befehl können in Kombination mit der Vererbung Teile von Modellen ausgetauscht werden. Dies ist vor allem sinnvoll wenn ähnliche Modelle in speziellen Ausprägungen oder verschiedenen Detaillierungsgraden untersucht werden sollen. So wäre es zum Beispiel möglich die in [Abbildung 6.11](#) auf Seite 173 dargestellt Schaltung mit detaillierteren Komponenten zu versehen. Eine Möglichkeit wäre ein Widerstand, welcher die Änderung des Widerstandwertes über die Temperatur berücksichtigt, oder eine Induktivität, welche die magnetische Sättigung berücksichtigt.

Es gibt grundsätzlich zwei Möglichkeiten Modelle auszutauschen. Die erste wird im Folgenden dargestellt (siehe [Code 6.44](#)).

```
1 model RedclareR
2   import Modelica.Electrical.Analog.Basic.HeatingResistor;
3   extends el_circuit (redeclare HeatingResistor R1(alpha = 3.9e-3)
4     );
5   Modelica.Thermal.HeatTransfer.Sources.FixedTemperature
6     fixedTemperature(T=373.15)
7 equation
8   connect(fixedTemperature.port, R1.heatPort);
9 end RedclareR;
```

Code 6.44: Austauschen des Widerstandsmodells über das `redeclare` Statement

Hier wird zuerst das Modell des Widerstands importiert, welches die Temperaturabhängigkeit berücksichtigt. Danach werden alle Eigenschaften (inkl. Bauteile) des Modell `el_circuit` geerbt. In der Klammer dieser Vererbung wird angegeben, dass das Objekt `R1`, welches bisher ein „normaler“ Widerstand war, durch einen temperaturveränderlichen ersetzt wird. Dazu können angepasste Parameter übergeben werden. In Zeile 4 von [Code 6.44](#) wird der im neuen Modell vorhandene thermische Anschluss mit einer ebenfalls definierten Temperaturquelle verbunden.

Genau so ist es, ausgehend vom gleichen Schaltkreis, möglich die Induktivität zu ersetzen, wie es im Folgenden getan wird (siehe [Code 6.45](#)).

```
1 model RedeclareL
2   import Examples.Components.SaturatingInductor;
3   extends el_circuit (redeclare SaturatingInductor L(Inom = 1,
4     Lzer = 1.3e-3, Linf = 1e-6));
5 equation
6   ...
7 end RedeclareL
```

Code 6.45: Austauschen des Induktivitätsmodells über das `redeclare` Statement

Dabei werden die drei Parameter, welche im linearen Modell der Induktivität nicht nötig sind übergeben.

Anschließend kann im Modell, welches dieses Modell erweitert, die als `replaceable` definierte Klasse über folgenden Befehl durch eine Andere ersetzt werden.

```
1 model el_circuit
2   import an = Modelica.Electrical.Analog;
3
4   replaceable model replaceableResistor =
5     Modelica.Electrical.Analog.Basic.Resistor;
6   an.Basic.Ground ground;
```

```

7 | an.Basic.Capacitor C(C=22E-6);
8 | replaceableResistor R2(R=1E3);
9 | an.Basic.Inductor L(L=1E-3);
10 | replaceableResistor R1(R=10);
11 | an.Sources.ConstantVoltage constantVoltage(V=30);
12 |
13 | equation
14 | ...
15 | end el_circuit

```

Code 6.46: Austauschbarkeit der Widerstandsklasse über das `replaceable` Statement festlegen

Dabei werden wie vorher zusätzliche Parameter übergeben, dies kann allerdings nicht mehr für jedes Modell separat getan werden, sondern nur allgemein für einen Objekt-Typ.

Ähnlich dieser Vorgehensweise können nicht einzelne Objekte, sondern auch ganze Klassen ausgetauscht werden. Dazu muss die Klasse im Modell Grundmodell²⁷ als `replaceable` definiert werden. Zusätzlich müssen die austauschbaren Objekte üblicherweise mit einem neuen und austauschbaren Typ definiert sein. Da Modelle selten im vorhinein als `replaceable` definiert sind und das auch nicht in jedem Fall geändert werden kann. Dies ist im folgenden zu sehen.

```

1 | import Modelica.Electrical.Analog.Basic.HeatingResistor;
2 |
3 | extends el_circuit(redeclare model replaceableResistor =
4 |   HeatingResistor(alpha = 3.9e-3));

```

Code 6.47: Ersetzen der Widerstandsklasse über das `redeclare` Statement

Für die Verwendung der `redeclare` Funktionalität, müssen folgende Voraussetzungen erfüllt werden:

- Die neue Modellklasse (also in unserem Beispiel entweder `HeatingResistor` bzw. `SaturatingInductor`) müssen alle Parameter der ersetzten Klasse (hier `Resistor` bzw. `Inductor` beinhalten. Zusätzliche Parameter können beim Instanzieren von Objekten der neuen Klasse mit Werten versehen werden.
- Die neue Modellklasse muss mindestens die Schnittstellenvariablen der ersetzten Klasse besitzen.
- Variablennamen und Datentypen von Parametern und Schnittstellenvariablen müssen in neuer und ersetzter Klasse übereinstimmen.
- Variablen mit demselben Namen dürfen nach der Ersetzung der Klasse keine

²⁷Also das Modell von welchem geerbt wird.

vergrößerte Variabilität haben. Das bedeutet, dass wenn z.B. in der linearen Induktivität `L` als Parameter definiert ist, darf `L` in der `SaturatingInductor` nicht als veränderlicher Wert verwendet werden, sondern muss auch dort ein Parameter²⁸ sein.

Der letzte in der Aufzählung erwähnte Punkt macht die beiden hier aufgeführten Beispiele bei der Verwendung der Modelica Standard Library leider etwas umständlich. In beiden hier demonstrierten Fällen sind die im einfacheren Modell verwendeten Parameter im komplexeren Modell veränderliche Werte. Daher müssen für diese Anwendung händisch Modelle für `HeatingResistor` und `SaturatingInductor` definiert werden in denen `R` und `L` Parameter sind²⁹. Daher sind die oberhalb angeführten Beispiele auch nicht direkt funktionsfähig.

Der große Vorteil dieser Vorgangsweise, also dem Austauschen von einzelnen Elementen der Grundmodelle durch z.B. komplexere Teile liegt darin, dass bei einer Modifikation des Grundmodells sich diese Änderungen automatisch auch auf die davon abgeleiteten Varianten auswirken. Dies klingt ähnlich der Vererbung, kann aber in anderen Fällen als die Vererbung gezielt eingesetzt werden und bringt dort entscheidende Vorteile mit sich.

Ein weiterer für diese Fähigkeit von Modelica sehr passender Fall ist die Modellierung von Fluid-Strömungen. Hier ergibt sich der Umstand, dass verschiedene Modelle, wie etwa das von Pumpen, nur für einen gewissen Typ von Flüssigkeit zulässig sind. Es wird durch die hier besprochenen Fähigkeiten möglich, ein generell gültiges Modell einer solchen Pumpe zu erstellen [Sch09b].

6.7.13. Andere Programmiersprachen: C, Fortran

Da sich Modelica Funktionen und die Funktionen aus Programmiersprachen sehr ähnlich sind und Modelica im Normalfall vor der endgültigen Kompilierung in C übersetzt wird, ist es relativ einfach C und Fortran Funktionen einzubinden. In einigen Dokumenten werden auch C++ und JAVA erwähnt, diese sind allerdings nicht Teil der Modelica Spezifikation. Zur Einbindung genügt es in Modelica die Inputs und Outputs der Funktionen zu definieren und über `external` und die Angabe der Programmiersprache sowie des Funktionsnamens und der Reihenfolge der an die Funktion übergebenen Parameter anzugeben. Dies wird im folgenden Beispiel illustriert. Darin wird eine Modelica Funktion definiert, welche den Vergleich zweier Strings über eine in C erstellte Funktion durchgeföhrt.

```
1 function compare "Compare two strings lexicographically"  
2   extends Modelica.Icons.Function;
```

²⁸Parameter bleiben über die Zeit gesehen immer konstant

²⁹Stand in der Modelica Standard Library 3.2

```

3  input String string1;
4  input String string2;
5  input Boolean caseSensitive=true "= false, if case of letters
    is ignored";
6  output Modelica.Utilities.Types.Compare result "Result of
    comparison";
7  external "C" result = ModelicaStrings_compare(string1, string2,
    caseSensitive);
8  end compare;

```

Code 6.48: In C implementierte Modelica-Funktion zum Vergleich zweier Strings

6.7.14. Annotations: `annotation`

Annotations³⁰ werden in Modelica verwendet, um Informationen zu speichern, welche nicht direkt mit der Funktionalität des Modells zusammenhängen. Diese Informationen umfassen etwa folgende Punkte.

- Graphische Informationen wie Position, Größe und Orientierung von Icons, Verlauf von Verbindungslinien, Aufbau von Icons aus einfachen Grundelementen
- Dokumentation in einem Subset von HTML, welche im Info-Bereich des Modells abrufbar ist.
- Versionierung
- Code Generierung
- Eckdaten der Simulation, wie Start- und Stopzeit
- Viele weitere teilweise auch toolspezifische oder undokumentierte Annotations

Unterhalb zeigt ein Modell ein übliches `connect` Statement. Dabei wird zuerst angegeben welche Schnittstellen verbunden werden. Wie die grafische Linie gezogen wird, ist danach in der `annotation` festgelegt. Diese Information ist für die Funktionalität des Modells nicht nötig, allerdings wird die Linie ohne diesen Zusatz nicht dargestellt, die Schnittstellen sind aber physikalisch verbunden. Bei der grafischen Definition von Objekten über Drag and Drop aus dem Komponentenbrowser werden diese Zusatzinformationen automatisch erzeugt. Dies ist auch der Grund warum die rein textuell definierten Beispiele keine grafische Darstellung haben, allerdings trotzdem funktionieren.

```

1  connect (a.x, b.x)
2  annotation(Line(points={{-25,30}, {10,30}, {10, -20}, {40,-20}}
    ));

```

³⁰Deutsch: Anmerkungen

Code 6.49: Annotation zur Definition der grafischen Eigenschaften einer Verbindung

Auch die Dokumentation welche im Info-Bereich des Modells einsehbar ist, wird direkt im Modelica Code über die `Documentation` Annotation hinterlegt. Dabei wird ein reduziertes HTML Set verwendet. Bilder werden extern in Dateien abgelegt.

```
1 documentation_annotation:  
2   annotation(" Documentation "(" "info" "=" STRING  
3               ["," "revisions" "=" STRING ] )" ") "
```

Code 6.50: Annotation in welcher die Dokumentation gespeichert wird, welche im Info Bereich des Modells angezeigt wird.

Üblicherweise stellen die Modelica-Umgebungen eine Möglichkeit zur Verfügung, um annotations auszublenden, da sie im vergleich zum eigentlich Modell oft sehr viel Platz einnehmen und oft nicht von Hand erstellt werden. Auf diesem Wege kann die Kernfunktionalität des Modells einfach und übersichtlich dargestellt werden.

6.7.15. Skripte in Modelica

Ziel des *Scripting* in Modelica ist es, komplexe Simulationsläufe z.B. mit variierenden Parametern oder unterschiedlicher Belastung ohne Eingriff durch den User zu ermöglichen. Dazu kann man über *Scripting* Libraries laden, Parameter setzen, Simulationsparameter ändern, Simulationen starten und Variablen speichern, weiterverarbeiten oder plotten. Grundsätzlich sind alle Modelica Funktionen und auch externe Funktionen verfügbar.

Es gibt zwei Möglichkeiten Skripte umzusetzen. Die erste sind die bereits auf Seite 183 in Abschnitt „Funktionen: `function`“ besprochenen Funktionen, die andere sind `*.mos` Dateien. Im folgenden soll kurz auf die Unterschiede zwischen den beiden eingegangen werden.

Funktionen: `function`

Funktionen haben einen nur für sie reservierten Speicherbereich. Sie greifen nicht auf den Workspace von Dymola zu. Dadurch und durch ihren prozeduralen Aufbau (`algorithm` Abschnitt) können sie gut verwendet werden, um Modelle z.B. in Schleifen aufzurufen und dabei z.B. Parameter zu variieren. Eine Variation davon wurde in [Sch09a] verwendet, um ein nichtlineares Motorradmodell bei verschiedenen Geschwindigkeiten zu linearisieren und einen stückweise linearen Regler dafür auszulegen. Funktionen werden üblicherweise als Teil eines Package (also in einer `.mo` Datei) abgelegt.

Skriptdateien * .mos

Eine Skript Datei kann verwendet werden, um eine Reihe von Befehlen oder Funktionsaufrufen automatisch hintereinander abzuarbeiten und wird in einer .mos Datei abgelegt³¹. Zusätzlich kann eine Skript Datei auch eine andere Skriptdatei aufrufen. Sie bestehen grundsätzlich aus Befehlen welche auch in der Kommando-Zeile eingegeben werden können.

Eine einfache Variante sich ein Modelica Skript zu erstellen, ist folgende. Viele der Modelica Simulationsumgebungen haben eine Kommandozeile, in welcher Befehle direkt eingegeben werden können und ein Fenster, in welchem die Rückgabewerte von ausgeführten Funktionen angezeigt werden. Hier können Befehle per direkter Eingabe ausgeführt, kopiert und direkt in ein Modelica Skript eingefügt werden. Zusätzliche Funktionen wie z.B. Schleifen sind bezüglich der Syntax meist gleich wie die Versionen in den `algorithm` Abschnitten. Gute Dokumentationen zu diesen Skript Files sind am wahrscheinlichsten in der Dokumentation des Simulationsprogramms selbst zu finden. Dymola hat hier eine relativ ausführliche Dokumentation.

6.8. Strukturvariabilität

Eine große Herausforderung in der aktuellen Forschung stellen strukturvariable Systeme dar. Die grundlegende Eigenschaft solcher Systeme ist es, dass sich die Anzahl der Zustandsvariablen während der Simulation und auch abhängig von anderen Zuständen ändern kann. Dadurch verändern sich die Dimensionen der zu lösenden Matrixgleichungen und diese müssen unter Umständen auch neu aufgebaut werden. Diese Aufgabe ist äußerst schwierig zu bewältigen und daher noch Gegenstand aktueller Forschung. Weitere Informationen und ein möglicher Ansatz zur Lösung dieser Herausforderungen sind in [Zim10] beschrieben.

Ein sehr einfache Lösung dieses Problems ist folgende. Es kann für beide Modelle ein eigenständiges (Modelica) Modell erstellt werden. Diese können separat kompiliert werden und in verschiedene Ordner auf der Festplatte gespeichert werden. Es muss anschließend definiert werden unter welchen Umständen welches Modell gültig ist. Dazu werden in beiden Modellen Abbruchbedingungen definiert³². Wird ein Modell ungültig, kann das andere Modell - das nun gültig sein muss - mit den letzten Zuständen des terminierten Modells initialisiert werden. Nun kann das zweite Modell so lange simuliert werden, bis dieses ungültig wird. Zu diesem Zeitpunkt muss wiederum auf ein anderes gültiges Modell umgeschaltet werden. Dieser Vorgang wird mit so vielen Modellen wie nötig so lange fortgesetzt, bis die Gesamtsimulationszeit erreicht ist. Die Umschaltung zwischen den beiden Modellen kann dabei über verschiedenste Methoden

³¹Ähnlich einer Batch Datei, also in einer reinen Textdatei.

³²In Modelica kann das über ein `terminate` erreicht werden.

erfolgen. Denkbar sind Modelica Funktionen oder Skripte, aber auch MATLAB oder eine generelle Programmiersprache wie C oder Python. Grundsätzlich muss nicht mehr getan werden, als die von Dymola erstellten `dymosim.exe` Dateien mit den richtigen Initialwerten aufzurufen, und die von den einzelnen Simulationen gelieferten Ergebnisse zu einem Ergebnisfile zusammenzusetzen.

An einer Sprachdefinitionen, Übersetzung- und Lösungsalgorithmen, welche dieses Thema komfortabler lösen können wird zur Zeit gearbeitet. Modelica 4.0 wird in wenigen Jahren erscheinen und soll zumindest teilweise diese strukturvariablen Systeme simulieren können. Wenn es dazu Neuigkeiten gibt, wird dieser Abschnitt erweitert.

6.9. Symbolische Vorverarbeitung

In allen bisherigen Ausführungen wurde davon ausgegangen, dass das zu lösende Gleichungssystem in Form von ODEs, sprich in der Zustandsraumdarstellung vorliegt. Wie wir allerdings in diesem Kapitel festgestellt haben, ist es in der komponenten- bzw. objektorientierten Modellierung unumgänglich Komponenten in Form von Gleichungen zu beschreiben.

Man spricht in diesem Zusammenhang auch von expliziten Gleichungen, welche bereits so formuliert sind, dass sie als Zuweisung in einer Programmiersprache verwendbar sind, oder von impliziten Gleichungen welche in der Form $f(\dot{x}, x, u, t) = 0$ vorliegen (DAE). Nur auf einer Basis von DAEs ist die objektorientierte Modellierung möglich.

Es bieten sich nun grundsätzlich zwei Möglichkeiten an mit diesen DAEs umzugehen. Man kann Lösungsalgorithmen (Solver) verwenden, welche DAEs direkt lösen, oder die DAEs symbolisch in ODEs umformen und diese dann einem ODE Solver übergeben. Zur Zeit ist es eher üblich die aus den objektorientierten Modellen entstehenden DAEs in ODEs umzuformen und diese dann zu lösen. Dies liegt unter anderem daran, dass ODE Solver ein sehr gut erarbeitetes Feld der Wissenschaft darstellen und bereits sehr viel Forschung in diese Richtung erfolgt ist (siehe [CK06]). In den folgenden Abschnitten wird daher vor allem die Umformung von DAEs auf ODEs erläutert. Dieser Vorgang wird als symbolische Vorverarbeitung bezeichnet und bringt einige Schwierigkeiten mit sich. Sie ist gegenwärtig noch Forschungsgegenstand und wegen der großen Unterschiede in der Effizienz in der Lösung der erzeugten Systeme auch eines der Unterscheidungsmerkmale der verschiedenen objektorientierten Modellierungstools.

6.9.1. Sortieren von Gleichungen: „Index 0“ Systeme

Das Sortieren der Gleichungen hat die Aufgabe aus den Gleichungen des objektorientierten Modells Zuweisungen zu machen, damit diese in einer für den Computer „verständlichen“ Sprache (üblicherweise C) als Zuweisungen formuliert werden können.

Dies könnte grundsätzlich auch manuell durchgeführt werden, wie es in [Kapitel 3](#) aufgezeigt wurde. Diese Prozedur ist allerdings aufwändig und fehleranfällig.

Bei der Umformung von Gleichungen auf Zuweisung spricht man von der horizontalen Sortierung, weil dabei Variablen über symbolische Umformung von der linken auf die rechte Seite des Gleichheitszeichens und umgekehrt gebracht werden. Auf diese Weise wird aus der Gleichung eine Zuweisung. Jetzt könnten diese Zuweisungen verwendet werden, um sie z.B. in Simulink zu implementieren, da es sich bei Simulink nicht um eine prozedurale Sprache handelt und somit die Reihenfolge der Zuweisungen keine Rolle spielt.

Damit diese Zuweisungen in einer prozeduralen Programmiersprache funktionsfähig implementiert werden können, müssen die Zuweisungen noch vertikal sortiert werden. Dabei wird dafür gesorgt werden, dass keine Variable verwendet wird, bevor sie einen Wert zugewiesen bekommen hat. Nun kann das Gleichungssystem in einer prozeduralen Programmiersprache (C, C++, MATLAB M-Code...) implementiert werden. Es gibt auch Algorithmen, welche die horizontale und vertikale Sortierung in einem einzelnen Schritt vornehmen, ein solcher Algorithmus wird in diesem Abschnitt vorgestellt. Zusätzlich kann gesagt werden, dass diese Algorithmen sehr effizient sind und Gleichungssysteme mit mehreren 10.000 Gleichungen in wenigen Sekunden sortieren können.

Im Folgenden sollen die hier bereits theoretisch vorgestellten Vorgänge an einem Beispiel klar erläutert werden. Betrachten wird wiederum das Beispiel welches schon in [Kapitel 3](#) Verwendung fand und in [Abbildung 6.16](#) erneut dargestellt ist, ergeben sich folgende Gleichungen für eine komplette Beschreibung des Systems.

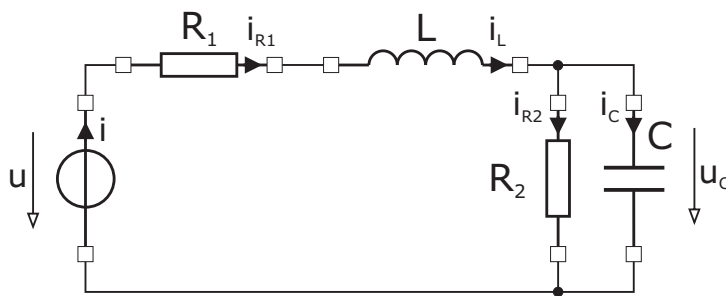


Abb. 6.16.: Passive elektronische Schaltung

$$u = 30V \tag{6.24}$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \tag{6.25}$$

$$u_L - L \cdot \frac{di_L}{dt} = 0 \tag{6.26}$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \tag{6.27}$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \tag{6.28}$$

$$i - i_{R_1} = 0 \tag{6.29}$$

$$i_{R_1} - i_L = 0 \tag{6.30}$$

$$u_{R_2} - u_C = 0 \tag{6.31}$$

$$u - u_{R_1} - u_L - u_C = 0 \tag{6.32}$$

$$i_L - i_{R_2} - i_C = 0 \tag{6.33}$$

Ein Lösungsalgorithmus für explizite Gleichungssysteme (also ein ODE Solver) kann mit dieser Formulierung des Systems, obwohl diese absolut korrekt ist, nichts anfangen. Wir müssen dafür sorgen, dass diese Gleichungen horizontal und vertikal sortiert werden. Dafür können zwei einfache Regeln angewendet werden³³:

- Kommt eine Variable nur in einer Gleichung vor, muss diese Gleichung verwendet werden, um diese Variable zu berechnen (z.B. Gleichung 6.26).
- Kommt in einer Gleichung nur eine unbekannte Variable vor muss die Variable aus dieser Gleichung berechnet werden (z.B. Gleichung 6.33).

Dabei ist es noch wichtig zu beachten, dass Zustandsvariablen im aktuellen Simulationsschritt bekannt sind. Der Grund dafür ist relativ einfach zu erklären. Im ersten Simulationsschritt zum Zeitpunkt $t = 0$ müssen die Zustandsvariablen initialisiert werden. Der Initialwert beschreibt den Verlauf der Eingangsgröße des Integrators in der Vergangenheit bis zum aktuellen Zeitpunkt. Für die nächsten Schritte sorgt der Lösungsalgorithmus dafür, dass die Zustandsvariablen bekannt sind, da dieser grundsätzlich die Zustandsgrößen des nächsten Simulationsschrittes errechnet. Mathematisch beschrieben haben die Lösungsalgorithmen immer die Aufgabe $x(t+h) = f(x(t), u(t))$ zu berechnen. Sie stellen also aus dem Simulationsergebnis $x(t)$ und $\dot{x}(t)$ die zukünftigen Werte $x(t+h)$ zur Verfügung. Somit sind diese im nächsten Schleifendurchlauf, welcher $x(t+2h)$ aus $x(t+h)$ berechnet, bekannt.

Im ersten Schritt sind also Eingänge, die vom Anwender vorgegeben werden müssen und Zustandsvariablen bekannt. Diese bekannten Variablen werden doppelt unterstrichen

³³Das Ergebnis dieser Vorgehensweise wurde zwar in Kapitel 3 gezeigt, der Weg dorthin aber nicht erläutert. Oft wird diese Umformung manuell und über langwieriges Versuchen erstellt. Hier soll eine algorithmische Methode zur Sortierung der Gleichungen erläutert werden.

und somit als bekannt markiert. Die Variablen nach welchen eine Gleichung gelöst werden soll, werden mit einer Umrahmung gekennzeichnet. Parameter werden im folgenden der Übersichtlichkeit halber in grau dargestellt.

Wendet man nun die beiden Regeln an, ergibt nach dem ersten Schritt das folgende Bild.

$$u = \underline{30V} \quad (6.34)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.35)$$

$$u_L - L \cdot \frac{di_L}{dt} = 0 \quad (6.36)$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \quad (6.37)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad (6.38)$$

$$\underline{i} = i_{R_1} \quad (6.39)$$

$$i_{R_1} - \underline{i_L} = 0 \quad (6.40)$$

$$u_{R_2} - \underline{u_C} = 0 \quad (6.41)$$

$$u - u_{R_1} - u_L - \underline{u_C} = 0 \quad (6.42)$$

$$i_L - i_{R_2} - i_C = 0 \quad (6.43)$$

Dabei ist die Klemmenspannung 30V offensichtlich bekannt da es sich bei u um den Eingang in die Simulation handelt. Die Zustandsvariablen i_L und u_C sind bekannt weil sie initialisiert werden müssen. Die beiden Ableitungen di_L/dt und du_C/dt sind umrahmt weil sie nur in diesen Gleichungen vorkommen.

Somit haben wir begonnen die horizontale Sortierung vorzunehmen. Dabei ist es aber auch wichtig, die vertikale Sortierung festzulegen, um einen prozedural ausführbaren Code zu erhalten. Um dies zu ermöglichen gibt es noch zwei einfache Regeln:

- Wird die Kausalität einer Gleichung dadurch festgelegt, dass die Variable die einzige Unbekannte in einer Gleichung ist, wird diese Gleichung mit einer aufsteigenden Nummer beginnend bei eins versehen.
- Wird die Kausalität durch den Umstand festgelegt, dass die Unbekannte nur in dieser Gleichung vorkommt, wird diese von der Anzahl der Gleichungen startend mit einer sinkenden Nummer markiert.

Werden mehrere Gleichungen in einem Schritt kausalisiert, ist es unerheblich in welcher Reihenfolge die in einem Schritt kausalisierten Gleichungen nummeriert werden.

Daher ergibt sich für die bisher makierten Unbekannten folgende Auswertungsreihenfolge.

$$u = \underline{\underline{30V}} \quad \rightarrow \textcircled{1} \quad (6.44)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.45)$$

$$u_L - L \cdot \frac{di_L}{dt} = 0 \quad \rightarrow \textcircled{10} \quad (6.46)$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \quad (6.47)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad \rightarrow \textcircled{9} \quad (6.48)$$

$$\boxed{i} = i_{R_1} \quad \rightarrow \textcircled{8} \quad (6.49)$$

$$i_{R_1} - \underline{\underline{i_L}} = 0 \quad (6.50)$$

$$u_{R_2} - \underline{\underline{u_C}} = 0 \quad (6.51)$$

$$u - u_{R_1} - u_L - \underline{\underline{u_C}} = 0 \quad (6.52)$$

$$i_L - i_{R_2} - i_C = 0 \quad (6.53)$$

Die beiden Regeln zur Kausalisierung werden zusammen mit den Regeln für die Nummerierung angewand und wir erhalten den folgenden Schritt.

$$\boxed{u} = \underline{\underline{30V}} \quad \rightarrow \textcircled{1} \quad (6.54)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.55)$$

$$u_L - L \cdot \frac{di_L}{dt} = 0 \quad \rightarrow \textcircled{10} \quad (6.56)$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \quad (6.57)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad \rightarrow \textcircled{9} \quad (6.58)$$

$$\boxed{i} = \underline{\underline{i_{R_1}}} \quad \rightarrow \textcircled{8} \quad (6.59)$$

$$\boxed{i_{R_1}} - \underline{\underline{i_L}} = 0 \quad \rightarrow \textcircled{2} \quad (6.60)$$

$$\boxed{u_{R_2}} - \underline{\underline{u_C}} = 0 \quad \rightarrow \textcircled{3} \quad (6.61)$$

$$u - u_{R_1} - u_L - \underline{\underline{u_C}} = 0 \quad (6.62)$$

$$i_L - i_{R_2} - i_C = 0 \quad (6.63)$$

Im nächsten Schritt können die nun bestimmten Variablen in den anderen Gleichungen als bekannt angenommen werden.

$$\boxed{u} = \underline{30V} \quad \rightarrow \textcircled{1} \quad (6.64)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.65)$$

$$u_L - L \cdot \frac{di_L}{dt} = 0 \quad \rightarrow \textcircled{10} \quad (6.66)$$

$$\underline{u_{R_2}} - i_{R_2} \cdot R_2 = 0 \quad (6.67)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad \rightarrow \textcircled{9} \quad (6.68)$$

$$\boxed{i} = i_{R_1} \quad \rightarrow \textcircled{8} \quad (6.69)$$

$$\boxed{i_{R_1}} - \underline{i_L} = 0 \quad \rightarrow \textcircled{2} \quad (6.70)$$

$$\underline{u_{R_2}} - \underline{u_C} = 0 \quad \rightarrow \textcircled{3} \quad (6.71)$$

$$\underline{u} - u_{R_1} - u_L - \underline{u_C} = 0 \quad (6.72)$$

$$i_L - i_{R_2} - i_C = 0 \quad (6.73)$$

Dadurch können wieder neue Variablen berechnet werden.

$$\boxed{u} = \underline{30V} \quad \rightarrow \textcircled{1} \quad (6.74)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.75)$$

$$u_L - L \cdot \frac{di_L}{dt} = 0 \quad \rightarrow \textcircled{10} \quad (6.76)$$

$$\underline{u_{R_2}} - \boxed{i_{R_2}} \cdot R_2 = 0 \quad \rightarrow \textcircled{4} \quad (6.77)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad \rightarrow \textcircled{9} \quad (6.78)$$

$$\boxed{i} = \underline{i_{R_1}} \quad \rightarrow \textcircled{8} \quad (6.79)$$

$$\boxed{i_{R_1}} - \underline{i_L} = 0 \quad \rightarrow \textcircled{2} \quad (6.80)$$

$$\underline{u_{R_2}} - \underline{u_C} = 0 \quad \rightarrow \textcircled{3} \quad (6.81)$$

$$\underline{u} - u_{R_1} - u_L - \underline{u_C} = 0 \quad (6.82)$$

$$i_L - i_{R_2} - i_C = 0 \quad (6.83)$$

Im nächsten Schritt ergibt sich folgendes Bild.

$$\boxed{u} = \underline{30V} \quad \rightarrow \textcircled{1} \quad (6.84)$$

$$\boxed{u_{R_1}} - \underline{i_{R_1}} \cdot R_1 = 0 \quad \rightarrow \textcircled{5} \quad (6.85)$$

$$\underline{u_L} - L \cdot \frac{di_L}{dt} = 0 \quad \rightarrow \textcircled{10} \quad (6.86)$$

$$\underline{u_{R_2}} - \boxed{i_{R_2}} \cdot R_2 = 0 \quad \rightarrow \textcircled{4} \quad (6.87)$$

$$\underline{i_C} - C \cdot \frac{du_C}{dt} = 0 \quad \rightarrow \textcircled{9} \quad (6.88)$$

$$\boxed{i} = \underline{i_{R_1}} \quad \rightarrow \textcircled{8} \quad (6.89)$$

$$\boxed{i_{R_1}} - \underline{i_L} = 0 \quad \rightarrow \textcircled{2} \quad (6.90)$$

$$\boxed{u_{R_2}} - \underline{u_C} = 0 \quad \rightarrow \textcircled{3} \quad (6.91)$$

$$\underline{u} - \underline{u_{R_1}} - \boxed{u_L} - \underline{u_C} = 0 \quad \rightarrow \textcircled{6} \quad (6.92)$$

$$\underline{i_L} - \underline{i_{R_2}} - \boxed{i_C} = 0 \quad \rightarrow \textcircled{7} \quad (6.93)$$

Jetzt ergeben sich noch zwei Aufgaben. Einmal müssen die Gleichungen vertikal umsortiert werden, was der vertikalen Sortierung entspricht. Etwas komplizierter ist die horizontale Sortierung, weil dabei eine symbolische Umformung der Gleichungen nötig wird. Auch diese Aufgabe wird von der symbolischen Vorverarbeitung der Modellierungsumgebung übernommen.

Das Gleichungssystem ergibt sich also zu

$$u := 30V \quad \rightarrow \quad \textcircled{1} \quad (6.94)$$

$$i_{R_1} := i_L \quad \rightarrow \quad \textcircled{2} \quad (6.95)$$

$$u_{R_2} := u_C \quad \rightarrow \quad \textcircled{3} \quad (6.96)$$

$$i_{R_2} := \frac{u_{R_2}}{R_2} \quad \rightarrow \quad \textcircled{4} \quad (6.97)$$

$$u_{R_1} := i_{R_1} \cdot R_1 \quad \rightarrow \quad \textcircled{5} \quad (6.98)$$

$$u_L := u - u_{R_1} - u_C \quad \rightarrow \quad \textcircled{6} \quad (6.99)$$

$$i_C := i_L - i_{R_2} \quad \rightarrow \quad \textcircled{7} \quad (6.100)$$

$$i := i_{R_1} \quad \rightarrow \quad \textcircled{8} \quad (6.101)$$

$$\frac{du_C}{dt} := \frac{i_C}{C} \quad \rightarrow \quad \textcircled{9} \quad (6.102)$$

$$\frac{di_L}{dt} := \frac{u_L}{L} \quad \rightarrow \quad \textcircled{10} \quad (6.103)$$

Somit haben wir das Gleichungssystem nun auf einem sehr algorithmischen Weg direkt in einer Programmiersprache implementierbares Gleichungssystem gefunden. Diese Vorgehensweise wird Tarjan-Algorithmus genannt und für den Menschen auch etwas verträglicher als ein Structure Digraph dargestellt werden. Auf diese grafische Art der Formulierung des Algorithmus wird allerdings in diesem Skript nicht weiter eingegangen, da diese Aufgaben heute in allen praktisch relevanten Fällen von Computern übernommen werden.

Über das Einsetzen der Gleichungen in die letzten beiden Gleichungen könnte daraus auch ein Zustandsraum erzeugt werden. Dies ist allerdings nicht in allen Fällen effizient und wird daher auch nicht immer getan.

Wichtig ist es zu erkennen, dass jetzt keine Variable mehr verwendet wird, bevor dieser ein sinnvoller Wert zugewiesen wurde. Das Ergebnis der obenstehenden Gleichungen entspricht exakt dem, der in [Abschnitt 3.4](#) auf Seite 42 dargestellten Gleichungen. Die Reihenfolge der vertikalen Sortierung ist im Vergleich zu [Abschnitt 3.4](#) teilweise unterschiedlich. Dies kommt daher, dass in einem Schritt oft mehrere Gleichungen vertikal sortiert werden können und sich somit ein gewisser Spielraum für diese Sortierung ergibt, der allerdings für die Berechnung keine Rolle spielt.

Anschließend kann dieses Gleichungssystem beispielsweise über die in [Abschnitt 3.5](#) vorgestellten Verfahren implementiert werden.

6.9.2. Algebraische Schleifen: „Index 1“ Systeme

Wir versuchen noch einmal den vorher erfolgreich angewandten Algorithmus an einem anderen, ähnlich einfachen Beispiel anzuwenden. Als Beispiel wird die in [Abbildung 6.17](#) dargestellte und bereits auf Seite 106 in [Unterabschnitt 5.7.2](#) verwendete Schaltung verwendet.

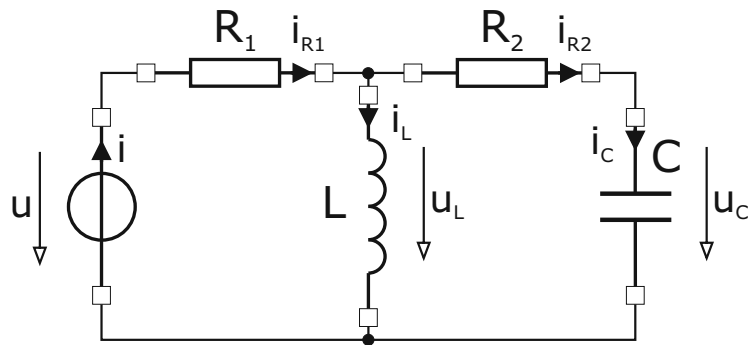


Abb. 6.17.: Elektrische Schaltung welche eine algebraische Schleife aufweist.

Dabei ergeben sich folgende Modellgleichungen³⁴.

$$u = 30V \quad (6.104)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.105)$$

$$u_L - L \cdot \frac{di}{dt} = 0 \quad (6.106)$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \quad (6.107)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad (6.108)$$

$$i - i_{R_1} = 0 \quad (6.109)$$

$$i_C - i_{R_2} = 0 \quad (6.110)$$

$$i_{R_1} - i_L - i_{R_2} = 0 \quad (6.111)$$

$$u - u_{R_1} - u_L = 0 \quad (6.112)$$

$$u_L - u_{R_2} - u_C = 0 \quad (6.113)$$

Führt man jetzt wieder die im vorherigen Abschnitt eingeführte Vorgangsweise aus, kommt man bereits nach dem ersten Schritt ins Stocken, weil keine der beiden Regeln

³⁴Bei der Bondgraphendarstellung in [Unterabschnitt 5.7.2](#) ergaben sich nur acht Gleichungen weil die trivialen Gleichungen (6.109) und (6.110) nicht explizit aufgeführt sind, sondern durch eine Umbenennung der Variablen vorgenommen wurden.

mehr anschlägt. Jede Variable kommt mindesten in zwei Gleichungen vor und jede Gleichung hat mindestens zwei Unbekannte.

$$\boxed{u} = \underline{30V} \quad (6.114)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.115)$$

$$u_L - L \cdot \frac{di}{dt} = 0 \quad (6.116)$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \quad (6.117)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad (6.118)$$

$$i - i_{R_1} = 0 \quad (6.119)$$

$$i_C - i_{R_2} = 0 \quad (6.120)$$

$$i_{R_1} - \underline{i_L} - i_{R_2} = 0 \quad (6.121)$$

$$\underline{u} - u_{R_1} - u_L = 0 \quad (6.122)$$

$$u_L - u_{R_2} - \underline{u_C} = 0 \quad (6.123)$$

Wir sind also auf eine sogenannte algebraische Schleife gestoßen, wie sie in sehr vielen Systemen zu finden ist. Wie aus dem Beispiel gut zu erkennen ist, treten diese Schwierigkeiten auch schon bei relativ kompakten Systemen auf. Es gibt verschiedene Möglichkeiten diesem Problem zu entgegnen. Eine ist, eine Variable als bekannt anzunehmen und mit dem Algorithmus fortzufahren, als ob diese Variable wirklich bekannt wäre. Dabei werden die Variablen, welche auf dieser so-genannten Tearing Variable³⁵ beruhend berechnet werden als einzeln unterstrichen dargestellt. Die Zuweisungen welche sich daraus geben nicht von einem Kreis, sondern von einem Viereck umrandet. Dabei wird die als „residual equation“ (also die eigentlich nicht berechenbare Gleichung) mit einem doppelten Rahmen gekennzeichnet. In diesem Fall wählen wir u_{R_2} als bekannt und legen fest, dass diese aus Gleichung (6.127) errechnet wird. Daraus

³⁵Tearing Variable entspricht in Deutsch etwa einer „Aufbrech-Variablen“.

ergibt sich folgender Zusammenhang:

$$\boxed{u} = \underline{30V} \quad \rightarrow \textcircled{1} \quad (6.124)$$

$$u_{R_1} - i_{R_1} \cdot R_1 = 0 \quad (6.125)$$

$$u_L - L \cdot \frac{di}{dt} = 0 \quad \rightarrow \textcircled{10} \quad (6.126)$$

$$\boxed{u_{R_2}} - i_{R_2} \cdot R_2 = 0 \quad \rightarrow \boxed{2} \quad (6.127)$$

$$i_C - C \cdot \frac{du_C}{dt} = 0 \quad \rightarrow \textcircled{9} \quad (6.128)$$

$$i - i_{R_1} = 0 \quad (6.129)$$

$$i_C - i_{R_2} = 0 \quad (6.130)$$

$$i_{R_1} - \underline{i_L} - i_{R_2} = 0 \quad (6.131)$$

$$\underline{u} - u_{R_1} - u_L = 0 \quad (6.132)$$

$$u_L - \underline{u_{R_2}} - \underline{u_C} = 0 \quad (6.133)$$

Setzen wir die Sortierung nun fort, kann diese ohne weitere Schwierigkeiten abgeschlossen werden und es ergibt sich folgendes Bild.

$$\boxed{u} = \underline{30V} \quad \rightarrow \textcircled{1} \quad (6.134)$$

$$u_{R_1} - \boxed{i_{R_1}} \cdot R_1 = 0 \quad \rightarrow \boxed{6} \quad (6.135)$$

$$\underline{u_L} - L \cdot \frac{di}{dt} = 0 \quad \rightarrow \boxed{5} \quad (6.136)$$

$$\boxed{u_{R_2}} - i_{R_2} \cdot R_2 = 0 \quad \rightarrow \boxed{2} \quad (6.137)$$

$$\underline{i_C} - C \cdot \frac{du_C}{dt} = 0 \quad \rightarrow \textcircled{10} \quad (6.138)$$

$$i - \boxed{i_{R_1}} = 0 \quad \rightarrow \boxed{7} \quad (6.139)$$

$$\boxed{i_C} - i_{R_2} = 0 \quad \rightarrow \textcircled{9} \quad (6.140)$$

$$\underline{i_{R_1}} - \underline{i_L} - \boxed{i_{R_2}} = 0 \quad \rightarrow \boxed{8} \quad (6.141)$$

$$\underline{u} - \boxed{u_{R_1}} - u_L = 0 \quad \rightarrow \boxed{4} \quad (6.142)$$

$$\boxed{u_L} - \underline{u_{R_2}} - \underline{u_C} = 0 \quad \rightarrow \boxed{3} \quad (6.143)$$

Dies entspricht dem folgenden Satz an Zuweisungen.

$$u := 30V \quad \rightarrow \textcircled{1} \quad (6.144)$$

$$\boxed{u_{R_2}} := i_{R_2} \cdot R_2 \quad \rightarrow \textcircled{2} \quad (6.145)$$

$$u_L := u_{R_2} + u_C \quad \rightarrow \textcircled{3} \quad (6.146)$$

$$u_{R_1} := u - u_L \quad \rightarrow \textcircled{4} \quad (6.147)$$

$$\frac{di}{dt} := \frac{u_L}{L} \quad \rightarrow \textcircled{5} \quad (6.148)$$

$$i_{R_1} := \frac{u_{R_1}}{R_1} \quad \rightarrow \textcircled{6} \quad (6.149)$$

$$i_{R_1} := i \quad \rightarrow \textcircled{7} \quad (6.150)$$

$$i_{R_2} := i_{R_1} - i_L \quad \rightarrow \textcircled{8} \quad (6.151)$$

$$i_C := i_{R_2} \quad \rightarrow \textcircled{9} \quad (6.152)$$

$$\frac{du_C}{dt} := \frac{i_C}{C} \quad \rightarrow \textcircled{10} \quad (6.153)$$

Nun findet sich allerdings die Tearingvariable u_{R_2} bereits in der zweiten Gleichung wieder, was nicht anders zu erwarten war, da wir vor der Wahl dieser nur eine Gleichung kausalisieren konnten, welche am Beginn des Gleichungssystems steht. Es ist aber durch die Kausalisierung des Systems jetzt einfach einen gültigen Ausdruck aus den vorhandenen Zuweisungen zu finden. Genauer gesagt sind dazu diese nötig, welche im vorherigen Schritt mit einem Viereck gekennzeichnet wurden. Welches die Gleichungen $\boxed{2}$ bis $\boxed{8}$ sind³⁶. Setzt man diese in umgekehrter Reihenfolgen ineinander ein, ergibt sich Folgendes:

$$u_{R_2} = i_{R_2} \cdot R_2 \quad (6.154)$$

$$= (i - i_L) \cdot R_2 \quad (6.155)$$

$$= (i_{R_1} - i_L) \cdot R_2 \quad (6.156)$$

$$= \left(\frac{u_{R_1}}{R_1} - i_L \right) \cdot R_2 \quad (6.157)$$

$$= \left(\frac{u - u_L}{R_1} - i_L \right) \cdot R_2 \quad (6.158)$$

$$= \left(\frac{u - (u_{R_2} + u_C)}{R_1} - i_L \right) \cdot R_2 \quad (6.159)$$

Der letzte Ausdruck besteht nur noch aus in diesem Rechenschritt bekannten Größen: Eingänge, Zustandsvariablen, Parametern und der Tearing Variable selbst. Daher kann daraus nun die Tearing Variable u_{R_2} bestimmt werden.

³⁶Wobei nicht alle zwingend nötig sein müssen.

Eine einfache Umformung führt zu der Lösung:

$$u_{R_2} = \frac{R_2}{R_1 + R_2} \cdot (u - u_C - i_L \cdot R_1) \quad (6.160)$$

Woraus sich nun das folgende berechenbare Gleichungssystem ergibt.

$$u := 30V \quad (6.161)$$

$$u_{R_2} := \frac{R_2}{R_1 + R_2} \cdot (u - u_C - i_L \cdot R_1) \quad (6.162)$$

$$u_L := u_{R_2} + u_C \quad (6.163)$$

$$u_{R_1} := u - u_L \quad (6.164)$$

$$\frac{di}{dt} := \frac{u_L}{L} \quad (6.165)$$

$$i := \frac{u_{R_1}}{R_1} \quad (6.166)$$

$$i_{R_1} := i \quad (6.167)$$

$$i_{R_2} := i - i_L \quad (6.168)$$

$$i_C := i_{R_2} \quad (6.169)$$

$$\frac{du_C}{dt} := \frac{i_C}{C} \quad (6.170)$$

Für eine gute Verständlichkeit wurde hier auf ein relativ einfaches Beispiel zurückgegriffen. Bei größeren Systemen stellt sich schnell die Frage, welche der möglichen Variablen eine „gute“ Tearing Variable ist. Leider ist die analytische Lösung dieses Problems mit einem exponentiell wachsenden Aufwand mit steigender Variablenanzahl verbunden. Daher werden für die Lösung dieser Aufgabe Heuristiken³⁷ eingesetzt, welche Eigenschaften bewerten wie z.B. „Man wähle eine Tearingvariable mit der sich in weitere Folge möglichst viele Gleichungen kausalisieren lassen“, oder ähnliches. Dabei wird zwar nicht in allen Fällen die ideale, aber zumindest eine gute Tearing Variable gefunden. Die Qualität der Heuristiken ist ein entscheidendes Merkmal für die Performance der Modellierungsumgebung. Es können auch verschachtelte Schleifen auftreten bei denen mehrere Tearing Variablen gewählt werden müssen.

In diesem Abschnitt wurde das Tearing Verfahren in Kombination mit der Substitutionstechnik angewandt. Dieser Vorgang kann zu sehr großen und unhandlichen Gleichungen führen. Daher ist es in vielen Fällen auch sinnvoll, die Gleichung in welcher die als bekannt angenommene Variable bestimmt wird, an das Ende der Gleichungen zu schreiben, welche die algebraische Schleife bilden (die mit \boxed{x} gekennzeichneten) und über dieses Gleichungssystem eine Newtoniteration anzuwenden. Bei der Implementie-

³⁷Eine Heuristik beschreibt den Fall, wenn man mit unvollständigem Wissen und eingeschränkter Zeit zu einer guten aber möglicherweise nicht perfekten Lösung kommen muss.

rung in Algorithmen kann es unter Umständen sinnvoller sein, statt für die Variable einzusetzen eine Newton Iteration durchzuführen (was z.B. auch Dymola macht).

Noch einen Hinweis: Systeme mit mindestens einer algebraischen Schleife werden Index 1 Systeme genannt. Systeme welche sich direkt kausalisieren lassen, haben den Index 0.

6.9.3. Strukturelle Singularitäten: „höhere Index“ Systeme

An dem in [Abbildung 6.18](#) dargestellten Beispiel soll eine weitere Hürde für den Tarjan Algorithmus erläutert werden. Dabei handelt es sich um das einfachste denkbare Beispiel.

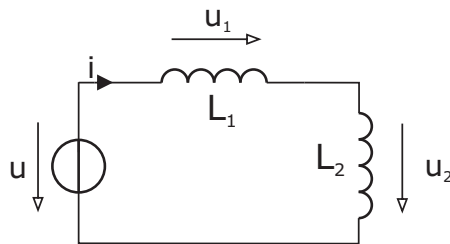


Abb. 6.18.: Spannungsquelle mit zwei in Serie geschalteten Induktivitäten

Die Gleichungen aus diesem Beispiel ergeben sich zu:

$$u = 30V \quad (6.171)$$

$$u_{L_1} - L_1 \cdot \frac{di_{L_1}}{dt} = 0 \quad (6.172)$$

$$u_{L_2} - L_2 \cdot \frac{di_{L_2}}{dt} = 0 \quad (6.173)$$

$$u - u_{L_1} - u_{L_2} = 0 \quad (6.174)$$

$$i - i_{L_1} = 0 \quad (6.175)$$

$$i_{L_1} - i_{L_2} = 0 \quad (6.176)$$

Beginnt man nun wieder mit der Prozedur des Tarjan Algorithmus ergibt sich schon

nach Beenden ersten Schrittes folgende Konstellation.

$$\boxed{u} = \underline{\underline{30V}} \quad (6.177)$$

$$u_{L_1} - L_1 \cdot \frac{di_{L_1}}{dt} = 0 \quad (6.178)$$

$$u_{L_2} - L_2 \cdot \frac{di_{L_2}}{dt} = 0 \quad (6.179)$$

$$\underline{u} - u_{L_1} - u_{L_2} = 0 \quad (6.180)$$

$$i - \underline{\underline{i_{L_1}}} = 0 \quad (6.181)$$

$$\underline{\underline{i_{L_1}}} - \underline{\underline{i_{L_2}}} = 0 \quad (6.182)$$

Dabei sticht ins Auge, dass in Gleichung (6.182) keine Variable berechnet werden kann, da alle schon bekannt sind. In diesem Fall ist diese Zwangsbedingung (engl. constraint equation) dadurch gegeben, dass durch die Induktivitäten gezwungenermaßen derselbe Strom fließen muss. Die Probleme ergeben sich in den Gleichungen daraus, dass beide Ströme Zustandsvariablen sind und daher bekannt sind.

Grundsätzlich wäre eine differenzielle Darstellung einer der beiden Gleichungen für die Induktivität möglich. Dabei würde die Initialisierung des Integrators entfallen und es würde sich keine Zwangsbedingung ergeben. Problematisch dabei ist, dass eine numerischen Differenziation hochgradig instabil ist. Wird numerisch mittels einer impliziten Gleichung differenziert, ist diese grundsätzlich gleich stabil wie ein Integrator, allerdings wird in diesem Fall wieder eine Iteration nötig, was die Berechnung im Vergleich zu der folgenden Lösung ineffizient macht. Daher wird dieser Lösungsansatz nicht weiter verfolgt.

In [CK06] wird der ursprüngliche Pantelides Algorithmus erweitert und somit wird die Strukturelle Singularität wie folgt gelöst. Dieser Weg wird z.B. auch von Dymola verwendet, ist effizient und wird daher im Folgenden präsentiert. Die Zwangsbedingung wird abgeleitet und zum ursprünglichen Satz von Differenzialgleichungen hinzu-

gefügt.

$$\boxed{u} = \underline{\underline{30V}} \quad (6.183)$$

$$u_{L_1} - L_1 \cdot \frac{di_{L_1}}{dt} = 0 \quad (6.184)$$

$$u_{L_2} - L_2 \cdot \frac{di_{L_2}}{dt} = 0 \quad (6.185)$$

$$\underline{u} - u_{L_1} - u_{L_2} = 0 \quad (6.186)$$

$$i - \underline{\underline{i_{L_1}}} = 0 \quad (6.187)$$

$$\underline{\underline{i_{L_1}}} - \underline{\underline{i_{L_2}}} = 0 \quad (6.188)$$

$$\frac{di_{L_1}}{dt} - \frac{di_{L_2}}{dt} = 0 \quad (6.189)$$

Dadurch ergibt sich eine Gleichung zu viel im Vergleich zu der Anzahl der Unbekannten. Um dies wieder auszugleichen wird ein Integrator entfernt. Das bedeutet, dass z.B. sowohl i_2 wie auch di_2/dt als unbekannt angenommen werden. Der bekannte Zusammenhang zwischen den beiden Variablen über einen Integrator wird also „verschwiegen“. Die Kennzeichnung eines solchen eliminierten Integrators erfolge über die Entfernung des $/dt$, so soll darauf hingewiesen werden, dass es sich hierbei um eine „normale“ algebraische Variable handelt. Dadurch ergibt sich auch gegenüber dem originalen Pantelides Algorithmus der Vorteil, dass dieses System erster Ordnung auch wirklich nur einen Integrator enthält, während beim originalen Algorithmus zwei Integratoren nötig gewesen wären. Der wichtigste Teil dieser Vorgangsweise ist, dass durch die Eliminierung des Integrators auch eine der beiden Initialisierungen entfällt. Daher entfällt die Zwangsbedingung und wird zu einer sortierbaren Gleichung.

$$\boxed{u} = \underline{\underline{30V}} \quad (6.190)$$

$$u_{L_1} - L_1 \cdot \frac{di_{L_1}}{dt} = 0 \quad (6.191)$$

$$u_{L_2} - L_2 \cdot \frac{di_{L_2}}{dt} = 0 \quad (6.192)$$

$$\underline{u} - u_{L_1} - u_{L_2} = 0 \quad (6.193)$$

$$i - \underline{\underline{i_{L_1}}} = 0 \quad (6.194)$$

$$\underline{\underline{i_{L_1}}} - i_{L_2} = 0 \quad (6.195)$$

$$\frac{di_{L_1}}{dt} - di_{L_2} = 0 \quad (6.196)$$

Wichtig zu erkennen ist, dass das Verhalten des Systems dadurch nicht verändert wird, oder die Berechnung ungenauer ist. Dieser Zusammenhang soll durch [Abbildung 6.19](#) noch einmal dargestellt werden.

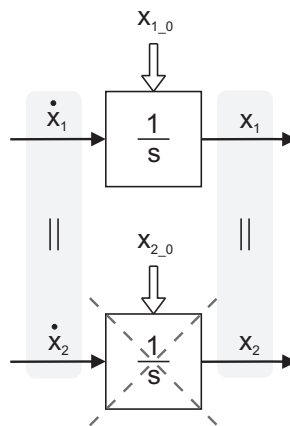


Abb. 6.19.: Zusammenhänge mit einem eliminierten Integrator

Dieser Fall ist absichtlich einfach gewählt. Kommen in der Zwangsbedingung zusätzliche Variablen vor, werden diese durch das Anwenden des modifizierten Pantelides Algorithmus ebenfalls differenziert und werden zu neuen Unbekannten. Daher müssen auch die Gleichungen mit welchen diese Variablen berechnet werden differenziert werden, um das Gleichgewicht zwischen Gleichungen und Unbekannten beizubehalten. Daher müssen u.U. viele Variablen differenziert werden. Eine zwingende Voraussetzung dafür, dass es sich um eine Zwangsbedingung handeln kann ist, dass alle Variablen in dieser Gleichung bekannt sind. Variablen können aus zwei Gründen bekannt sein. Einerseits kann es sich um Zustandsvariablen handeln, andererseits können sie aus einer anderen Gleichung berechnet werden. Für den Fall, dass es sich um Zustandsvariablen handelt, können wenn nötig Integratoren eliminiert werden. Wenn die Variablen aus einer anderen Gleichung errechnet wurden, und diese Variable wird abgeleitet sind diese Ableitungen im Normalfall neue Variablen. Allerdings ist klar aus welcher Gleichung die ursprüngliche Variable errechnet wurde, daher können diese Gleichungen abgeleitet werden um für die neue Variable wieder eine Gleichung zu erhalten³⁸. Dies muss so lange fortgesetzt werden, bis differenzierbare Ausdrücke entstehen.

Systeme die eine strukturelle Singularität enthalten sind Systeme höheren Indexes (higher index Systems oder Index ≥ 2). Wird der hier kurz beschriebene Algorithmus auf ein System höherer Ordnung angewandt reduziert sich der Index um eins. Dies ist der übliche Fall, es entsteht aus einem System mit einer strukturellen Singularität im Normalfall eines mit einer algebraischen Schleife. Ab und zu kommt es auch vor, dass der Index 1 übersprungen wird und das System nach dem Lösen der strukturellen Singularität direkt kausalisierbar ist. Es sind vor allem in der 3D Mechanik Systeme mit einem Index größer 2 möglich. Dabei muss der Pantelides Algorithmus mehrmals angewandt

³⁸Man darf dabei nicht vergessen, dass das Gleichgewicht zwischen Gleichungen und Unbekannten gewahrt werden muss.

werden oder es ist sogar ein händischer Eingriff über so genannte „cut joints“ nötig.

6.9.4. Eine Alternative: Direkte Lösung von impliziten Gleichungssystemen

Neben der symbolischen Vorverarbeitung, welche ein System von DAEs zu einem gleichwertigen von ODEs macht, ist es auch möglich die noch implizit formulierten Gleichungen direkt zu lösen. Diese liegen dann in der Form

$$f(\dot{x}, x, u, t) = 0 \quad (6.197)$$

vor. Verwendet man nun einen geeigneten Solver wie z.B. die Backward Difference Formulae

$$x_{t+h} = \frac{6}{11}h \cdot \dot{x}_{t+h} + \frac{18}{11}x_t - \frac{9}{11}x_{t-h} + \frac{2}{11}x_{t-2h} \quad (6.198)$$

lässt sich daraus die Ableitung zu

$$\dot{x}_{t+h} = \frac{1}{h} \left(\frac{11}{6}x_{t+h} - 3x_t + \frac{3}{2}x_{t-h} - \frac{1}{3}x_{t-2h} \right) \quad (6.199)$$

berechnen. Diese kann nun verwendet werden um die Ableitung in (6.197) zu ersetzen. Man erhält dadurch ein Gleichungssystem welches nur noch von dem aktuellen Zustandswert und von vergangenen Zustandswerten abhängt. Daher kann auf dieses direkt eine Newton Iteration angewandt werden.

$$F(x_{t+h}) = f \left(x_{t+h}, \frac{1}{h} \left(\frac{11}{6}x_{t+h} - 3x_t + \frac{3}{2}x_{t-h} - \frac{1}{3}x_{t-2h}, u_{t+h}, t \right) \right) = 0 \quad (6.200)$$

Somit lässt sich die neue Lösung x_{t+h} finden.

Wichtig ist es dabei, dass die Stabilitätseigenschaften der verwendeten Algorithmen durch diese Umformulierung nicht beeinflusst werden. Voraussetzung dafür ist, dass es sich bei den simulierten Systemen nicht um Systeme mit einem Index > 1 handelt. Da der Pantelides Algorithmus sehr gute Eigenschaften bezüglich des Rechenaufwandes hat, ist es in praktisch jedem Fall effizienter diesen anzuwenden, als dem DAE Solver dieses zu übergeben. Zusätzlich ist nicht gewährleistet, dass dieser mit dem System-Index > 1 zurecht kommen würde. Die Implementierung des DASSL Algorithmus welcher in Dymola verwendet wird bietet Schnittstellen um diesen als ODE oder als DAE Solver zu verwenden. Dymola verwendet dabei immer das ODE Interface und reduziert den Index der simulierten Systeme vor der Simulation auf null.

6.9.5. Zusammenfassung

Wieso hat die Zustandsraumdarstellung in der Modellbildung und Simulation so eine große Bedeutung? Die Antwort auf diese Frage ist historisch begründet. Da zu Beginn der Simulationstechnik ausschließlich explizite Lösungsverfahren (also ODE Solver) verwendet wurden, war für eine Simulation die Zustandsraumdarstellung zwingend nötig. Daraus ergab es sich, dass sich auf der einen Seite Personen mit der Erstellung der Zustandsraummodelle beschäftigten, die anderen kümmerten sich darum, diese numerisch zu simulieren. Mit der Zustandsraumdarstellung wurde hier eine sehr gute Schnittstelle zwischen diesen beiden Fachgebieten eingeführt.

Da für ODE Solver nur die Zustandsraumdarstellung verständlich ist, wurden die Algorithmen zur Überführung von DAEs in ODEs entwickelt. Dazu war es auch nötig, die Algorithmen zur Indexreduktion zu entwickeln. Eine Übersicht zu den hier vorgestellten Varianten ist in [Abbildung 6.20](#) dargestellt. Dabei gibt es noch andere Wege die Indexreduktion durchzuführen. Eine vollständigere Aufstellung dazu ist in [CK06] zu finden.

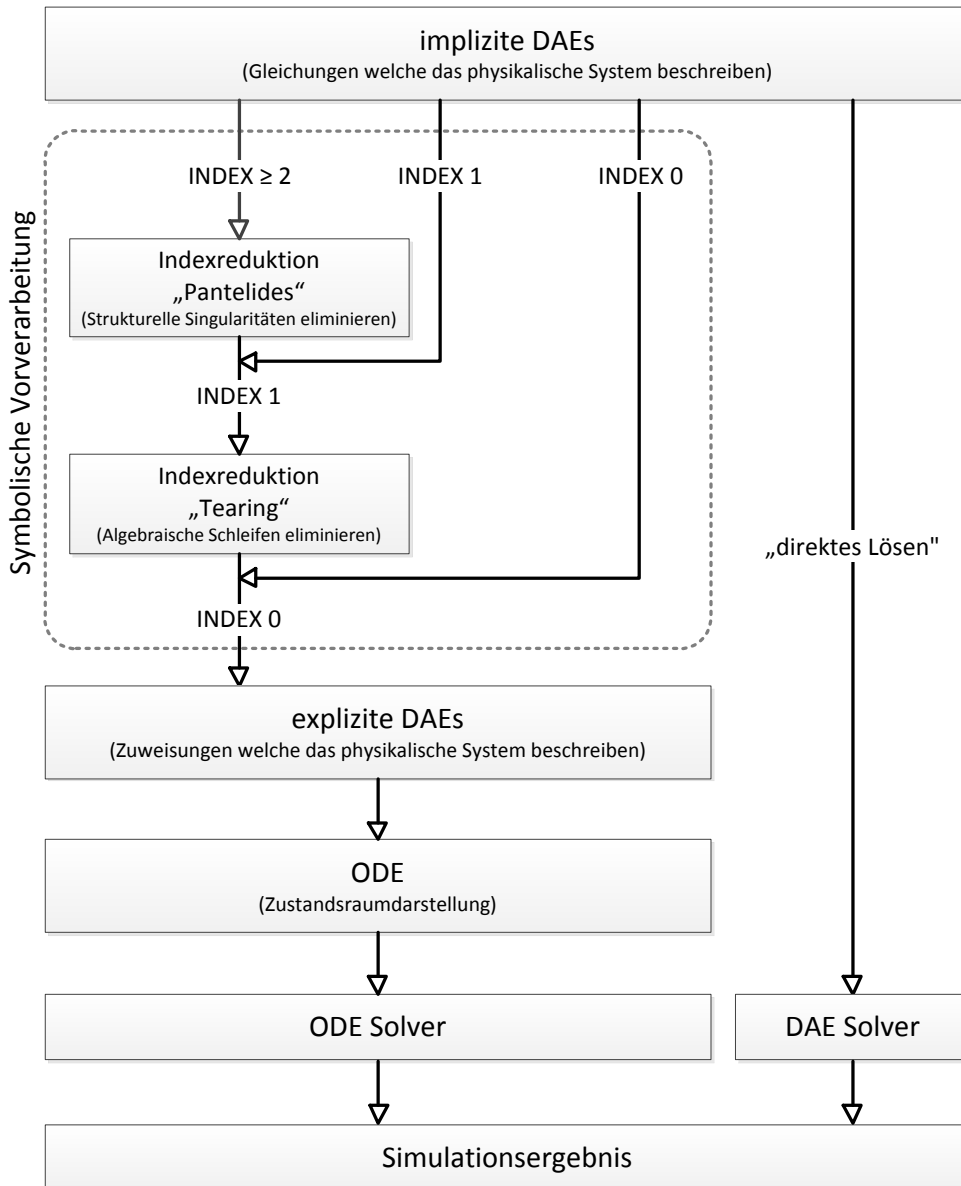


Abb. 6.20.: Mögliche Wege zur Überführung von DAEs zum Simulationsergebnis

7. Simulation und Numerische Integrationsverfahren

Mit einem modellierten technischen System lässt sich ohne die nötige Simulation - oder anders formuliert, das Experimentieren damit - nicht viel anfangen. Daher ist es Standard, dass jede Modellierungsumgebung¹ die nötigen Voraussetzungen für die Simulation mit sich bringt. Es werden also die nötigen Algorithmen zur Verfügung gestellt, um das erstellte Modell in einen vom Rechner ausführbaren Code umzuwandeln. Dabei werden oft mehrere teils sehr komplexe Vorgänge durchgeführt. In diesem Kapitel soll vor allem auf die numerische Integration und deren Einfluss auf die Genauigkeit des Ergebnisses eingegangen werden.

7.1. Einleitung und Allgemeines

Ist ein Modell des technischen Systems erstellt, kann davon eine Simulation ausgeführt werden, um die Reaktion des Modells in verschiedensten Situationen zu testen. Für eine erste Erläuterung wird im vorliegenden Skript davon ausgegangen, dass das mathematische Modell in einer Zustandsraumdarstellung vorliegt, wie sie in [Abbildung 2.6](#) auf Seite 28 zu sehen ist. Dabei werden von dem Modell die Matrizen A , B , C und D definiert. Die in diesem Fall kausalen Summenzeichen können direkt über Zuweisungen in der jeweiligen Programmiersprache realisiert werden. Nun bleibt noch ein Element übrig, welches nicht ohne weiteres implementiert werden kann: Der ideale Integrator $1/s$. Der Zusammenhang dieses Umstandes und der in [Kapitel 2: Einführung in die Zustandsraumdarstellung](#) oft verwendeten grafischen Darstellung ist in [Abbildung 7.1](#) aufgezeigt.

Aufgabe der numerischen Integrationsverfahren ist es grundsätzlich diesen kontinuierlichen Integrator durch eine möglichst genaue Näherung zu ersetzen, während die Zustandsraumdarstellung aus aktuellen Zuständen die Änderungen der Zustände berechnet. Aber wieso ist diese Näherung denn überhaupt nötig? Wieso nicht einfach einen idealen Integrator implementieren? Diese Frage kann relativ einfach beantwortet werden. Das Verhalten eines Systems kann, wie jeder Funktionsverlauf, über die Taylorreihe angenähert werden. Dazu muss der Zustandsvektor $x(t)$ zu einem gewissen Zeitpunkt t bekannt sein, und eine Schrittweite h vorgegeben werden, zu welcher der

¹Also ein Programm, das für die Erstellung von Modellen und deren Simulation erstellt ist.

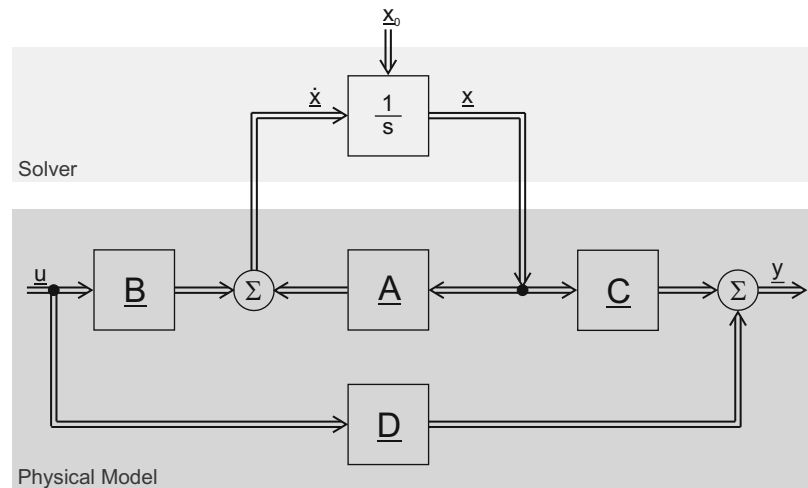


Abb. 7.1.: Veranschaulichung des Zusammenhanges von Integrationsalgorithmen und Zustandsraumdarstellung

nächste Stützwert berechnet werden soll.

$$x_i(t+h) = x_i(t) + \underbrace{\frac{dx_i(t)}{dt}}_{\dot{x}(t)} \cdot h + \underbrace{\frac{d^2x_i(t)}{dt^2}}_{\ddot{x}(t)} \cdot \frac{h^2}{2!} + \dots \quad (7.1)$$

Setzt man nun die Modellgleichung $\dot{x}(t) = f(x(t), u(t), t)$ ein, ergibt sich folgender Zusammenhang, wobei in diesem (linearen, zeitinvarianten Fall) $f = A \cdot x(t) + b \cdot u(t)$ ist.

$$x_i(t+h) = x_i(t) + f_i(t) \cdot h + \frac{df_i(t)}{dt} \cdot \frac{h^2}{2!} + \dots \quad (7.2)$$

Dabei sind zwei Probleme einfach zu erkennen. Als erstes fällt auf, dass diese Reihe für eine exakte Lösung unendlich lang wird, was auch von einem noch so schnellen Rechner nicht in endlicher Zeit bewältigt werden kann. Zweitens ergibt sich das Problem, dass die Funktion $f(t)$, also die Systemmatrix A , sowie die Eingänge u differenziert werden müssen. Das ist für die Systemmatrix durchaus machbar, wenn auch aufwändig, führt aber bei den beliebig vorgebbaren Eingängen zu Problemen.

7.2. Die Annäherung an die Realität

In der Modellierung und Simulation wird generell versucht die Realität nachzubilden. Differenzen entstehen durch Vereinfachungen in der Modellierung selbst, durch nicht ex-

akte Parameter und durch Ungenauigkeiten in der numerischen Integration. Während die ersten beiden Punkte schwer generell zu erläutern sind soll in den nächsten Abschnitten ein wenig auf die Unterschiede zwischen der analytischen und der numerischen Integration eingegangen werden.

7.2.1. Fehler in der numerischen Integration

Aus Implementierung und der digitalen Verarbeitung der Simulationsdaten ergeben sich einige Eigenheiten, welche die numerische Lösung der Simulation von der analytischen Abweichen lassen. Diese sollen in den nächsten Abschnitten beschrieben werden.

Diskretisierungsfehler

Es ist klar, dass es nicht möglich ist, die Taylor-Reihe bis ins Unendliche fortzusetzen. Es wird also eine Näherung erstellt, welche ab einem gewissen Term nicht mehr exakt den richtigen Koeffizienten vor den Termen f_i , df_i/dt , $d^2f_i/dt^2 \dots$ liefert. Es ist dabei egal ob der Koeffizient 0 ist, die Reihe also komplett abgebrochen ist, oder der Term nur von dem Faktor der idealen Reihe abweicht. Der letzte Term der nötigen Approximation welcher exakt dem Term der Taylor Reihe entspricht, gibt also die Ordnung des Näherungsverfahrens an. Gleichung (7.3) gibt demnach einen Lösungsalgorithmus der Ordnung drei an. Die Koeffizienten vor dem Term $\mathcal{O}(h^4)$ und denen höherer Ordnung können beliebige Werte annehmen.

$$x_i(t+h) = x_i(t) + f_i(t) \cdot h + \frac{df_i(t)}{dt} \cdot \frac{h^2}{2!} + \frac{d^2f_i(t)}{dt^2} \cdot \frac{h^3}{3!} + \mathcal{O}(h^4) \quad (7.3)$$

Die Abweichungen dieser Koeffizienten zu denen der Taylor Reihe wird als „Truncation Error“ oder Diskretisierungsfehler bezeichnet. Dieser Fehler steigt also im gegebenen Fall mit der vierten Potenz der Schrittweite h an.

Rundungsfehler

Neben dem Truncation Error gibt es auch noch den Roundoff Error (Rundungsfehler). Dieser ist allerdings meist nur bei Berechnungen in einfacher Präzision (Single Precision) wichtig, da hier schnell die Auflösung der Darstellbarkeit der Zahlen erreicht wird. Da praktisch alle Simulationsprogramme in Double Precision rechnen, ist dieser Fehler nur bei sehr kleinen Schrittweiten von Bedeutung². Wie aus den oberhalb angeführten Taylor Reihen zu erkennen ist, werden Terme addiert welche steigende Potenzen der Schrittweite h enthalten. Wird diese Schrittweite sehr klein und die Potenzen größer

²Eine Ausnahme bildet hier die Hardware in the Loop Simulation bei welcher aus Perfomancgründen oft in Single Precision gerechnet wird.

kann auch die Auflösung der in double precision gespeicherten Werte erreicht werden. Es gibt also für jede Simulation einen idealen Wert der Schrittweite. Es bringt also nicht in jedem Fall eine Verkleinerung der Schrittweite auch eine Verbesserung des Ergebnisses mit sich.

Fehlerfortpflanzung

Die oberhalb erläuterten Abweichungen in der Simulation sind für jeden Simulationsschritt gültig. Daher können sie sich über die gesamte Simulationszeit aufsummieren und beträchtliche Werte annehmen. Durch die zufällige Verteilung der Fehler gleichen sie sich über die Simulationszeit im Normalfall aus. Es kann allerdings bei der Simulation instabiler Systeme passieren, dass die Fehler sich tatsächlich aufkumulieren und somit in Betracht gezogen werden müssen [CK06]. Bei der Simulation üblicher technischer Systeme ist das allerdings nicht der Fall.

7.2.2. Fehlerbegriffe

In den folgenden Abschnitten sollen gängige Begriffe der Simulationstechnik erläutert werden.

Lokaler Fehler

Der lokale Fehler bezeichnet die Abweichung zwischen der exakten analytischen Lösung des Gleichungssystems und der numerischen Berechnung in *einem Zeitschritt* passiert. Der lokale Fehler hängt somit von der gewählten Schrittweite h ab. Eine weitere Abhängigkeit ergibt sich von der Ordnung des numerischen Lösungsverfahrens.

Für die beiden grundlegenden Integrationsverfahren ist in [Sue04] Ordnung des Zusammenhangs von lokalem Fehler und Schrittweite hergeleitet. Für das in [Abschnitt 7.3.1](#) erläuterte Vorwärtseulerverfahren steigt der lokale Fehler e_L mit quadratischer Abhängigkeit von der Schrittweite h an, $e_L = O(h^2)$. Für das in [Abschnitt 7.3.2](#) vorgestellte Heun-Verfahren gilt $e_L = O(h^3)$.

Es ist also generell davon auszugehen, dass der lokale Fehler mit steigender Ordnung der Integrationsverfahren und gleicher Schrittweite abnimmt. Allerdings ist die Abhängigkeit von der Schrittweite von einer höheren Ordnung. Daraus ergibt sich, dass bei einer Vergrößerung der Schrittweite der lokale Fehler eines Verfahrens höherer Ordnung größer werden kann als der eines Verfahrens niedrigerer Ordnung. Generell gilt allerdings, dass bei einer sinnvollen Wahl der Schrittweite Verfahren höherer Ordnung genauer sind, als Verfahren niedrigerer Ordnung.

Globaler Fehler

Der globale Fehler gibt die Abweichung des numerisch errechneten Ergebnisses nach der gesamten Simulationszeit zu der exakten Lösung an. Generelle wird erwartet, dass eine kleiner Schrittweite zu genaueren Ergebnissen führt. Dies mag naheliegend sein, ist allerdings in Anbetracht des z.B. Rundungsfehlers welche mit sinkender Schrittweite steigenden Einfluss hat nicht generell gültig. Weit verbreitete Implementierungen von Lösungsalgorithmen sorgen allerdings im Allgemeinen dafür, dass die unteren Grenzen der Schrittweite so gelegt werden, dass diese Fehler nicht von Bedeutung sind.

7.3. Einschrittverfahren

Für die folgenden Betrachtungen wird von einem linearen, zeitinvarianten System ausgegangen. Es wird also:

$$A(x, t) = A = \text{const.} \quad (7.4)$$

Dadurch werden diese grundlegenden Untersuchungen erleichtert bzw. erst möglich, da manche der angewandten Kriterien für nichtlineare Systeme nicht oder nur teilweise gültig sind.

7.3.1. Eulerverfahren

Die Eulerverfahren sind die Grundlage aller Einschrittverfahren und werden deshalb vergleichsweise genau betrachtet. Es ist weiters wichtig zu bemerken, dass sich jedes Einschrittverfahren in seiner niedrigsten Ordnung³ auf ein Eulerverfahren reduziert.

Vorwärtseuler oder explizites Eulerverfahren

Das Vorwärtseuler (engl. Forward Euler - FE) Verfahren, oder auch einfach Eulerverfahren, ist die einfachste Möglichkeit einen analytischen Integrator numerisch zu approximieren. Dabei wird die Taylorreihe nach dem linearen Term - also der ersten Ableitung - abgebrochen. Daraus ergibt sich folgende Gleichung:

$$x(t+h) = x(t) + \dot{x}(t) \cdot h \quad (7.5)$$

$$x(t+h) = x(t) + f(x, t) \cdot h \quad (7.6)$$

$$x(t+h) = x(t) + A \cdot x(t) \cdot h \quad (7.7)$$

³Zur Erinnerung: Die Anzahl der exakt approximierten Terme der Taylor Reihe.

Gleichung (7.5) ist dabei die allgemeinste Formulierung des Vorwärtseuler Verfahrens. In Gleichung (7.6) wurde Gleichung (6.1) eingesetzt. Der Einfachheit halber wird das System auf ein autonomes System reduziert, das also keine Eingangsgrößen hat. Das Verfahren ist in [Abbildung 7.2](#) noch einmal grafisch veranschaulicht.

Dabei ist es wichtig zu erwähnen, dass das Gleichheitszeichen in Gleichungen (7.5) bis (7.7) nicht bedeutet, dass die analytische Lösung von dem numerischen Integrationsalgorithmus exakt berechnet wird. Allerdings wird die numerische Lösung aus diesen Gleichungen berechnet und daher finden die Gleichheitszeichen Verwendung.

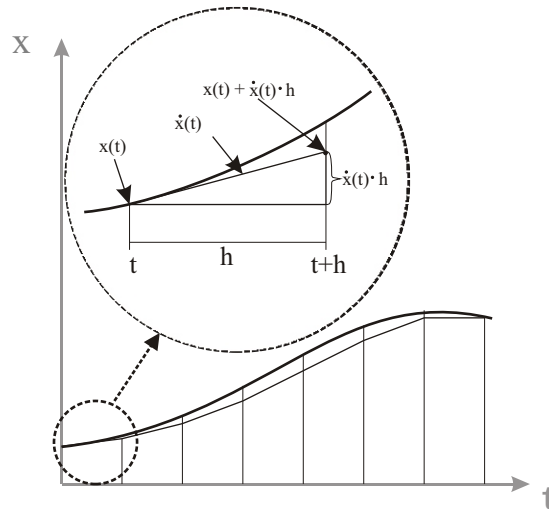


Abb. 7.2.: Veranschaulichung des Vorwärtseulerverfahren

Um die Veränderungen darzustellen, welche der Übergang zur diskreten Berechnung mit sich bringt wird eine von der ursprünglichen Zustandsraumdarstellung

$$\dot{\underline{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t) \quad (7.8)$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) + \underline{D} \cdot \underline{u}(t) \quad (7.9)$$

abweichende Form verwendet. Diese wird üblicherweise in der in (7.10) dargestellten Form angeschrieben.

$$\underline{x}(t+h) = \underline{F} \cdot \underline{x}(t) + \underline{G} \cdot \underline{u}(t) \quad (7.10)$$

$$\underline{y}(t) = \underline{H} \cdot \underline{x}(t) + \underline{I} \cdot \underline{u}(t) \quad (7.11)$$

Dabei bleiben die Bezeichnungen der einzelnen Matrizen bzw. Vektoren erhalten. F ist also die diskrete Zustandsmatrix, G ist die Eingangsmatrix, H ist die Ausgangsmatrix und I ist die Durchgangsmatrix.

Ein äußerst wichtiger Unterschied zwischen den beiden Beschreibungsformen ist, dass

man in der diskreten Version nicht mehr die Ableitung der aktuellen Zustände berechnet, sondern die *absoluten Werte der Zustände* zum Zeitpunkt $t + h$.

Der Übersichtlichkeit halber reduzieren wir unser System wieder auf ein skalares, autonomes System betrachten nur noch die Zustandsgleichung. Somit erhalten wir

$$x(t + h) = F \cdot x(t) \quad (7.12)$$

Einige Worte zur Diskretisierung

Aus der Mathematik bzw. Regelungstechnik wissen wir, dass es für ein stabiles System nötig ist, dass alle Pole des Systems in der linken Halbebene der kontinuierlichen s -Ebene liegen wie es in [Abbildung 7.3](#) dargestellt ist. Über die z -Transformation wird dieser analytisch stabile Bereich, in die z -Ebene transformiert, wo der Stabilitätsbereich innerhalb des Einheitskreises liegt. Dies soll im Folgenden noch einmal kurz erläutert werden. Der Abschnitt ist in [Unb00] in einer ähnlichen Form niedergeschrieben und etwas genauer diskutiert. [Unb00] dient somit als Grundlage für diesen Abschnitt.

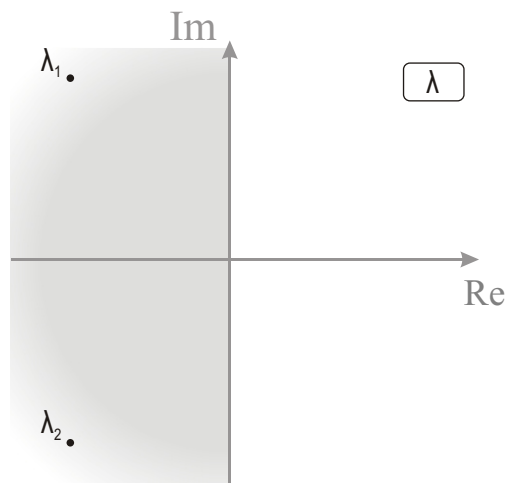


Abb. 7.3.: Darstellung der analytisch stabilen Region (grau dargestellt) in der kontinuierlichen λ Ebene.

Um den Zusammenhang zwischen s - und z -Ebene zu wiederholen bzw. auf einer andere Art darzustellen, rufen wir uns folgenden Zusammenhang in Erinnerung. Dabei verwenden wir anstatt der in der Mathematik üblichen Variable T für die Abtastzeit die Variable h , um mit dem Rest des Dokumentes konsistent zu bleiben.

$$z = e^{sT} = e^{s \cdot h} \quad (7.13)$$

Mit

$$s = \sigma + j\omega \quad (7.14)$$

wird daraus

$$|z| = e^{\sigma \cdot h} \quad \text{und} \quad \arg(z) = \phi = \omega \cdot h \quad (7.15)$$

Bildet man nun den Ursprung der s-Ebene ($\sigma = 0, j\omega = 0$) in die z-Ebene ab, ergibt sich

$$z = e^{s \cdot h} = e^{(\sigma + j\omega)h} = e^{0 + j0} = 1 + j0 \quad (7.16)$$

Erhöht man den Wert von $j\omega$ kontinuierlich, wandert man in der s-Ebene der imaginären Achse entlang nach oben. Man bewegt sich also an der Grenze der analytischen Stabilität. Betrachtet man wiederum (7.15) bzw. (7.16) ändert sich durch die Vergrößerung des Wertes von $j\omega$ nur der Winkel des Punktes in der z-Ebene. Eine Bewegung entlang der imaginären Achse in der s-Ebene bewirkt also eine Veränderung des Winkels in der z-Ebene, wobei eine positive Änderung von $j\omega$ einen sich vergrößernden Winkel ϕ zur Folge hat. Der Punkt in der z-Ebene bewegt sich also auf einem Kreis mit dem Radius 1 um den Ursprung: der Einheitskreis. Dabei wird auch klar, dass Abhängig von der Schrittweite h der Punkt $1 + j0$ in der z-Ebene mehrfach durchlaufen wird. Es gibt also für den Punkt $1 + j0$ in der z-Ebene unendlich viele Äquivalente in der s-Ebene [Unb00].

Der Zusammenhang zwischen (7.12) erkennen wir über eine z-Transformation von (7.12).

$$x(t + h) = F \cdot x(t) \quad (7.17)$$

ergibt nach der z-Tranformation

$$z \cdot (x(z) - x(0)) = F \cdot x(z) \quad \text{mit } x(0) = 0 \text{ und nach Division durch } z \quad (7.18)$$

$$x(z) = \frac{1}{z} (F \cdot x(z)) \quad (7.19)$$

$$x(z) = F \cdot (x(z) \cdot z^{-1}) \quad (7.20)$$

Wir haben nun also erkannt, dass (7.12) nur stabil ist, wenn die Eigenwerte der F Matrix innerhalb des Einheitskreises liegen. Um herauszufinden wie diese Bedingung bei der Anwendung des Vorwärtseulers erfüllt werden kann, vergleichen wir (7.7) und

(7.12). Über einer einfach Umformung von (7.7) erkennen wir Folgendes.

$$x(t+h) = x(t) + A \cdot x(t) \cdot h \quad (7.21)$$

$$x(t+h) = \underbrace{(\mathbb{1} + A \cdot h)}_F \cdot x(t) \quad (7.22)$$

Nun ist der Zusammenhang zwischen Gleichung (7.12) und der z-Transformation klar. Um die Stabilität des diskretisierten Systems zu garantieren müssen die Pole der transformierten Systemmatrix F im Einheitskreis liegen. Hier liegt aber nicht mehr nur eine einfache Diskretisierung vor. Die Addition der Einheitsmatrix in (7.22) bewirkt eine Verschiebung des Einheitskreises um 1 nach links. Aus der Verschiebung des Stabilitätsbereiches ergibt sich der in [Abbildung 7.4](#) dargestellte Stabilitätsbereich des Vorwärtseuler Integrationsalgorithmus. Wir müssen also nun dafür sorgen, dass die diskreten Pole unseres Systems innerhalb dieses Stabilitätsbereiches liegen, damit wir ein korrektes Stabilitätsverhalten der numerischen Simulation erhalten. Liegen die Pole der numerischen Simulation zwar in der linken Halbebene, aber außerhalb dieses in [Abbildung 7.4](#) dargestellten stabilen Bereiches, werden Simulationen von analytisch stabilen Systemen instabile numerisch berechnete Ergebnisse liefern.

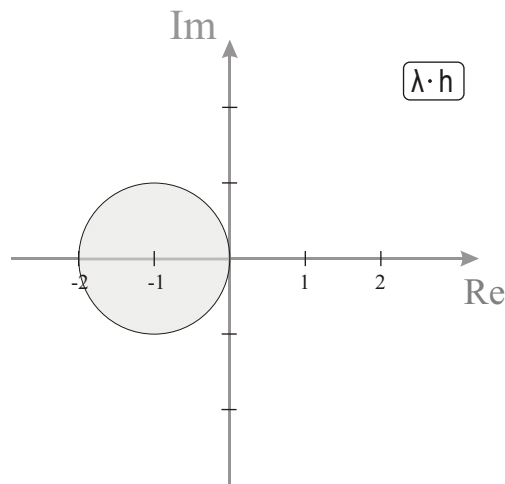


Abb. 7.4.: Stabile Region des Vorwärtseuler Verfahrens (grau dargestellt) in der diskreten $\lambda \cdot h$ Ebene.

Wichtig ist zu erkennen, dass es sich in den Beschreibungen von analytischer Stabilität ([Abbildung 7.3](#)) und numerischer Stabilität (z.B. [Abbildung 7.4](#)) um verschiedene Ebenen handelt. In [Abbildung 7.3](#) wird die kontinuierliche λ Ebene dargestellt, in welcher die Eigenwerte der Systemmatrix A dargestellt werden. In [Abbildung 7.4](#) wird hingegen die diskrete $\lambda \cdot h$ Ebene dargestellt. Diese Beschreibung soll verdeutlichen, wie über die

Schrittweite h Einfluss auf die Stabilität der Simulation genommen werden kann.

Aus (7.15) erkennen wir weiters, dass wir die Lage der Pole in der $\lambda \cdot h$ Ebene mit der Schrittweite h in ihrem Betrag und Winkel beeinflussen können. Es ist uns also möglich, die Pole des numerischen Systems über die Schrittweite zu verschieben ohne Einfluss auf die analytische Beschreibung zu nehmen. Somit können wir über die Wahl der Schrittweite die numerische Stabilität der Simulation beeinflussen. Unsere Aufgabe besteht nun also darin eine für das gegebene analytische System passende Schrittweite zu wählen, um numerische Stabilität zu garantieren. Wird die Schrittweite kleiner gewählt nähern sich die Pole dem Ursprung der $\lambda \cdot h$ Ebene. Pole in der linken s-Halbebene können also in den numerisch stabilen Bereich verschoben werden, während Pole in der die instabilen pole der rechten s-Halbebene nicht in den stabilen Bereich gelangen können⁴. Es kann aus diesem Umstand relativ einfach auf eine Stabilitätsbedingung für die Wahl von h geschlossen werden. Diese ergibt sich wie in [Abbildung 7.5](#) dargestellt aus dem Verhältnis des Abstands der Polstelle zu dem Schnittpunkt mit dem Stabilitätsbereichs.

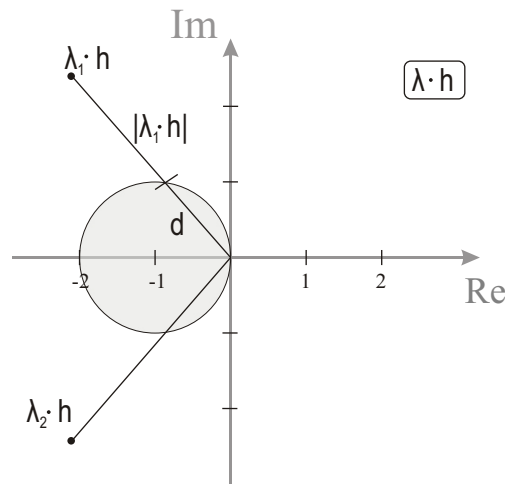


Abb. 7.5.: Zusammenhang von h und der Stabilität

Dies lässt sich mathematisch mittels folgender Gleichung beschreiben.

$$h_{max} = \frac{d}{|\lambda_1|} \tag{7.23}$$

Wegen der Einfachheit und guten Verständlichkeit des Vorwärtseuler Verfahrens wird

⁴Später wird sich zeigen, dass aus Solver existieren, bei denen Pole aus der instabilen rechten s-Halbebene in den numerisch stabilen Bereich gelangen können, was komplett falsche Simulationsergebnisse zur Folge hat.

dieses auch heute noch regelmäßig eingesetzt, obwohl deutlich fortgeschrittenere Verfahren existieren. Eine Implementierung des Vorwärtseulerverfahrens könnte in MATLAB folgendermaßen aussehen, wie es in [Code 7.1](#) dargestellt ist.

```

1  % Paramters
2  A = -1;
3  c = 1;
4
5  % Initial Conditions
6  x0 = 1;
7
8  % Simulation Properties
9  t_start = 0;
10 t_end = 10;
11 h = 1e-3;
12
13 % Artificial Variables
14 i = 1; \
15 x = zeros(size(t_start:h:t_end));
16 x(i) = x0;
17
18 for t = t_start:h:t_end
19     % FE
20     dx_dt(i) = A*x(i);
21     x(i+1) = x(i) + dx_dt(i)*h;
22
23     % for Plotting
24     t_vec(i) = t;
25     i = i+1;
26 end

```

Code 7.1: Implementierung eines einfachen Vorwärtseuleralgorithmus in Matlab

Dabei ergeben sich für verschiedene Schrittweiten die in [Abbildung 7.6](#) dargestellten Ergebnisse.

Eine alternative Stabilitätsbetrachtung ergibt sich aus der BIBO (bounded input, bounded output) Stabilität. Diese besagt, dass sich für jeglichen beschränkten Input auch ein beschränkter Output ergeben muss. Daher ergibt sich für (7.22) das Kriterium < 1 . Dies resultiert über ähnliche Schritte wie in der ersten Erläuterung dargestellt in der folgenden Stabilitätsdomänen. Der stabile Bereich ist dabei grau hinterlegt.

Rückwärtseuler oder implizites Eulerverfahren

Eine eher unscheinbare Änderung des Vorwärtseuler Verfahrens führt zum Rückwärtseuler (engl. Backward Euler - BE) Verfahren. Die Änderung bezieht sich auf den Zeitpunkt zu welchem die Ableitung durchgeführt wird. Diese wird beim Rückwärtseuler

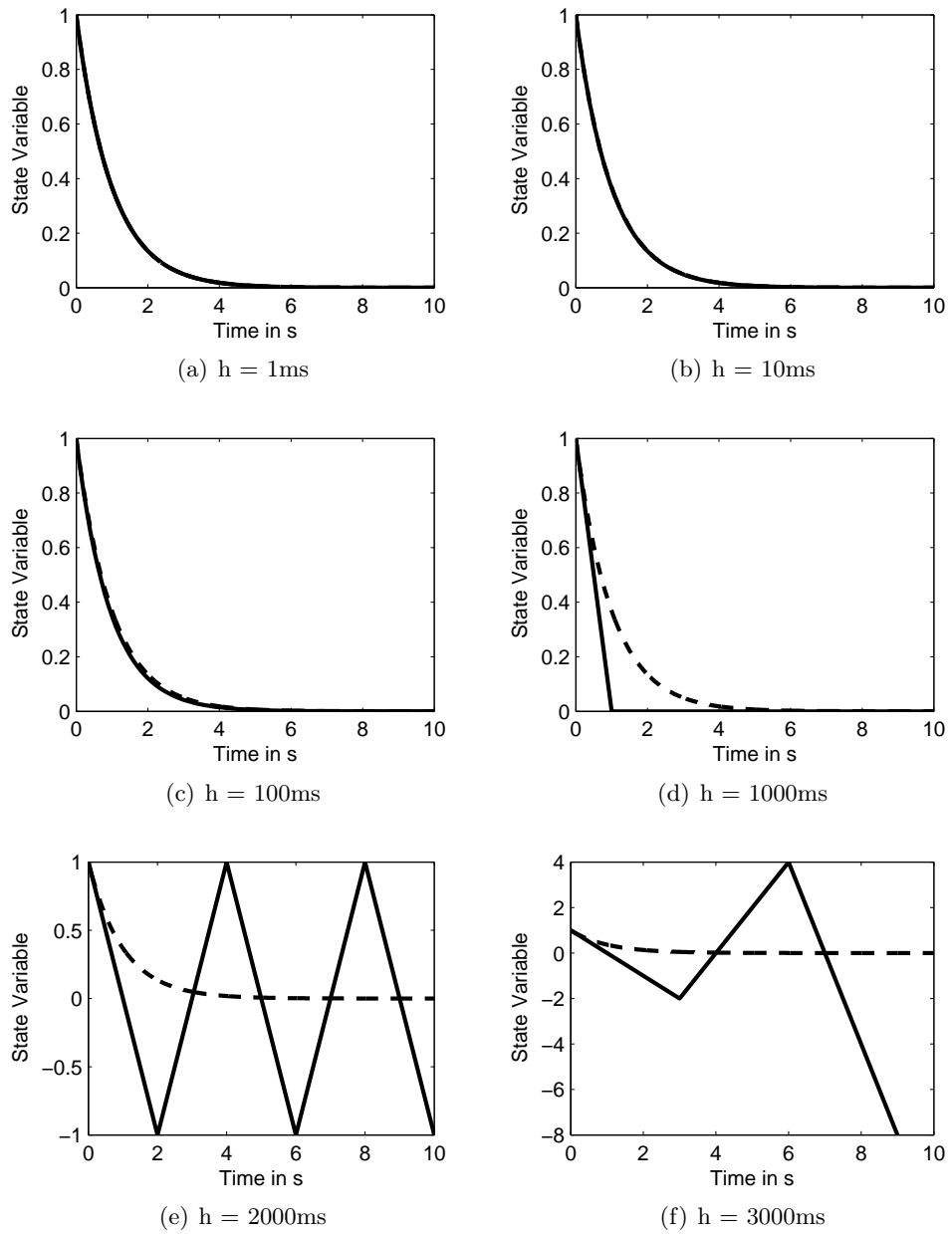


Abb. 7.6.: Der Einfluss der Schrittweite auf das Ergebnis bei einem Vorwärtseuler Algorithmus.

nicht mehr von dem aktuellen Wert durchgeführt, sondern von dem in der Zukunft liegenden Punkt $x(t+h)$. Das bringt komplett veränderte Stabilitätseigenschaften und eine deutlich kompliziertere Berechnung mit sich. Die Gleichung des Rückwärtseuler Verfahrens lautet wie folgt:

$$x(t+h) \approx x(t) + \dot{x}(t+h) \cdot h \quad (7.24)$$

$$x(t+h) \approx x(t) + f(x(t+h), t+h) \cdot h \quad (7.25)$$

$$x(t+h) \approx x(t) + A(x, t) \cdot x(t+h) \cdot h \quad (7.26)$$

$$x(t+h) \approx (\mathbb{1} - A(x, t) \cdot h)^{-1} \cdot x(t) \quad (7.27)$$

Eine grafische Veranschaulichung des Verfahrens ist in [Abbildung 7.7](#) dargestellt.

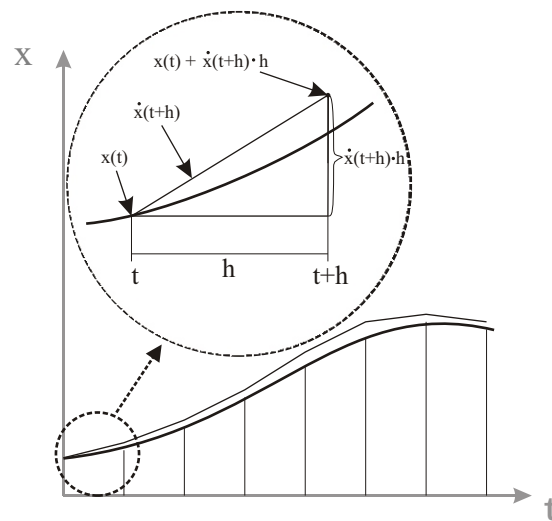


Abb. 7.7.: Veranschaulichung des Rückwärtseulerverfahren

Aus diesem Verfahren ergibt sich die in [Abbildung 7.8](#) dargestellte Stabilitätsregion. Grundsätzlich bietet sich also hier der Vorteil, dass die Wahl der Schrittweite h für analytisch stabile Systeme theoretisch keinen Einfluss auf die numerische Stabilität des Verfahrens hat.

Über Gleichung (7.27) scheint die Berechnung des neuen Zustandwertes also problemlos möglich zu sein. Dies ist allerdings nur in dem hier angenommenen linearen Fall so einfach möglich. Bedenken wir noch einmal, dass wir uns für diese Betrachtungen auf lineare zeitinvariante Problemstellungen beschränkt haben. In diesem Fall ist das implizite Rückwärtseulerverfahren⁵ über eine analytische Gleichung direkt lösbar. Wird aber von einem nichtlinearen System ausgegangen, ist die Systemmatrix von den Zustands-

⁵Auch andere implizite Verfahren haben diese Eigenschaft für LTI Systeme.

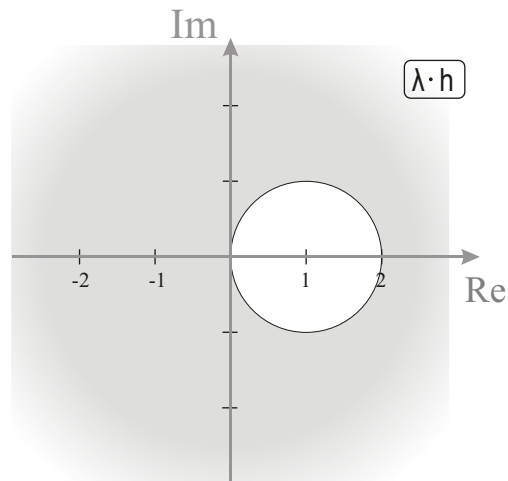


Abb. 7.8.: Stabile Region des Rückwärtseuler Verfahrens, in grau dargestellt.

variablen abhängig, $A = f(x)$, erweitert sich der Term $A \cdot x(t+h)$ zu $A(x(t+h)) \cdot x(t+h)$ wodurch Gleichung (7.27) nicht mehr lösbar ist, da zwei Unbekannte in ihr vorkommen, $x(t+h)$ und $A(x(t+h))$. Der Wert von $x(t+h)$ muss also über eine Iteration angenähert werden. Die gängigste Lösung dieses Problems wird im folgenden Abschnitt dargestellt.

Bei der Implementierung des Rückwärtseuleres für lineare Systeme muss also nur die Schleife aus Code 7.1 auf die in Code 7.2 angepasst werden. Für verschiedene Schrittweiten stellt sich das in Abbildung 7.9 gezeigte Ergebnis ein.

```
1 for t = t_start:h:t_end
2     % BE
3     x(i+1) = (1-A*h)^-1 * x(i);
4
5     % for Plotting
6     t_vec(i) = t;
7     i = i+1;
8 end
```

Code 7.2: Schleife zur Berechnung eines Rückwärtseuleres eines homogenen Zustandsraumes

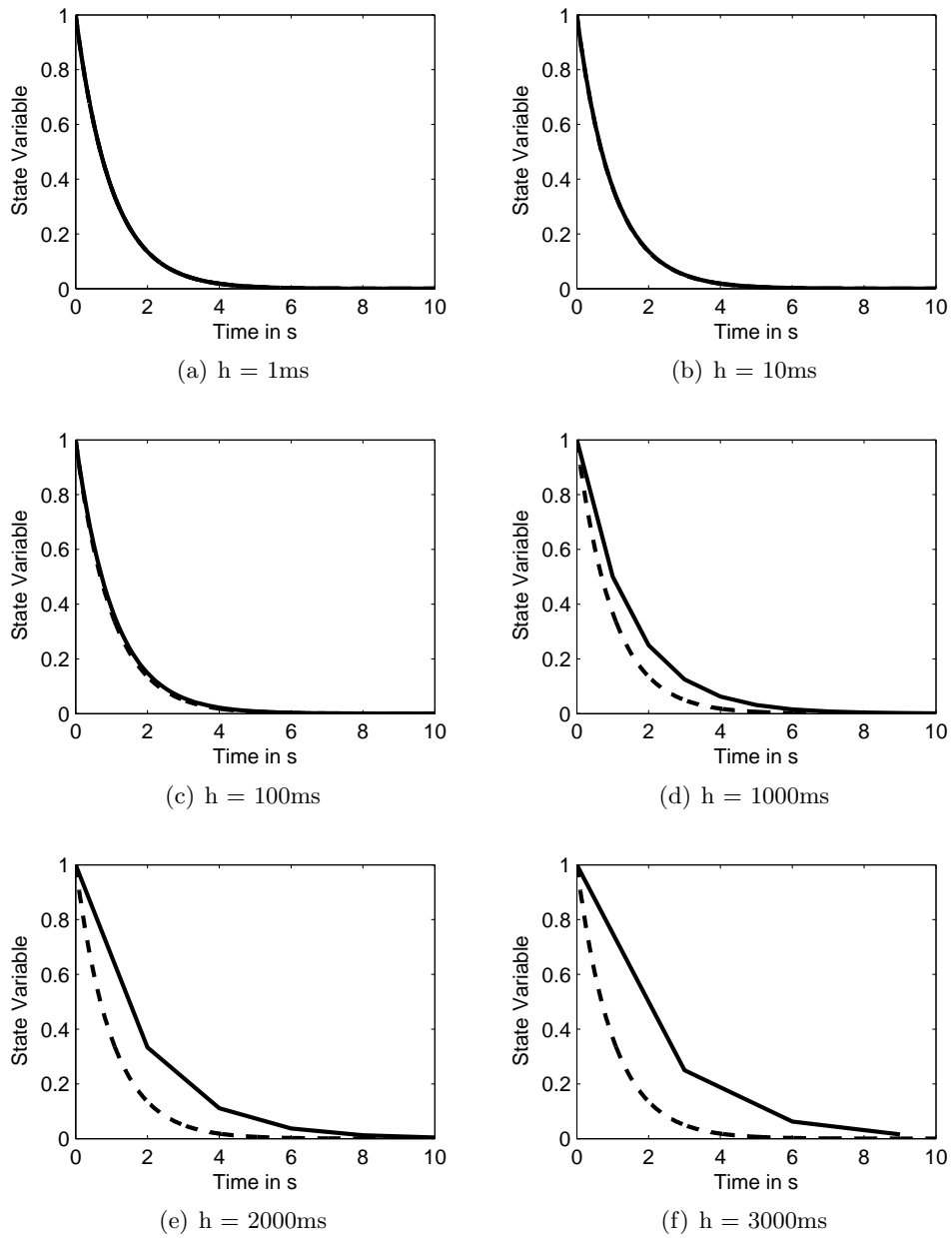


Abb. 7.9.: Der Einfluss der Schrittweite auf das Ergebnis bei einem Rückwärtseuler Algorithmus.

Newton Iteration

Für die Lösung des Iterationsproblems bieten sich mehrere Möglichkeiten an. Die verbreitetste Lösung dafür ist die Newton Iteration. Diese wird im Folgenden überblicksartig erläutert.

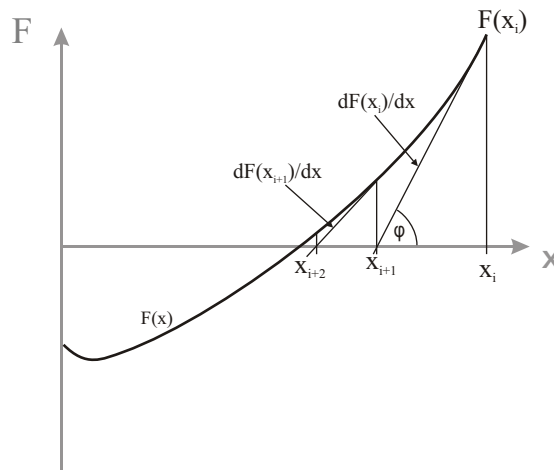


Abb. 7.10.: Newton Iteration zur Ermittlung eines Nulldurchgangs

Dabei ergeben sich aus [Abbildung 7.10](#) die folgenden Zusammenhänge.

$$\tan(\phi) = \frac{\partial F^i}{\partial x} = \frac{F^i - 0}{x^i - x^{i+1}} \quad (7.28)$$

Wobei i den aktuellen Iterationsschritt angibt. Daraus ergibt sich über eine Umwandlung folgende Approximation für den nächsten Iterationswert.

$$x^{i+1} = x^i - \frac{F^i}{\partial F^i / \partial x} \quad (7.29)$$

Diese Iterationsgleichung wird nun so oft ausgeführt, bis F^i gegen 0 geht. Üblicherweise definiert man hier einen kleinen Wert, der als Abweichung toleriert werden kann, also sich über viele Simulationsschritte nicht zu einem zu großen Fehler kumuliert.

Um dieses Verfahren auf den Rückwärtseuler anwenden zu können, muss ein Ausdruck für F gefunden werden. Da die Newton Iteration einen Nulldurchgang der Funktion F sucht, muss diese als eine Funktion formuliert werden, die 0 ergibt wenn der gewünschte Wert erreicht ist. Dies ist aus Gleichung (7.26) über eine einfache Umformung zu finden.

Es ergibt sich also folgendes.

$$F \equiv x(t) + h \cdot f(x(t+h)) - x(t+h) = 0 \quad (7.30)$$

Zusätzlich zu F ist noch $\partial F/\partial x$ nötig. Dies ergibt sich aus der obigen Gleichung über deren Ableitung nach x_{t+h} zu:

$$\frac{\partial F}{\partial x} = h \cdot \frac{\partial f(x(t+h))}{\partial x} - 1 \quad (7.31)$$

Der erste Wert von $x(t+h)$ muss dabei geschätzt werden und wird x_i genannt. Er stellt also den Startwert der Iteration dar. Man kann bei einer kleinen Schrittweite z.B. davon ausgehen, dass er nicht zu weit vom aktuellen Zustand $x(t)$ abweicht, also könnte man $x^i = x(t)$ setzen. Da dies allerdings nur eine Möglichkeit ist den Startwert der Iteration zu wählen, wird dieser in weiterer Folge mit x_i bezeichnet.

$$F^i = x(t) + h \cdot f(x^i) - x^i \quad (7.32)$$

$$\frac{\partial F^i}{\partial x} = h \cdot \frac{\partial f(x^i)}{\partial x} - 1 \quad (7.33)$$

Daraus ergibt sich nun wiederum die Gesamtgleichung für die Iteration zu folgender. Wichtig ist es dabei zu erkennen, dass die Newton Iteration die Zustände x als unabhängige Variablen auf der horizontalen Achse aufträgt, wie das auch in [Abbildung 7.10](#) dargestellt ist. Es kommen nur die Variablen $x(t)$ und Iterationsvariablen zum Zeitpunkt $t+h$ vor. Daher wurde wegen einer besseren Leserlichkeit bei den Iterationsvariablen die ausdrückliche Kennzeichnung des Zeitpunktes nicht vorgenommen. Alle Iterationsvariablen gelten zum Zeitpunkt $t+h$, $x^i(t+h) \rightarrow x^i$ bzw. $x^{i+1}(t+h) \rightarrow x^{i+1}$.

$$x^{i+1} = x^i - \frac{x(t) + h \cdot f(x^i, t+h) - x^i}{h \cdot \partial f(x^i, t+h)/\partial x - 1} \quad (7.34)$$

Problematisch bei der Newtoniteration ist, dass nicht garantiert werden kann, dass diese in einer bestimmten Anzahl von Iterationen einen Endwert liefert. Dabei ist für Systeme mit starken Nichtlinearitäten auch nicht garantiert, dass die Iteration überhaupt ein sinnvolles Ergebnis liefert, also konvergiert. Wählt man in [Abbildung 7.10](#) einen Schätzwert zu weit links wird die Iteration nicht konvergieren. Es ist daher eine nicht zu große Schrittweite zu wählen, um ein sinnvolles Ergebnis zu garantieren. Somit ist bei der Anwendung des Rückwärtseulers mit einer Newtoniteration die Schrittweite durch die Eigenschaften der Iteration begrenzt.

Für den Fall, dass ein lineares System mittels einer Newton Iteration simuliert wird, ergibt sich in [Abbildung 7.10](#) F zu einer Geraden. Daher ist die Iteration nach dem ersten Schritt abgeschlossen. Für lineare Systeme ist eine direkte Berechnung trotzdem

effizienter.

Jakobi und Hess'sche Matrix

Diese beiden Begriffe werden in der Simulationstechnik oft verwendet und werden auch oft verwirrend oder sogar fälschlich eingesetzt. Daher sollen sie hier kurz erläutert werden.

Die Jakobi Matrix ist folgendermaßen definiert.

$$J = \frac{\partial f}{\partial x} = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \cdots & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \cdots & \partial f_2/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \partial f_n/\partial x_2 & \cdots & \partial f_n/\partial x_n \end{pmatrix} \quad (7.35)$$

Setzt man nun also für $f = A \cdot x + B \cdot u$ ein, muss dieser Ausdruck nach den Zuständen abgeleitet werden. Da die Eingänge eines Systems nicht von dessen Zuständen, sondern von der Zeit abhängen, fällt der zweite Term weg. Kommen in der Systemmatrix A keine Zustände vor, das System ist also linear, entspricht die Jakobimatrix J genau der Systemmatrix A .

$$J = A \dots \text{für ein lineares System} \quad (7.36)$$

Ist das der Fall vereinfachen sich viele Berechnungen deutlich. Es gibt für lineare Systeme auch spezielle Lösungsalgorithmen. Einer davon wäre der Adams-Bashforth⁶, welcher mit der Ordnung drei z.B. von der `lsim` Funktion von MATLAB verwendet wird.

Die Hess'sche Matrix ist Teil der Newton Iteration. Dabei wird die Funktion F (7.30), von welcher der Nulldurchgang gesucht wird, nach deren unabhängigen Variablen abgeleitet. In unserem Fall sind dies die Zustandsvariablen x . Es ergibt sich also für die Hess'sche Matrix H

$$H = \frac{\partial F}{\partial x} = \begin{pmatrix} \partial F_1/\partial x_1 & \partial F_1/\partial x_2 & \cdots & \partial F_1/\partial x_n \\ \partial F_2/\partial x_1 & \partial F_2/\partial x_2 & \cdots & \partial F_2/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial F_n/\partial x_1 & \partial F_n/\partial x_2 & \cdots & \partial F_n/\partial x_n \end{pmatrix} \quad (7.37)$$

Es lässt sich jetzt also die Newtoniteration für den Rückwärtseuler folgendermaßen

⁶Adams-Bashforth ist im Unterschied zu den in diesem Abschnitt vorgestellten Algorithmen ein Mehrschrittverfahren. Diese werden im folgenden Abschnitt kurz beleuchtet.

darstellen.

$$\begin{aligned}
 x_{t+h}^{i+1} &= x_{t+h}^i - \frac{x_k + h \cdot f(x_{t+h}^i, t+h) - x_{k+1}^i}{\underbrace{h \cdot \partial f(x_{t+h}^i, t+h) / \partial x}_{J_{k+1}^i}} = \\
 &= x_{t+h}^i - \frac{\overbrace{x_t + h \cdot f(x_t + h \cdot f(x_{t+h}^i), t+h) - x_{t+h}^i}^{F^i}}{\underbrace{(h \cdot J_{k+1}^i - 1)}_{H^i = \partial F^i / \partial x}} = \\
 &= x_{t+h}^i - H^{i-1} \cdot F^i
 \end{aligned} \tag{7.38}$$

Oft wird z.B. bei der Berechnung der Newton Iteration angenommen, dass die Jakobi und somit auch die Hess'sche Matrix während einer Iteration konstant bleiben und werden daher nicht für jeden Iterationsschritt erneut berechnet. Dies ist für sinnvolle Fälle auf jeden Fall gerechtfertigt und beschleunigt die Berechnung deutlich, da vor allem die Invertierung der Hess'schen Matrix sehr aufwändig werden kann.

Eine Umsetzung der Newton Iteration in Pseudocode könnte also wie in [Code 7.3](#) lauten:

```

1  while (Fi > 1e-5)
2  {
3      Fi = x(t) + h * J - xi
4      H = h * J - 1
5      xi_1 = xi - H^-1 * F
6      xi = xi_1
7  }

```

Code 7.3: Pseudocode einer Newton Iteration

7.3.2. Runge Kutta Verfahren

Allen Einschrittverfahren stehen nur die Informationen über die aktuellen Zustandsvariablen x und deren zeitliche Ableitung $\dot{x} = A \cdot x + b \cdot u$ zur Verfügung. Um Lösungsalgorithmen mit einer höheren Ordnung erstellen zu können, werden Informationen „zwischen“ den eigentlichen Simulationsschritten errechnet. Dies wird im Folgenden genauer erläutert.

Die Herleitung dieser Klasse von Lösungsverfahren kann gut über das „Predictor-Corrector“ Schema erklärt werden. Dabei werden die geschätzten Predictor Werte (Vorwärtseuler) durch Corrector Werte (Rückwärtseuler) verbessert, so ist eine Iteration auf einen korrekten Endwert möglich. Danach werden Vorwärts- und Rückwärtseuler

Schritte mit verschiedenen Schrittweiten und Gewichtungsfaktoren kombiniert und aufsummiert.

Predictor:

$$\dot{x}(t) = f(x, t) \quad (7.39)$$

$$x^P(t+h) = x(t) + h \cdot \dot{x}(t) \quad (7.40)$$

Corrector:

$$\dot{x}^P(t+h) = f(x^P(t+h), t+h) \quad (7.41)$$

$$x^C(t+h) = x(t) + h \cdot \dot{x}^P(t+h) \quad (7.42)$$

Daraus ergibt sich, wenn die Gleichungen von oben beginnend ineinander eingesetzt werden, folgendes Ergebnis:

$$x(t+h) = x^C(t+h) = x(t) + h \cdot f(x(t), t) + h \cdot f(x^P(t+h), t+h) \quad (7.43)$$

Um den Ausdruck $f(x(t) + h \cdot f(x(t), t), t+h)$ zu beschreiben, bietet sich eine mehrdimensionale Taylorreihe an.

$$f(x + \Delta x, y + \Delta y) \approx f(x, y) + \frac{\partial f(x, y)}{\partial x} \cdot \Delta x + \frac{\partial f(x, y)}{\partial y} \Delta y \quad (7.44)$$

Ersetzt man nun folgende Ausdrücke

$$x = x(t) \text{ bzw. } \Delta x = h \cdot f(x, t) \quad (7.45)$$

$$y = t \text{ bzw. } \Delta y = h \quad (7.46)$$

ergibt sich folgender Zusammenhang

$$f(x(t) + h \cdot f(x, t), t+h) \approx f(x, t) + \frac{\partial f(x, t)}{\partial x} \cdot h \cdot f(x, t) + \frac{\partial f(x, t)}{\partial t} \cdot h \quad (7.47)$$

Durch Einsetzen des Ausdrucks (7.47) in (7.43) und wenigen Umformungen⁷ ergibt sich daraus folgendes.

$$x(t+h) \approx x(t) + h \cdot f(x, t) + h^2 \cdot \left(\frac{\partial f(x, t)}{\partial x} \cdot f(x, t) + \frac{\partial f(x, t)}{\partial t} \right) \quad (7.48)$$

Es ist immer noch nicht klar, was der Ausdruck $\frac{\partial f(x, t)}{\partial x} \cdot f_t(x, t) + \frac{\partial f(x, t)}{\partial t}$ genau bedeutet. Um dies zu klären bietet sich das totale Differential an.

$$df(x, t) = \frac{\partial f(x, t)}{\partial x} \cdot dx + \frac{\partial f(x, t)}{\partial y} \cdot dy \quad (7.49)$$

⁷Herausheben von h .

setzt man nun $y = t$ und dividiert durch dt ergibt sich folgendes.

$$\frac{df(x, t)}{dt} = \frac{\partial f(x, t)}{\partial x} \cdot \dot{x}(t) + \frac{\partial f(x, t)}{\partial t} \quad (7.50)$$

Mit

$$f(x, t) = \frac{dx(t)}{dt} = \dot{x}(t) \quad (7.51)$$

ergibt sich also

$$\frac{df(x, t)}{dt} = \frac{\partial f(x, t)}{\partial x} \cdot f(x, t) + \frac{\partial f(x, t)}{\partial t} = \dot{f}(x, t), \quad (7.52)$$

wobei der Teil rechts vom Gleichheitszeichen genau dem Klammerausdruck in (7.48) entspricht.

Daher lässt sich sich (7.43) mit der Umbenennung von $x(t+h) = x^{PC}(t+h)$ auch folgendermaßen beschreiben:

$$x^{PC}(t+h) \approx x(t) + h \cdot f(x, t) + h^2 \cdot \dot{f}(x, t) \quad (7.53)$$

Wichtig zu erkennen, dass man also über die Kombination eines Predictors und eines Korrektors eine Ableitung von f errechnet hat. Dadurch ist es nun möglich, durch geeignete Kombinationen von Predictor-Korrektor Schemas numerisches Integrationsverfahren mit höherer Ordnung herzuleiten.

Heun algorithm oder Runge Kutta 2. Ordnung

Vergleicht man (7.53) mit der unterhalb angeführten Taylorreihe, welche nach dem zweiten Term abgebrochen wurde

$$x_{t+h} \approx x_t + h \cdot f(x_t, t) + \frac{h^2}{2!} \cdot \dot{f}(x_t, t) \quad (7.54)$$

erkennt man eine große Ähnlichkeit.

Kombiniert man die Gleichung des Predictor Corrector (PC) Schemas mit einem Vorwärtseuler Schritt ergibt sich folgendes:

$$x^{FE}(t+h) = x_t + h \cdot f(x_t, t) \quad (7.55)$$

$$x^{PC}(t+h) = x_t + h \cdot f(x_t, t) + h^2 \cdot \dot{f}(x_t, t) \quad (7.56)$$

Addiert man diese beiden Gleichungen und multipliziert mit $1/2$ ergibt sich:

$$x^{Heun} \equiv \frac{1}{2} \cdot (x^{FE} + x^{PC}) = x_t + h \cdot f(x_t, t) + \frac{1}{2} \cdot h^2 \cdot \dot{f}(x_t, t) \quad (7.57)$$

Wird diese nun mit einem Faktor $1/2$ multipliziert erhält man exakt die in (7.54) aufgeführte Taylor Reihe.

Es ist nun also möglich über das Predictor-Corrector Verfahren die Taylor Reihe bis zu ihrem zweiten Term exakt abzubilden, ohne die Ableitung df/dt berechnen zu müssen. Eine Implementierung wäre folgendermaßen möglich:

Predictor:

$$\dot{x}_t = f(x, t) \quad (7.58)$$

$$x_{t+h}^P = x_t + h \cdot \dot{x}_t \quad (7.59)$$

Corrector:

$$\dot{x}_{t+h}^P = f(x_{t+h}^P, t + h) \quad (7.60)$$

$$x_{t+h}^C = x_t + h \cdot (\dot{x}_t + \dot{x}_{t+h}^P) \quad (7.61)$$

Dieses Verfahren wird *Heun Integration* genannt.

Allgemeine Herleitung von Runge Kutta Verfahren

Es ist möglich dieses Vorgehen zu verallgemeinern. Dabei werden die möglichen Gewichtungsfaktoren als Variablen durch die Berechnung gezogen und aus dem Vergleich mit der Taylor Reihe einer bestimmten Ordnung ergeben sich Bedingungen für die Wahl dieser Gewichtungsfaktoren. Diesem Vorgehen ergibt sich über

Predictor:

$$\dot{x}_t = f(x, t) \quad (7.62)$$

$$x_{t+h}^P = x_t + h \cdot \beta_{11} \cdot \dot{x}_t \quad (7.63)$$

Corrector:

$$\dot{x}_{t+h}^P = f(x_{t+h}^P, \alpha_1 t + h) \quad (7.64)$$

$$x_{t+h}^C = x_t + 0.5 \cdot h \cdot (\beta_{21} \cdot \dot{x}_t + \beta_{22} \cdot \dot{x}_{t+h}^P) \quad (7.65)$$

und die Entwicklung der Taylor Reihe daraus folgende Gleichung:

$$x_{t+h}^C = x(t) + h \cdot \beta_{21} + \beta_{22} \cdot f_t + \frac{h^2}{2} \cdot \left(2 \cdot \beta_{11} \cdot \beta_{22} \cdot \frac{\partial f_t}{\partial x} \cdot f_t + 2 \cdot \alpha_1 \cdot \beta_{22} \cdot \frac{\partial f_t}{\partial t} \right) \quad (7.66)$$

Vergleicht man dieses Ergebnis nun wieder mit (7.54) ergeben sich folgende Einschränkungen für die Wahl der Koeffizienten

$$\beta_{21} + \beta_{22} = 1 \quad (7.67)$$

$$2 \cdot \alpha_1 \cdot \beta_{22} = 1 \quad (7.68)$$

$$2 \cdot \beta_{11} \cdot \beta_{22} = 1 \quad (7.69)$$

$$(7.70)$$

Man erkennt schnell, dass es sich bei diesem Gleichungssystem um ein überbestimmtes mit vier Unbekannten und drei Gleichungen handelt. Daher sind grundsätzlich unendlich viele Lösungen möglich. Von diesen liefert jede einen Lösungsalgorithmus zweiter Ordnung. Da sich diese in der Genauigkeit der Approximation nicht unterscheiden bietet es sich an, möglichst viele der Koeffizienten null zu machen, um sich Rechenzeit zu sparen und die Simulation somit zu beschleunigen.

Vor allem für Algorithmen höherer Ordnung welches etwas später behandelt werden bietet es sich an, die Koeffizienten in einer Matrixschreibweise darzustellen. Deshalb auch die eigenwillige Indizierung der Faktoren α und β in der obigen Berechnung.

$$\alpha = \begin{pmatrix} 1 \\ 1 \end{pmatrix}; \quad \beta = \begin{pmatrix} 1 & 0 \\ 0.5 & 0.5 \end{pmatrix} \quad (7.71)$$

Gleichung (7.71) ist die Auflistung der Koeffizienten für den Heun Algorithmus, wobei der unterste Eintrag in der α Matrix immer eins ist, da dieser die Schrittweite des letzten Zwischenschrittes angibt und bei Einschrittverfahren daher immer auf h liegen muss.

Explizite Mittelpunkregel

Die einfachste Möglichkeit einen Integrationsalgorithmus zweiter Ordnung zu erstellen ist über folgende Koeffizienten möglich.

$$\alpha = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix}; \quad \beta = \begin{pmatrix} 0.5 & 0 \\ 0 & 1 \end{pmatrix} \quad (7.72)$$

Sie wird Mittelpunkregel genannt, weil sich der Zwischenschritt exakt in der Mitte zwischen den beiden Stützwerten t und $t + h$ befindet.

Runge Kutta mit Ordnung vier - RK4

Wird der verallgemeinerte Ansatz für die Findung von Integrationsalgorithmen erweitert, um mehrere Predictor Korrektor Schritte zu erlauben ergibt sich auch die Möglichkeit Algorithmen höherer Ordnung zu realisieren. Allerdings soll hier nicht wei-

ter auf die Herleitung eingegangen werden, sondern nur die bestehenden Algorithmen präsentiert werden. Diese sollten mit dem bisher vermittelten Wissen verständlich sein. Da es auch hier unendlich viele Lösungen für das überbestimmte Gleichungssystem gibt, werden nur die verbreitetsten Algorithmen präsentiert.

Der am weitesten verbreitete Algorithmus dieser Klasse ist über die folgenden Koeffizienten charakterisiert.

$$\alpha = \begin{pmatrix} 1/2 \\ 1 \\ 1/2 \\ 1/2 \end{pmatrix}; \quad \beta = \begin{pmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1/6 & 1/3 & 1/3 & 1/6 \end{pmatrix} \quad (7.73)$$

Dieser kann in vier Berechnungsstufen implementiert werden.

Stufe 0:

$$\dot{x}_t = f(x, t) \quad (7.74)$$

Stufe 1:

$$x^{P1} = x_t + \frac{h}{2} \cdot \dot{x}_t \quad (7.75)$$

$$\dot{x}^{P1} = f(x^{P1}, t + \frac{h}{2}) \quad (7.76)$$

Stufe 2:

$$x^{P2} = x_t + \frac{h}{2} \cdot \dot{x}_t^{P1} \quad (7.77)$$

$$\dot{x}^{P2} = f(x^{P2}, t + \frac{h}{2}) \quad (7.78)$$

Stufe 3:

$$x^{P3} = x_t + h \cdot \dot{x}_t^{P2} \quad (7.79)$$

$$\dot{x}^{P3} = f(x^{P3}, t + \frac{h}{2}) \quad (7.80)$$

Stufe 4:

$$x_{t+h} = x_k + \frac{h}{6} \cdot (\dot{x}_k + 2 \cdot \dot{x}^{P1} + 2 \cdot \dot{x}^{P2} + \dot{x}^{P3}) \quad (7.81)$$

Mittlerweile stehen auch Runge Kutta Algorithmen höherer Ordnung zur Verfügung. Die Berechnung der Koeffizienten für diese Verfahren wird äußerst aufwändig, daher wurden diese erst mit der Einführung von computerunterstützten analytischen Berechnungsprogrammen wie Maple oder Mathematica berechnet. Diese Algorithmen machen allerdings für mechatronische Aufgabenstellungen nur sehr selten Sinn. Diese Verfahren bis zur Ordnung acht und mehr werden dann beispielsweise in der Berechnung von Sternbewegungen eingesetzt. Als Daumenregel gilt in vielen Fällen: Ein Solver n-ter

Ordnung simuliert Systeme bis auf die n-te Nachkommastelle genau.

In [Abbildung 7.11](#) werden die Stabilitätsbereiche der Runge-Kutta Verfahren mit steigender Ordnung dargestellt. Man erkennt, dass dies mit steigender Ordnung größer werden. Mit einem Lösungsverfahren höherer Ordnung kann also eine größere Schrittweite gewählt werden. Daher ist es möglich, dass die Simulation mit einem Solver höherer Ordnung effizienter ist, als mit einem Verfahren niedrigerer Ordnung. Zusätzlich wird die Genauigkeit der Lösung verbessert, was mit der genaueren Approximation der analytischen Lösung begründet ist.

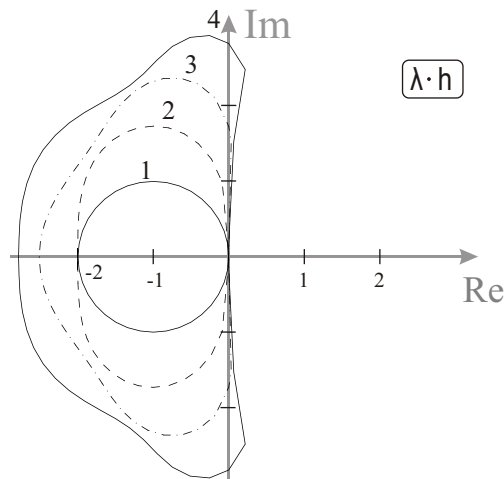


Abb. 7.11.: Stabilitätsbereiche verschiedener von Runge Kutta Verfahren mit steigender Ordnung

Dieses Verfahren kann wiederum ähnlich zu dem in [Code 7.1](#) gezeigten Programm implementiert werden. Dazu muss wiederum nur die Schleife gegen [Code 7.4](#) ausgetauscht werden.

```

1  for t = t_start:h:t_end
2      % RK4
3      dx_dt = A*x(i);
4      xP1 = x(i) + h/2 * dx_dt;
5      dxP1_dt = A*xP1;
6      xP2 = x(i) + h/2 * dxP1_dt;
7      dxP2_dt = A*xP2;
8      xP3 = x(i) + h*dxP2_dt;
9      dxP3_dt = A*xP3;
10     x(i+1) = x(i) + h/6*(dx_dt + 2*dxP1_dt + 2*dxP2_dt + dxP3_dt)
        ;
11
12     % for Plotting

```

```
13     t_vec(i) = t;  
14     i = i+1;  
15 end
```

Code 7.4: Implementierung eines Runge Kutta Integrationsalgorithmus vierter Ordnung mit den in Gleichung (7.73) angegebenen Koeffizienten

7.3.3. Rückwärtsinterpolation

Wie schon erwähnt gibt es für jede Ordnung von Integrationsalgorithmen unendlich viele mögliche Kombinationen von Koeffizienten. Dazu gibt es noch verschiedene Herangehensweisen an die Herleitung der Integrationsalgorithmen, welche teilweise speziell auf gegebene Systemeigenschaften eingehen. Ein Beispiel dafür sind Algorithmen welche ungedämpfte Systeme sehr genau simulieren können. Für diese speziellen Algorithmen wird der Leser auf weitere Literatur wie z.B. [CK06] verwiesen.

Eine relativ einfach zu erklärende Methode neue Lösungsalgorithmen zu schaffen ist die Rückwärtsinterpolation. Dabei werden bestehende Algorithmen verwendet wobei diese nicht wie üblich mit ansteigender Zeit rechnen, sondern rückwärts durch die Zeit. Dabei wird für den Wert von x_{k+1} ein Schätzwert angenommen und über eine negative Schrittweite $-h$ ein gewöhnlicher Integrationsalgorithmus (z.B. RK4) verwendet, um den Wert von \bar{x}_k zu berechnen. Über eine Iteration wird der Schätzwert von x_{k+1} so lange angepasst, bis der berechnete Wert \bar{x}_k mit dem vorhandenen Zustandswert x_k genau genug übereinstimmt.

Daraus resultiert eine Veränderung des Stabilitätsbereiches ähnlich der von Vorwärts- zu Rückwärts Euler, also eine Spiegelung an der imaginären Achse. Es können also auf diese Art explizite Algorithmen sehr einfach in implizite umgewandelt werden. Dabei können sogar große Teile der Implementierung beibehalten werden, da die einzige Änderung sich auf die im Falle der Rückwärtsinterpolation negative Schritte bezieht.

7.4. Mehrschrittverfahren

Die bisher vorgestellten Verfahren haben die nötigen Informationen zur Berechnung einer Lösung mit höherer Ordnung über mehrere Rechenschritte in einem Simulationsschritt erhalten. Dabei wurden diese „Zwischeninformationen“ nach jedem Rechenschritt verworfen und die nötigen Berechnungsstufen für jeden Schritt erneut durchgeführt. Es bietet sich hier eine potentiell deutlich effizientere Möglichkeit an. Es können die Informationen aus vergangenen Lösungspunkten verwendet werden, anstatt die für die Lösung unerheblichen Zwischenschritte zu berechnen. Dies ist das Grundprinzip auf dem alle Mehrschrittverfahren basieren. In diesem Kapitel soll allerdings keine ausführliche Herleitung der verschiedenen Verfahren erfolgen, sondern nur ein kurzer

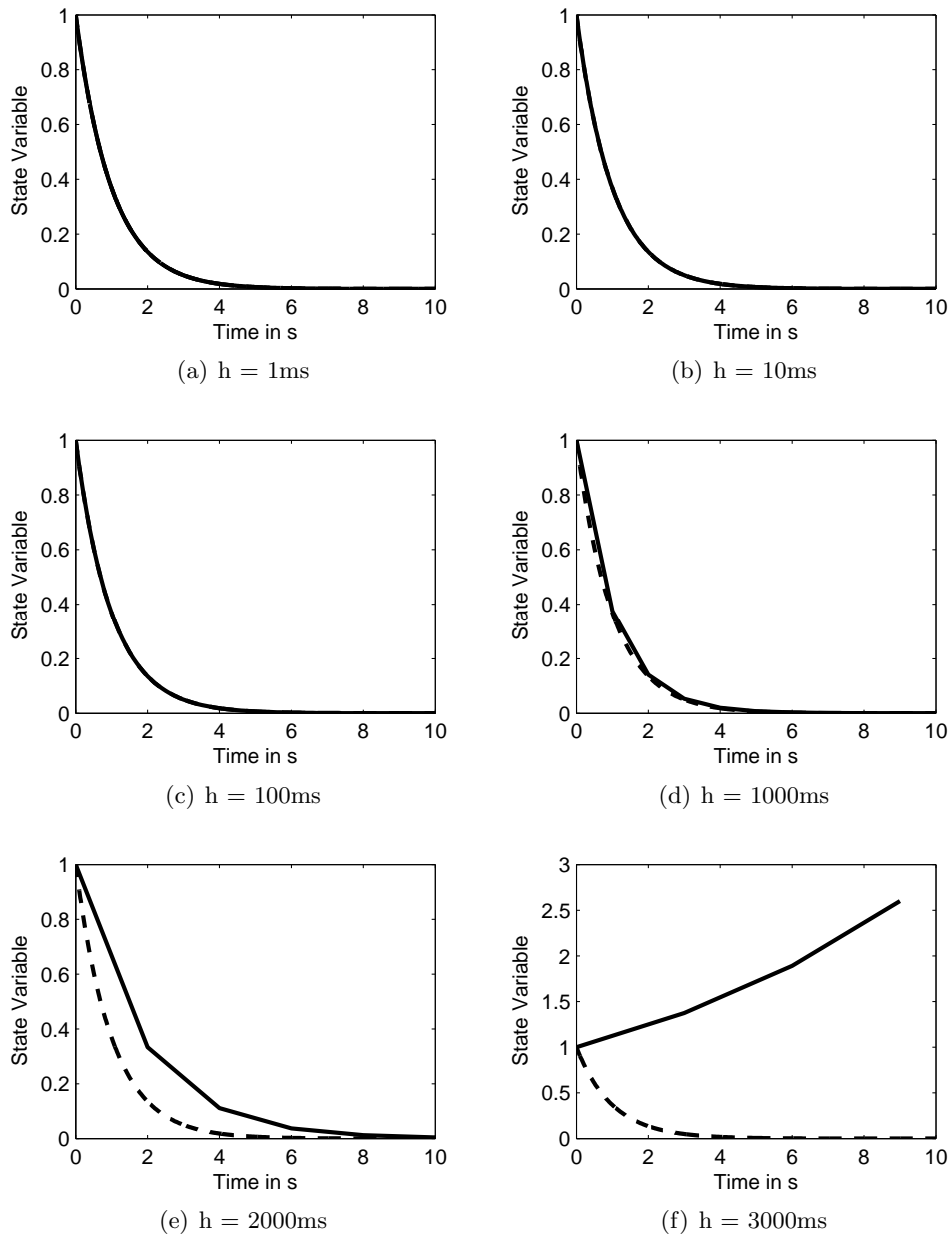


Abb. 7.12.: Der Einfluss der Schrittweite auf das Ergebnis bei einem Runge-Kutta Algorithmus vierter Ordnung.

Vergleich zu den Einschrittverfahren durchgeführt werden.

Grundsätzlich erfolgt die Herleitung dieser Klasse von Verfahren über Newton-Gregory Polynome. Diese wiederum setzen eine äquidistante Diskretisierung voraus. Es wird also für die Berechnung eine konstante Schrittweite h vorausgesetzt.

7.4.1. Adams-Bashforth (AB)

Bei den Adams-Bashforth Algorithmen handelt es sich um explizite Verfahren. Diese beruhen auf einer Extrapolation, welche versucht ein Funktion durch die vergangenen Punkte zu legen und so abzuschätzen wo die neue Lösung liegt. Mit steigender Ordnung werden mehr Punkte in der Vergangenheit verwendet und daher auch Funktionen höherer Ordnung verwendet. Da allerdings Polynome höherer Ordnung außerhalb des Bereiches in welchem die Funktion durch die bekannten Punkte gelegt wurden schnell ansteigen oder abfallen, werden die Verfahren mit höherer Ordnung zwar genauer, aber die schrittweite muss kleiner gewählt werden. Dies geht soweit, dass diese Verfahren über eine Ordnung von sechs grundsätzlich instabil sind.

7.4.2. Backward Difference Formulae (BDF)

Diese Klasse von Verfahren beruht auf einem Backward Newton-Gregory Polynom und ist ein implizites Verfahren. In der ersten Ordnung reduzieren sich diese Verfahren exakt auf einen Rückwärtsseuler. In den Ordnungen zwei bis fünf können diese Verfahren für die Simulation von technischen Systemen eingesetzt werden. BDF sechster Ordnung (BDF6) ist nur noch für Systeme ohne oszillatorischen Anteil (Chemie und Thermodynamik) geeignet. Auch Verfahren höherer Ordnung sind vorhanden, werden allerdings nicht mehr in technischen Simulationen eingesetzt. Trotzdem zählen Algorithmen dieser Klasse den zur Zeit am häufigst eingesetzten, weil sie in ihren niedrigeren Ordnungen sehr universell einsetzbar sind. Der Standardsolver von Dymola „Dassl“ ist beispielsweise ein BDF Algorithmus mit variabler Ordnung und variabler Schrittweite.

Die Gleichung für die Backward Difference Formulae der dritten Ordnung soll hier ohne eine weitere Herleitung aufgezeigt und kurz erläutert werden.

$$x_{t+h} = \frac{6}{11}h \cdot \dot{x}_{t+h} + \frac{18}{11}x_t - \frac{9}{11}x_{t-h} + \frac{2}{11}x_{t-2h} \quad (7.82)$$

Aus (7.82) lassen sich gut einige der zentralen Aspekte der Multistepverfahren erkennen. Deutlich wird dies durch die letzten beiden Terme mit x_{t-h} und x_{t-2h} . Diese greifen also auf Zustandsvariablen zurück welche in der Vergangenheit liegen. Der implizite Charakter des Verfahrens wird durch den ersten Term mit \dot{x}_{t+h} fixiert, weil weder Zustände noch Ableitungen der Zustände zu dem zukünftigen Zeitpunkt bekannt sind und daher iterativ bestimmt werden müssen. Verfahren höherer Ordnung verwenden

zusätzliche Punkte aus der Vergangenheit, um die erhöhte Ordnung zu erreichen.

7.4.3. Das Startup Problem

Alle Multistep Verfahren beruhen auf Informationen aus der Vergangenheit. Dies ist beim Start der Simulation ein Problem, da diese Punkte nicht vorhanden sind. Es gibt in aktuellen Implementationen von Lösungsalgorithmen zur Zeit grundsätzlich zwei Herangehensweisen an dieses Problem.

Der „erste Lösungsansatz“ steigert die Ordnung des Mehrschrittverfahrens mit jedem Simulationsschritt der berechnet wird. Es wird also für die Lösung des ersten Simulationsschrittes ein Verfahren erster Ordnung eingesetzt, also z.B. ein Rückwärtseuler. Für den zweiten Simulationsschritt kann dann beispielsweise ein BDF2, für den Folgenden ein BDF3 usw. eingesetzt werden bis die gewünschte Ordnung des Verfahrens erreicht ist. Problematisch dabei ist, dass die Schrittweite am Start dieser Simulation relativ klein gewählt werden muss, um die Genauigkeit der Lösung nicht negativ zu beeinflussen.

Der „zweite Ansatz“ ist es für die ersten Schritte Einschrittverfahren - meistens Runge Kutta Verfahren - einzusetzen. Auch diese haben den Nachteil, dass die Schrittweite geringe gewählt werden muss, um die Genauigkeitsforderungen erfüllen zu können. Dies ist vor allem bei steifen Systemen der Fall. Mehr dazu wird im folgenden Abschnitt erläutert.

7.5. Stabilität, Genauigkeit und steife Systeme

Betrachtet man die Schritte die zur Berechnung von expliziten und impliziten Verfahren nötig sind, ist die Berechnung von impliziten Verfahren wegen der nötigen Iteration deutlich komplexer. Es stellt sich also berechtigter Weise die Frage, wozu man solchen Aufwand betreiben soll. Die Antwort auf diese Frage liegt in der Berechnung sogenannter „steifer Systeme“. Diese Eigenschaft eines Systeme ist dadurch charakterisiert, dass ihre Eigenwerte weit über die linke Halbebene verstreut sind. Etwas genauer formuliert, weisen diese Systeme sehr große Verhältnisse zwischen den Realteilen der Eigenwerte auf. Es wird dazu in der Literatur kein genaues Verhältnis von größtem zu kleinstem Eigenwert angegeben. Liegt dieses allerdings über einem Bereich von 100-1000 oder darüber, kann mit Sicherheit von einem steifen System ausgegangen werden. Physikalisch interpretiert hat das System also sehr schnelle und sehr langsame Anteile (oder Zeitkonstanten), welche zu berechnen sind. Dies ist etwa der Fall wenn eine elektronische Schaltung inklusive deren thermischen Verhaltens berechnet wird. Die Zeitkonstanten der elektronischen Komponenten liegen oft im Bereich von Millisekunden, während sich für den thermischen Teil Zeitkonstanten im Bereich von einigen Sekunden bis zu Stunden üblich sind.

Betrachtet man nun nochmals den Stabilitätsbereich des Vorwärtseulerverfahrens (Abbildung 7.4 auf Seite 225) oder die Bereiche der Runge Kutta Verfahren (Abbildung 7.11 auf Seite 241), muss die Schrittweite h so gewählt werden, dass alle Eigenwerte in den stabilen Bereich der $\lambda \cdot h$ Ebene verschoben werden. Es bestimmt also der schnellste oder am weitesten links gelegene Eigenwert des Modells die Schrittweite des Lösungsverfahrens. Im Normalfall bestimmen die langsamen Komponenten im System die nötigen Simulationszeiten, um z.B. die Erwärmung einer elektronischen Schaltung bestimmen zu können. Es muss also über eine lange Simulationszeit mit einer verhältnismäßig kleinen Schrittweite gerechnet werden. Diese Tatsache sorgt für sehr lange Berechnungszeiten der Simulation. Das einfache Lösen eines steifen Systems stellt also kein großes Problem dar. Dazu muss nur die Schrittweite klein genug gewählt werden. Das System allerdings in einer sinnvollen Zeit zu berechnen erfordert richtig gewählte und konfigurierte Integrationsverfahren.

Bei impliziten Verfahren wie dem Rückwärtseulerverfahren (BE) oder der Backward Difference Formula (BDF), welche in ihrer ersten Ordnung dem Rückwärtseuler entspricht, ergibt sich eine grundsätzlich andere Stabilitätsdomäne (Abbildung 7.8 auf Seite 230). Diese hat die Eigenschaft, dass der numerisch stabile Bereich die gesamte linke Halbebene umfasst. Daher werden unabhängig von der Schrittweite alle Eigenwerte stabil berechnet. Es kann daher eine deutlich größere Schrittweite gewählt werden. Da es sich bei technischen Systemen - zumindest wenn diese ein gewisses Ausmaß erreichen - praktisch immer um steife Systeme handelt, sind implizite Lösungsverfahren oft als Standardsolver eingesetzt, trotz des erhöhten Rechenaufwands.

Neben der numerischen Stabilität der Berechnung ist natürlich auch die Genauigkeit der errechneten Lösung von zentraler Bedeutung. Für eine genau Simulation reicht es nicht aus, knapp an die Grenze der Stabilität zu gehen. Dies soll durch Abbildung 7.6, 7.9 und 7.12 verdeutlicht werden.

Die Beschreibung der Genauigkeitseigenschaften einer Lösung ist deutlich komplexer als die der Stabilität. Es soll daher hier auf diese Herleitung verzichtet werden. Der interessierte Leser wird auf [CK06] verwiesen, wo das Konzept der Accuracy Domain erläutert wird. Es soll aber klargestellt sein, dass Stabilität zwar ein notwendiges, aber kein hinreichendes Kriterium für eine genaue Lösung ist. Die Schrittweite für eine genaue Simulation muss deutlich kleiner gewählt werden, als es für die Stabilität nötig ist. Ein Faktor 10-100 muss hier gewählt werden. Es macht dabei Sinn die Schrittweite fortlaufend zu verringern und dann die größte Schrittweite zu wählen, bei der sich keine signifikante Änderung mehr ergibt.

7.5.1. Ungedämpfte Systeme

Eine zwar verständliche, aber nicht offensichtliche Eigenschaft von numerischen Lösungsverfahren stellt sich bei der Simulation von ungedämpften Systemen heraus. Simuliert man z.B. ein mathematisches Pendel wird sich dieses bei der Simulation

mit einem Rückwärtseuler wie ein leicht gedämpftes Pendel verhalten. Simuliert man hingegen mit einem Rückwärtseuler wird das System eine langsam aber exponentiell ansteigende Schwingung vollziehen. Dieses Verhalten lässt sich mit den verschiedenen Stabilitätsdomänen der beiden Solver erklären. Die Pole eines ungedämpften Systems liegen direkt auf der imaginären Achse und somit in der analytischen Lösung an der Grenze des Stabilitätsbereichs (Abbildung 7.3). Beim Rückwärtseuler liegen die Pole genau so auf der imaginären Achse und somit in der stabilen Region (Abbildung 7.8). Daher klingt die numerische Lösung mit der Zeit ab. Für den Vorwärtseuler liegen die Eigenwerte im instabilen Bereich, was die instabile Lösung und somit das Aufschwingen erklärt (Abbildung 7.4).

Es gibt für diese Problemstellungen eigens erstellte Lösungsverfahren genau für diese „velocity free“ Modelle erstellt sind. Es soll allerdings zu diesem Punkt nicht weiter auf diese eingegangen werden, da es sich um einen nicht zu häufig vorkommenden Spezialfall handelt.

7.6. Schrittweitensteuerung

Die Schrittweitensteuerung ist neben der Anpassung der Ordnung eine Variante das Abwägen zwischen Geschwindigkeit und Genauigkeit zu ermöglichen. Sind z.B. die schnellen Einschwingvorgänge eines Systems abgeschlossen, kann die Schrittweite vergrößert, bzw. die Ordnung des Lösungsalgorithmus verkleinert werden, um die Simulation zu beschleunigen. Bei den Runge-Kutta Algorithmen ist es üblicher nur die Schrittweite zu steuern. Dabei wird oft so vorgegangen, dass zwei verschiedene Integrationsalgorithmen zu Berechnung eines Schrittes verwendet werden. Nach der Berechnung werden die beiden vorliegenden Ergebnisse verglichen und sofern der Unterschied zwischen den beiden Lösungen nicht zu groß ist, wird dieser Schritt akzeptiert. Wird eine gewisse Fehlergrenze unterschritten, kann die Schrittweite vergrößert werden. Wird eine andere Fehlergrenze überschritten wird die Schrittweite verkleinert.

Natürlich gibt es zu diesem Gebiet deutlich ausgefeiltere Ansätze, welche z.B. regelungstechnische Ansätze einsetzen, um die Schrittweite der aufgrund des auftretenden Fehlers zu regeln. Hier soll aber nicht weiter auf die detaillierte Umsetzung der Schrittweitensteuerung eingegangen werden.

Trotzdem ist es äußerst ineffizient nur für die Schrittweitensteuerung einen nahezu verdoppelten Aufwand durch die zweifache Berechnung des Integrationsschrittes in Kauf zu nehmen. Daher wurde mit dem Runge-Kutta-Fehlberg 4/5 eine spezielle Variante des Runge-Kutta Algorithmus entwickelt, welche gleichzeitig eine Lösung vierter und fünfter Ordnung berechnet. Der Aufwand ist daher kaum höher als für die Lösung eines Solvers fünfter Ordnung, gleichzeitig bekommt man aber die Möglichkeit der Schrittweitensteuerung praktisch geschenkt.

Ein grundsätzlicher Unterschied zwischen verschiedenen Schrittweitensteuerungen soll aber trotz der Kompaktheit dieses Abschnitts kurz erwähnt werden. Es gibt sogenannte optimistische und pessimistische Schrittweitensteuerungen. Bei den pessimistischen wird nach einer Überschreitung der Fehlergrenze der zuvor berechnete Schritt verworfen und mit einer reduzierten Schrittweite wiederholt. Bei optimistischen Verfahren wird der vorangegangene Schritt trotz einer zu großen Abweichung akzeptiert und die Schrittweite erst für den folgenden Schritt reduziert. Hier sind wiederum die verschiedenen Anforderungen an die Simulationsumgebung, wie z.B. die Echtzeitfähigkeit ein Grund für die unterschiedlichen Ansätze.

7.6.1. Schrittweitensteuerung und Mehrschrittverfahren

Bei Mehrschrittverfahren ist die Schrittweitensteuerung deutlich komplizierter als bei den Einschrittverfahren. Dies ist dadurch begründet, dass in der Herleitung der Mehrschrittverfahren von einer äquidistanten zeitlichen Diskretisierung ausgegangen wird. Dies ist bei einer Schrittweitensteuerung nicht mehr gegeben. Daher muss hier eine Möglichkeit gefunden werden, die vorhandenen Datenpunkte in die neue Schrittweite umzurechnen. Dies sollte mit einer ausreichenden Genauigkeit (also Ordnung) geschehen, um durch die Schrittweitensteuerung keine Abstriche in der Genauigkeit machen zu müssen.

Dieses Problem wird bei den Mehrschrittverfahren über den „Nordsieck Vektor“ gelöst. Es soll hier allerdings wieder nicht genauer auf die Berechnung eingegangen werden. Wichtig zu wissen ist, dass durch die relativ aufwändige Umrechnung der vergangenen Werte auf die neue Schrittweite der Aufwand für die Schrittweitensteuerung deutlich höher ist als bei Einschrittverfahren.

7.6.2. Dense Output

Problematisch bei der Simulation mit variablen Schrittweiten ist, dass für die Ausgabe im Normalfall eine gewisse Anzahl von Samplewerten mit gleichem Abstand, oder direkt ein Sampleintervall vorgegeben wird. Da es für Mehrschrittverfahren rechentechnisch sehr aufwändig ist, die Schrittweite anzupassen, um zu den vorgesehenen Kommunikationsintervallen⁸ Werte zu errechnen, ist es effizienter die Schrittweite nicht anzupassen um jedes Kommunikationsintervall zu treffen, sondern diese Stützpunkte aus den tatsächlich simulierten Punkten zu errechnen. Auch hier findet wieder der Nordsieck Vektor Verwendung, da dieser eine Lösung dieses Problems mit derselben Ordnung wie die des eigentlichen Lösungsalgorithmus ermöglicht und rechentechnisch recht effizient ist.

⁸Diese errechneten Werte werden gespeichert und dem User angezeigt, auch wenn der Integrationalgorithmus deutlich mehr oder weniger Punkte berechnet.

Für Einschrittalgorithmen wird meist einfach ein Simulationsschritt an der Stelle des Kommunikationszeitpunktes gesetzt. Der Solver errechnet die Lösung zu diesem Zeitpunkt und fährt danach fort wie bisher. Alternativ wird ein Verfahren eingesetzt, das ähnlich wie der Runge-Kutta-Fehlberg Algorithmus mehrere (in diesem Fall drei) Algorithmen verschiedener Ordnung einsetzt. Damit ergeben sich für jeden Simulationsschritt drei Lösungen für den approximierten Wert, woraus mit einer passenden Ordnung auf einen beliebigen Zwischenwert geschlossen werden kann.

7.6.3. Radau IIA Algorithmus

Wie bereits in [Unterabschnitt 7.6.1](#) erwähnt, ist eine Schrittweitensteuerung eines Mehrschrittverfahrens relativ rechenaufwändig, da vergangene Punkte jeweils mit einem äquidistanten Abtastintervall vorliegen müssen. Dies ist ein grundsätzlicher Nachteil von Mehrschrittverfahren und wird auch von der Umformulierung in einen DAE Solver, wie das im letzten Abschnitt passiert ist nicht umgangen. Grundsätzlich ist es also so, dass Mehrschrittverfahren bei der Simulation von sehr stark nichtlinearen Systemen und Systemen mit vielen Events (wie das in jedem teildiskreten System der Fall ist - dazu in [Abschnitt 7.7](#) mehr), sehr ineffizient werden.

Eine Möglichkeit diesen Nachteil zu umgehen bieten implizite Runge Kutta Algorithmen wie der Radau IIA. Von diesem stehen bereits stabile Implementationen zur Verfügung, allerdings werden sie noch sehr selten als Standardsolver in den Modellierungsumgebungen verwendet. Der große Vorteil dieser Solver-Klasse besteht darin, dass sie für die Simulation steifer Systeme geeignet sind und zusätzlich ist hier eine Schrittweitensteuerung nicht mit übermäßigem Aufwand verbunden. Dieser Vorteil steht nun einem grundsätzlichen Vorteil der Mehrschrittverfahren gegenüber, dass diese keine Zwischenschritte berechnen müssen und diese daher effizienter sind. Je nach Anwendung kann nun einer der beiden Vorteile überwiegen. Grundsätzlich kann also gesagt werden, dass es für einfache Systeme bei aktuellen Rechenleistungen keinen merkbaren Unterschied macht, welcher Solver für die Simulation eingesetzt wird⁹. Bei großen, steifen Systemen welche mäßige Nichtlinearitäten beinhalten wird DASSL höchstwahrscheinlich die beste Performance liefern. Werden die Systeme sehr nichtlinear und haben viele diskrete Events zu bearbeiten ist Radau IIA höchstwahrscheinlich die beste Wahl.

⁹DASSL ist für die Simulation kleiner linearer Systeme auch nicht die optimale Wahl. Da diese allerdings im Normalfall in Zeitbereichen von Milisekunden berechnet werden, wurde auf eine Optimierung seitens Dymola hier verzichtet und dafür eine gute Performance bei großen Systemen mit den Standardeinstellungen ermöglicht.

7.6.4. Inline Integration

Bei dieser Methodik werden die Modellgleichungen direkt mit dem Lösungsalgorithmus verbunden. Bettet man z.B. einen Rückwärtseuler in die Gleichungen ein welche sich aus [Abbildung 6.16](#) ergeben, erhält man folgenden Zusammenhang.

$$u = 30V \quad (7.83)$$

$$u_L - L \cdot di_L = 0 \quad (7.84)$$

$$i_C - C \cdot du_C = 0 \quad (7.85)$$

$$u_{R_1} - i \cdot R_1 = 0 \quad (7.86)$$

$$u_{R_2} - i_{R_2} \cdot R_2 = 0 \quad (7.87)$$

$$i - i_{R_1} = 0 \quad (7.88)$$

$$i_C - i_{R_2} = 0 \quad (7.89)$$

$$i - i_L - i_{R_2} = 0 \quad (7.90)$$

$$u - u_{R_1} - u_L = 0 \quad (7.91)$$

$$u_L - u_{R_2} - u_C = 0 \quad (7.92)$$

$$i_L - i_L(t-h) - h \cdot di_L = 0 \quad (7.93)$$

$$u_C - u_C(t-h) - h \cdot du_C = 0 \quad (7.94)$$

Dabei ist es wichtig zu erkennen, dass nun nicht mehr aus den aktuellen Werten der Zustandsvariablen deren Ableitungen berechnet werden aus welchen der Solver dann die neuen Zustandsvariablen bestimmt. In diesem Fall werden aus den vergangenen Zuständen die (gekennzeichnet mit „t-h“) direkt die aktuellen Zustände berechnet. Da die Zustandsvariablen nun in zusätzlichen Gleichungen aufscheinen, kann es passieren, dass strukturelle Singularitäten, welche ohne die Inline Integration auftreten, vermieden werden. Allerdings wurde die Inline Integration nach einigen Jahren der Forschung für die Simulation nicht mehr weiter in Betracht gezogen.

7.7. Unstetigkeiten

Viele technische Systeme enthalten Diskontinuitäten. Diese können z.B. diskrete Teilsysteme eines Modells, schaltende Elemente in einer elektronischen Schaltung, Festkörperreibung in der Mechanik oder viele andere übliche Elemente enthalten. Grundsätzlich stellen diese Diskontinuitäten alle der bisher vorgestellten Lösungsalgorithmen vor unlösbare Aufgabe, da alle diese Lösungsalgorithmen über die Taylorreihe und somit eine Polynomreihe funktionieren. Diese sind - sofern nicht unendlich lang - nicht in der Lage Diskontinuitäten genau zu beschreiben.

Kommt ein schrittweitengesteuerter Lösungsalgorithmus in die Situation eine Diskon-

tinuität beschreiben zu müssen, wirkt die sehr schnelle Änderung der Variable auf den Solver wie eine sehr schnelle Polstelle. Der Solver reduziert die Schrittweite so weit bis entweder die untere Integrationsschrittweite erreicht wird, oder die höheren Terme der Taylor Reihe durch die kleine Schrittweite h so klein werden, dass sich beide Verfahren praktisch auf das selbe Verfahren reduzieren. Nach dem Auftreten der Diskontinuität kann der Solver die Schrittweite wieder vergrößern. Die Schrittweite nähert sich also langsam wieder dem optimalen Wert an. Dieses Verhalten ist weder effizient noch ist eine genaue Lösung sichergestellt. Es sollte also eine bessere Alternative gefunden werden.

Betrachtet man nun die Arten von diskreten Events, welche eine solche Diskontinuität auslösen können gibt es zwei Arten.

- Zeitgesteuerte Events, welche immer zu einem vorbestimmten Zeitpunkt ausgelöst werden. Dazu zählen z.B. das Zünden eines Thyristors, der mit einer bestimmten Ansteuerung versehen ist, oder einem diskreten Teil einer Simulation, wenn also z.B. ein digitaler Regler in Kombination mit einem kontinuierlichen System simuliert wird.
- Zustandgesteuerte Events, welche von einem Zustand abhängig ausgelöst werden. So wird z.B. ein Event generiert wenn der Thyristor einen Stromnulldurchgang hat (Stromabhängigkeit), oder wenn ein Körper zu rutschen beginnt (Geschwindigkeitsabhängigkeit).

Das Finden von zeitgesteuerten Events ist sehr einfach. Sie können schon vor der eigentlichen Simulation in eine Liste eingetragen werden und die Zeitpunkte dafür sind bekannt. Die Lokalisierung von zustandsgesteuerten Events ist deutlich komplizierter. Dazu werden im Normalfall Nulldurchgangserkennungs-Algorithmen (engl. Zero Crossing Detection) eingesetzt um den Zeitpunkt möglichst genau festzustellen. Dazu bieten sich Algorithmen wie Regula Falsi, Golden Section oder Newton Iteration an. Es soll aber hier nicht weiter auf diese Algorithmen eingegangen werden.

Tritt ein solches Event ein, wird folgendermaßen verfahren. Der Lösungsalgorithmus berechnet direkt vor dem Auftreten eines diskreten Events eine Lösung. Dann wird das Event abgearbeitet, was in manchen Fällen zu einem veränderten Systemverhalten führen kann. Anschließend müssen neue Initialbedingungen gefunden werden und die Simulation kann dann von einem Zeitpunkt direkt nach dem Event fortgesetzt werden. Großer Unterschied zu der Variante ohne ein explizites Event Handling ist, dass vor und nach dem Event mit einer großen Schrittweite gerechnet werden kann. Daher sind diese Simulationen deutlich effizienter und sicherer. Jede gute Modellierungsumgebung sollte also eine Möglichkeit haben ein Eventhandling für solche hybriden Systeme (diskrete Teile in Kombination mit kontinuierlichen Teilen) haben. Dazu ist es nötig einen Solver zu verwenden welcher einen Dense Output (vgl. [Unterabschnitt 7.6.2](#)) zu verwenden.

7.8. Faustregeln bei der Wahl von Solver und Schrittweite

Dieser Abschnitt soll einen groben Überblick über mögliche Zusammenhänge von Schrittweite, Solverart und Fehlergröße geben. Dabei handelt es sich um sehr grobe Faustregeln, welche aber ein erstes Indiz dafür liefern können, welche Anpassung sinnvoll sein könnte.

Grundsätzlich muss dafür gesorgt werden, dass die Simulation stabil ist, und Kriterien der Genauigkeit erfüllt werden. Dafür eine generell gültige Regel zu geben ist praktisch nicht möglich. Daher wird die passende Schrittweite am sinnvollsten über Versuche bestimmt. Dabei macht es Sinn die selbe Simulation mehrmals durchzuführen und dabei die Schrittweite zu verkleinern. Wenn die Änderung im Ergebnis deutlich kleiner ist, als die geforderte Genauigkeit kann diese Schrittweite als sinnvoll angesehen werden. Ist eine Toleranz von 10^{-410} gewählt. Darf sich die vierte Stelle hinter dem Komma nicht mehr ändern.

Die Schrittweite geht direkt in den Fehler ein. Wird die Schrittweite verkleinert sinkt üblicherweise auch der Fehler. Wenn die Schrittweite eines Solvers der Ordnung n um einen Faktor 10 verkleinert wird, verkleinert sich der globale Fehler um 10^n .

Für die Wahl der Solver in Dymola kann von folgenden Zusammenhängen ausgegangen werden.

1. Dassl: Ist ein sehr robustes Integrationsverfahren für die so gut wie alle Modelle. Im Prinzip muss über einen anderen Solver nur nachgedacht werden, wenn Dassl zu langsam erscheint.
2. Radau: Deutlich effizienter für schwach gedämpfte Systeme als Dassl.
3. Dopri45: Viel effizienter als Dassl für nicht-steife Abtastsysteme mit sehr kleiner Abtastzeit. Für steife Systeme mit kleiner Abtastzeit Radau versuchen.

Für eine Echtzeitsimulation werden grundsätzlich Solver gewählt, welche eine Ausführungszeit abschätzen lassen. Sie dürfen also keine Iteration aufweisen, also fallen implizite Solver grundsätzlich weg. Oft werden sehr einfache Solver für die Simulation verwendet.

7.9. Quantized State Simulation

Über die in diesem Kapitel mehrmals erwähnte Schrittweite h wird der zeitliche Verlauf der Simulationsgrößen diskretisiert. Dies ist zwar bei weitem die gängigste Methode der Diskretisierung, es gibt aber auch andere Ansätze. Im Unterschied zu allen bisher aufgezeigten Verfahren wird hier nicht zeitliche Achse diskretisiert, sondern ein Quat-

¹⁰Eine für technische Systeme bereits sehr kleine Toleranz, 10^{-3} reicht hier üblicherweise aus.

num definiert in welchem sich die Zustände ändern dürfen, bevor neu simuliert wird. Über verschiedene Verfahren wird abgeschätzt wann einer der Zustände diesen „Toleranzbereich“ über oder unterschreitet und zu diesem Zeitpunkt wird neu simuliert. Die Verfahren sind allerdings noch in Entwicklung und werden noch von keinem kommerziellen Simulationsprogramm eingesetzt. Daher werden sie hier auch nicht weiter behandelt. Diese QSS (Quantized State Simulation) Verfahren werden z.B. in [CK06] weiter erläutert.

A. Systeme linearer Differentialgleichungen 1. Ordnung mit konstanten Koeffizienten

Der Zustandsraumdarstellung wurde bereits ein eigenes Kapitel „*Einführung in die Zustandsraumdarstellung*“ gewidmet. Nichts desto trotz soll den Lesern an dieser Stelle ein alternativer Zugang verschafft werden.

Abbildung A.1 zeigt eine elektrische Schaltung bestehend aus zwei Induktivitäten und zwei Widerständen. Durch Anwenden der Kirchhoff'schen Maschenregel, welche besagt,

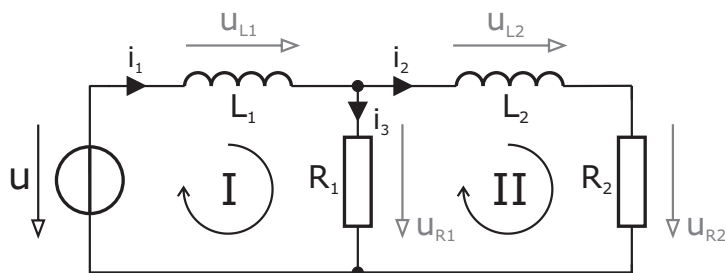


Abb. A.1.: Elektrische Schaltung

dass die Summe aller Spannungen gleich Null sein muss, erhalten wir aus den Maschen I und II folgende zwei Gleichungen:

$$u = u_{L_1} + u_{R_1} \tag{A.1}$$

$$u_{R_1} = u_{L_2} + u_{R_2} \tag{A.2}$$

Durch Substituieren der Spannungen mit den dazugehörigen Komponentengleichungen erhalten wir:

$$u = L_1 \cdot \frac{di_1}{dt} + i_3 \cdot R_1 \tag{A.3}$$

$$i_3 \cdot R_1 = L_2 \cdot \frac{di_2}{dt} + i_2 \cdot R_2 \tag{A.4}$$

Formt man obige Gleichungen so um, dass die zeitlichen Ableitungen der Ströme durch die Induktivitäten auf der linken Seite stehen, erhält man:

$$\frac{di_1}{dt} = \frac{1}{L_1} \cdot u - \frac{1}{L_1} \cdot R_1 \cdot i_3 \quad (\text{A.5})$$

$$\frac{di_2}{dt} = \frac{1}{L_2} \cdot R_1 \cdot i_3 - \frac{1}{L_2} \cdot R_2 \cdot i_2 \quad (\text{A.6})$$

Das physikalische System, sprich die elektrische Schaltung, kann demnach mit 2 *Differentialgleichungen 1. Ordnung* beschrieben werden. Genauer gesagt handelt es sich bei den Differentialgleichungen um Differentialgleichungen 1. Ordnung mit konstanten Koeffizienten, wobei Gleichung (A.5) inhomogener und Gleichung (A.6) homogener Natur ist. Die Differentialgleichungen sind zusätzlich noch miteinander verkoppelt. Dies ist anhand des Widerstands R_1 ersichtlich, dessen Spannung $u_{R_1} = R_1 \cdot i_3$ in beiden Gleichungen vorkommt.

In einer allgemeinen Form lässt sich unser System wie folgt darstellen:

$$\dot{x}_1 = a_{11} \cdot x_1 + a_{12} \cdot x_2 + z_1(t) \quad (\text{A.7})$$

$$\dot{x}_2 = a_{21} \cdot x_1 + a_{22} \cdot x_2 + z_2(t) \quad (\text{A.8})$$

Wobei die Ströme i_1 und i_2 , welche den Zustand des System beschreiben, und daher fortan als Zustandsgrößen bezeichnet werden, dabei mit x_1 bzw. mit x_2 ersetzt werden. Allgemein gilt, dass Zustandsgrößen mit x bezeichnet werden und die Ausgangsgrößen energiespeichernder Elemente sind (siehe [Unterabschnitt 2.2.2](#)). Die konstanten Koeffizienten werden mit $a_{11} \dots a_{22}$ beschrieben. Die Funktionen $z_1(t)$ sowie $z_2(t)$ werden als Störgrößen charakterisiert und später noch etwas präziser formuliert.

Um das Gleichungssystem (A.5) und (A.6) in diese allgemeine Form zu bringen muss der Strom i_3 welcher keine Zustandsgröße repräsentiert mittels Zustandsgrößen ausgedrückt werden. Dies wird mittels der Kirchhoff'schen Knotenregel erreicht:

$$\begin{aligned} i_1 &= i_2 + i_3 \\ \Rightarrow i_3 &= i_1 - i_2 \end{aligned} \quad (\text{A.9})$$

Die Gleichungen (A.5) und (A.6) nehmen dadurch folgende Form an:

$$\frac{di_1}{dt} = \frac{1}{L_1} \cdot u - \frac{1}{L} \cdot R_1 \cdot (i_1 - i_2) \quad (\text{A.10})$$

$$\frac{di_2}{dt} = \frac{1}{L_2} \cdot R_1 \cdot (i_1 - i_2) - \frac{1}{L_2} \cdot R_2 \cdot i_2 \quad (\text{A.11})$$

bzw. in einer strukturierteren Schreibweise, welche der allgemeinen Form aus den Glei-

chungen (A.7) und (A.8) entspricht.

$$\frac{di_1}{dt} = -\frac{R_1}{L_1} \cdot i_1 + \frac{R_1}{L_1} \cdot i_2 + \frac{1}{L_1} \cdot u \quad (\text{A.12})$$

$$\frac{di_2}{dt} = \frac{R_1}{L_2} \cdot i_1 - \frac{R_1 + R_2}{L_2} \cdot i_2 \quad (\text{A.13})$$

Betrachtet man nun den konstanten Ausdruck $-R_1/L_1$ in Gleichung (A.12), welcher mit dem Strom $i_1 = x_1$ multipliziert wird, so ist gut ersichtlich, dass dieser dem konstanten Koeffizient a_{11} in Gleichung (A.7) entspricht.

Matrizendarstellung von Systemen linearer Differentialgleichungen 1. Ordnung mit konstanten Koeffizienten

Die Gleichungen (A.7) und (A.8) lassen sich sehr kompakt zu einer Matrixgleichung zusammenfassen.

$$\dot{\underline{x}} = \underline{A} \cdot \underline{x} + \underline{z}(t) \quad (\text{A.14})$$

wobei

$$\underline{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad (\text{A.15})$$

und

$$\underline{z}(t) = \begin{pmatrix} z_1(t) \\ z_2(t) \end{pmatrix} \quad (\text{A.16})$$

Übergang zur Zustandsraumdarstellung

Die kompakte Gleichung (A.14) wird als *Zustandsgleichung* bezeichnet. Eine exakte Interpretation dieser Gleichung erfolgte bereits in [Unterabschnitt 2.3.3](#). Wie wir bereits aus [Abschnitt 2.1](#) wissen, werden technische Systeme mittels Eingangsvariablen $\underline{u}(t)$, Ausgangsvariablen $\underline{y}(t)$ und Zustandsvariablen $\underline{x}(t)$ beschrieben.

In Gleichung (A.14) ist allerdings noch nicht ersichtlich, dass es Eingangs- bzw. Ausgangsgrößen gibt. Die Eingangsgröße (elektrische Spannung u) ist zwar prinzipiell schon enthalten, jedoch wird diese noch durch den Störgrößenvektors $\underline{z}(t)$ ausgedrückt. Diese Bezeichnung rührt daher, dass es sich bei Systemen welche störgrößenbehaftet sind, um fremderregte Systeme handelt. Solche Systeme werden von außen erregt bzw. beeinflusst. Um dieses Verhalten besser beschreiben zu können, drücken wir den Störvektor

$\underline{z}(t)$ aus Gleichung (A.16) fortan wie folgt aus:

$$\underline{z}(t) = \underline{b} \cdot u(t) \quad (\text{A.17})$$

wobei

$$\underline{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad (\text{A.18})$$

ist. Gleichung (A.14) nimmt daher folgende Form an.

$$\dot{\underline{x}} = \underline{A} \cdot \underline{x} + \underline{b} \cdot u \quad (\text{A.19})$$

Bezogen auf unser System, welches durch die Gleichungen (A.12) und (A.13) beschrieben wird, erhalten wir folgende Zustandsgleichung:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{R_1}{L_1} & \frac{R_1}{L_1} \\ \frac{R_1}{L_2} & -\frac{R_1+R_2}{L_2} \end{pmatrix}}_A \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} \frac{1}{L_1} \\ 0 \end{pmatrix}}_b \cdot u \quad (\text{A.20})$$

Nun fehlt uns noch die Ausgangsgröße. Diese wird durch eine zusätzliche Gleichung definiert.

$$y = \underline{c}^T \cdot \underline{x} \quad (\text{A.21})$$

Diese Gleichung beschreibt den Zusammenhang zwischen Ausgangsgrößen $y(t)$ und Zustandsvariablen $\underline{x}(t)$. Dafür wird der Zeilenvektor \underline{c}^T eingeführt¹. Je nachdem an welcher Stelle im Vektor ein Eintrag steht, wird dadurch beschrieben welche Zustandsgröße als Ausgangsgröße fungiert. Wählen wir z.B. den Strom $i_1 = x_1$ als Ausgangsvariable, ergibt sich folgender Zusammenhang:

$$y = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{\underline{c}^T} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad (\text{A.22})$$

Gelegentlich wird die Ausgangsgleichung noch um einen zusätzlichen Term erweitert, welcher dafür sorgt, dass Eingangssignale einen direkten, d.h. proportionalen Einfluss auf den Ausgang haben. Mehr dazu in [Abschnitt 2.3](#).

¹In der Mathematik werden Vektoren als Spaltenvektoren angegeben (die Elemente stehen also untereinander). Der Zeilenvektor \underline{c}^T ist also ein transponierter Spaltenvektor

Literaturverzeichnis

- [And09] Markus Andres. Object-oriented modeling of wheels and tires in dymola/-modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.
- [Ass10] Modelica Association. Modelica 3.2 language specification. https://www.modelica.org/news_items/documents/ModelicaSpec32.pdf, 2010. date: 17.08.2011.
- [Bor10] W. Borutzky. *Bond Graph Methodology, Development and Analysis of Multidisciplinary Dynamic Systems Models*. Springer, 2010.
- [Bro99] J.F. Broenink. 20-sim software for hierarchical bond-graph/block-diagram models. In *Simulation Practice and Theory*, pages 7(5–6):481–492, 1999.
- [CE93] F. E. Cellier and H. Elmqvist. Automated formula manipulation supports object-oriented continuous-system modeling. In *IEEE Control Systems*, 1993.
- [Cel79] F.E. Cellier. *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*. PhD thesis, ETH Zürich, Switzerland, 1979. Diss ETH No 6483.
- [Cel91] Francois E. Cellier. *Continuous System Modeling*. Springer, 1991.
- [Cel96] F. E. Cellier. Object-oriented modeling: Means for dealing with system complexity. In *Proc. 15th Benelux Meeting on Systems and Control, Mierlo, The Netherlands*, pages 53–64, 1996.
- [CK06] Francois Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer, 2006.
- [ECK⁺11] M. Einhorn, F.V. Conte, C. Kral, C. Niclas, H.Popp, and J.Fleig. A modelica library for simulation of electric energy storages. In *Proceedings of the 8th Modelica Conference*, 2011. https://www.modelica.org/events/modelica2011/Proceedings/pages/papers/17_4_ID_105_a_fv.pdf.
- [ECO93] H. Elmqvist, F.E. Cellier, and M. Otter. Object-oriented modeling of hybrid systems. In *Proc. ESS'93, SCS European Simulation Symposium, Delft, The Netherlands*, pages xxxi–xli, 1993.

- [Föll08] Otto Föllinger. *Regelungstechnik, Einführung in die Methoden und ihre Anwendungen*. Hüthig, 2008. 10. Auflage.
- [Fri04] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. John Wiley & Sons, 2004.
- [KMR06] D.C. Karnopp, D.L. Margolis, and R.C. Rosenberg. *System Dynamics - Modeling and Simulation of Mechatronic Systems*. John Wiley & Sons, 2006. 4th edition.
- [LG94] Lennart Ljung and Torkel Glad. *Modeling of Dynamic Systems*. Prentice-Hall, 1994.
- [Mey06] Martin Meyer. *Signalverarbeitung: Analoge und digitale Signale, Systeme und Filter*. Vieweg-Verlag, 2006. 4., überarbeitete und aktualisierte Auflage.
- [Ott09] Martin Otter. Modeling, simulation and control with modelica 3.0 and dymola 7.0. Preliminary Draft, January 21, 2009; Vorlesungsskriptum, 2009.
- [Pap01] L. Papula. *Band 2: Mathematik für Ingenieure und Naturwissenschaftler*. Vieweg, 2001. 10., durchgelesene Auflage.
- [Pay61] H.M. Paynter. *Analysis and Design of Engineering Systems*. M.I.T. Press, 1961.
- [Sch09a] Thomas Schmitt. Modeling of a motorcycle in dymola/modelica. Master's thesis, Vorarlberg University of Applied Sciences, 2009.
- [Sch09b] D. Schröder. *Elektrische Antriebe - Regelung von Antriebssystemen*. Springer, 2009. 3. Auflage.
- [sEM98] K.J. Åström, H. Elmqvist, and S.E. Mattsson. Evolution of continuous-time modeling and simulation. In *The 12th European Simulation Multiconference, ESM'98, Manchester*, 1998.
- [Sue04] Dieter Suess. Kapitel 1 gewöhnliche differentialgleichungen. In *Skriptum der TU Wien*, 2004.
- [Unb00] Heinz Unbehauen. *Regelungstechnik II, Zustandsregelung, digitale und nicht-lineare Regelsysteme*. Vieweg, 2000. 8. Auflage.
- [Unb05] Heinz Unbehauen. *Regelungstechnik I, Klassische Verfahren zur Analyse und Synthese linearer kontinuierlicher Regelsysteme, Fuzzy-Regelsysteme*. Vieweg, 2005. 13. Auflage.
- [Zim10] Dirk Zimmer. *Equation-Based Modeling of Variable-Structure Systems*. PhD thesis, ETH Zürich, Switzerland, 2010. Diss ETH No 18924.