Entwicklung von intervallarithmetischen Methoden zur Berücksichtigung von Unsicherheiten in der Ökobilanzierung

von der Fakultät Bau- und Umweltingenieurwissenschaften der Universität Stuttgart
zur Erlangung der Würde einer Doktor-Ingenieurin (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von

Helen Luisa Hein

aus Stuttgart

Hauptberichter: Prof. Dr.-Ing. Harald Garrecht
Mitberichter: Prof. Dr.-Ing. Wolfgang Nowak
Mitberichterin: Prof. Dr. Marzia Traverso (Ph.D.)

Tag der mündlichen Prüfung: 27.09.2022

Institut für Werkstoffe im Bauwesen der Universität Stuttgart

2022

Ein Weg entsteht, wenn man ihn geht.

Vorwort

Diese Arbeit entstand während meiner Tätigkeit als wissenschaftliche Mitarbeiterin und Doktorandin am Institut für Werkstoffe im Bauwesen der Universität Stuttgart.
Herrn Prof. Harald Garrecht danke ich ganz herzlich für die Forschungsfreiheit, die er mir während meiner Zeit am Institut gewährt hat, sowie die fortwährende Unterstützung und stetige Betreuung meiner Arbeit. Ebenso danke ich Frau Prof. Marzia Traverso und Herrn Prof. Wolfgang Nowak sehr für die bereitwillige Übernahme des Mitberichts. Mein besonderer Dank gilt Dr. Joachim Schwarte für die zielführende Kritik und den festen Glauben an diese Dissertation.

Inhaltsverzeichnis

Abkürzungsverzeichnis

$A \in \mathbb{R}^{nxn}$	quadratische Matrix mit reellen Zahlen
$A^I \in \mathbb{IR}^{nxn}$	quadratische Intervall-Matrix mit Intervallgrenzen aus reellen Zahlen
AP	Versauerungspotenzial in kg SO_2 äq.(„**A**cid **P**otential“)
ADPE	Potenzial für den Abbau abiotischer Ressourcen in kg Sb äq.
ADPF	Potenzial für den Abbau abiotischer Brennstoffe in MJ
B	Umweltmatrix
EP	Eutrophierungspotenzial in kg PO_4^3 äq. („**E**utrification **P**otential“)
f	Bedarfsvektor
f_i	Bedarfsfaktor$_i$
g	Umweltvektor
g_i	Umweltfaktor$_i$
h	Wirkungsvektor
h_i	Wirkungsfaktor$_i$
GS	**G**leichungs**s**ystem
GUI	Grafische Benutzeroberfläche („**G**raphical **U**ser **I**nterface“)
GWP 100	Treibhauspotential in kg CO_2 äq. („**G**lobal **W**arming **P**otential“) (über Zeitraum von 100 Jahren)
inv.-pos.	invers-positiv, Elemente der Inverse ≥ 0
IPCC	**I**ntergovernmental **P**anel of **C**limate **C**hange, auch „Weltklimarat“
Ivari	Java-Paket, enthält in dieser Arbeit entwickeltes, intervallbasiertes Berechnungsverfahren
LCA	**L**ife **C**ycle **A**ssessment
LGS	**L**ineares **G**leichungs**s**ystem
MultiVaLCA	**Multi Va**lue **L**ife **C**ycle **A**ssessment
m%	Massenprozent
NP-schwer	Komplexitätsklasse der Informatik, NP steht für „**n**ichtdeterministisch **p**olynomielle Zeit“
ODP	Ozonabbaupotential der Stratosphäre in C_2H_4 äq. („**P**hotochemical **O**zone **C**reation **P**otential“)
P	Produktmatrix
PE(N)RT	Primärenergie (nicht-)erneuerbar total in MJ („**P**rimary **E**nergy (**N**on-)**R**enewable **T**otal“)
POCP	Ozonbildungspotenzial der Troposphäre in R11 äq. („**P**hotochemical **O**zone **C**reation **P**otential“)
PM_i	Prozessmodul$_i$
PS_i	Produktsystem$_i$
Q	Charakterisierungsmatrix
s	Skalierungsvektor
s_i	Skalierungsfaktor$_i$
UnsI	**Uns**icherheits-**I**ndex, in dieser Arbeit entwickelt

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzfassung

Die Ökobilanzierung gilt als wissenschaftlich anerkanntes Verfahren, mit dem potentielle Umweltwirkungen von technischen Systemen abgeschätzt werden können. Hierfür wird eine Vielzahl von Informationen aus unterschiedlichen Bereichen der Ingenieur- und Naturwissenschaften benötigt. Zur Erfassung der technischen Systeme sind sehr sensible Herstellerdaten sowie tiefgreifende Informationen zu vor- und nachgelagerten Prozessen notwendig. Diese sind oftmals nicht umfassend verfügbar. Die Abbildung der Umweltwirkungen basiert auf mitunter sehr komplexen Modellen, deren Ergebnisse von signifikanten Unsicherheiten geprägt sein können. Dies hat zur Folge, dass die Sicherheit der Ergebnisse von Ökobilanzen häufig in Frage gestellt wird. Es fehlt ein ingenieur- und praxistaugliches Verfahren, mit dem Unsicherheiten umfassend und transparent berücksichtigt werden können. Problematisch hierbei sind die Heterogenität der Unsicherheit sowie der relative Ansatz, der mit der Methodik der Ökobilanzierung einhergeht. So eignen sich je nach Art der Unsicherheit unterschiedliche Methoden zu deren Analyse. Die Zusammenführung diverser Verfahren ist mitunter zwar möglich, erschwert aber die Interpretation der Ergebnisse. Durch den relativen Ansatz fehlt darüber hinaus die „sichere Seite“: eine Veränderung *eines* Kennwerts verändert nicht nur diesen, sondern auch den relativen Bezug und die Beurteilung *aller anderen* Kennwerte, die mit diesem im Verhältnis stehen. Unsichere Daten können daher nicht mit Sicherheitsfaktoren wie bei statischen Bemessungen *auf der sicheren Seite liegend* belegt werden, da sonst alle Systeme, welche diese Kennwerte verwenden, zu optimistisch oder zu pessimistisch eingeschätzt werden können.

Vor diesem Hintergrund sollte in dieser Arbeit die Methodik der Ökobilanzierung um einen intervallbasierten Ansatz erweitert werden. Mit diesem sollen Unsicherheiten in Form von Intervallen erfasst werden, wobei die Unter- und Obergrenzen die Grenzen zum Unmöglichen oder Irrelevanten darstellen. Dies verhindert Über- oder Unterschätzungen, indem bei unsicheren Daten die gesamte Bandbreite des Möglichen und Relevanten einbezogen wird. Aus den resultierenden Ergebnissen können robuste Aussagen über potentielle Umweltwirkungen abgeleitet werden. Die Lösung intervallarithmetischer Probleme weist eine hohe Komplexität auf. Im Zuge der Berechnung können Intervalle durch mehrfaches Einsetzen zu sehr aufgeweitet werden, was die Aussagekraft der Resultate reduziert. Der Fokus lag daher auf der Entwicklung und Implementierung eines intervallbasierten Berechnungsverfahrens, welches unter Berücksichtigung des Rechenzeitaufwands möglichst enge Ergebnisintervalle ermittelt. Hierfür wurden diverse Gleichungslöser implementiert und mit Testreihen auf Basis fiktiver Produktsysteme untersucht. Es ließ sich kein einzelner Gleichungslöser identifizieren, der hinsichtlich der Intervallweiten und der Rechenzeit für alle Arten und Dimensionen von Produktmatrizen besonders geeignet ist. Stattdessen stellt die Kombination mehrerer Gleichungslöser eine gute Alternative dar. Es wurde ein allgemeingültiges Verfahren abgeleitet, das unter Berücksichtigung der vorliegenden Dimension und Art der Produktmatrix geeignete Gleichungslöser aufruft. Hierfür wurde auch ein Algorithmus entworfen, mit dem der Rechenaufwand durch Splitten großer Matrizen in kleinere Submatrizen reduziert werden kann. Das entwickelte Verfahren zur intervallbasierten Ökobilanzierung kann den Grundstein für eine neue Generation von Ökobilanzen darstellen, mit denen belastbare und abgesicherte Ergebnisse ermittelt werden können.

Abstract

Life cycle assessment (LCA) is a scientifically recognised method for estimating the potential environmental impacts of technical systems. This requires a large amount of information from different fields of engineering and natural sciences. In the context of technical systems, very sensitive manufacturer data as well as in-depth information on upstream and downstream processes are necessary, which are often not comprehensively available. The evaulation of environmental impacts is based on sometimes very complex models, the results of which can be characterised by significant uncertainties. As a result, the reliability of the results of life cycle assessments is often questioned. There is a lack of an engineering and practical procedure with which uncertainties can be comprehensively and transparently taken into account. The heterogeneity of uncertainty and the relative approach associated with the methodology of life cycle assessment are problematic here. Depending on the type of uncertainty, different methods are suitable for its analysis. Although it is sometimes possible to combine various methods, this makes the interpretation of the results more difficult. In addition, the relative approach lacks a „secure side“: a change in *one* parameter not only changes it, but also the relative reference and the assessment of *all other parameters* that are related to it. Uncertain data can therefore not be assigned safety factors as in static designs *lying on the safe side*, otherwise all systems using these characteristic values can be assessed too optimistically or too pessimistically.

Against this background, the methodology of life cycle assessment should be extended in this work to include an interval-based approach. With this approach, uncertainties are to be recorded in the form of intervals, whereby the lower and upper limits represent the boundaries to the impossible or irrelevant. This prevents over- or underestimation by including the full range of what is possible and relevant in uncertain data. Robust statements about potential environmental impacts can be derived from the resulting results. The solution of interval arithmetic problems is highly complex. In the course of the calculation, intervals can be widened too much by multiple insertions, which reduces the significance of the results. The focus was therefore on the development and implementation of an interval-based calculation method that determines the narrowest possible result intervals while taking the computing time into account. For this purpose, various equation solvers were implemented and examined with test series based on fictitious product systems. No single equation solver could be identified that is particularly suitable for all types and dimensions of product matrices in terms of interval widths and computing time. Instead, the combination of several equation solvers represents a good alternative. A generally valid procedure was derived that calls up suitable equation solvers, taking into account the dimension and type of product matrix at hand. An algorithm was also designed for this purpose, with which the computational effort can be reduced by splitting large matrices into smaller submatrices. The developed method for interval-based LCA can be the cornerstone for a new generation of LCAs that can deliver reliable and validated results.

Kapitel 1

Einleitung

1.1 Motivation

Die Ingenieurwissenschaften befassen sich mit der Erforschung, Entwicklung und Produktion von Technik. Dabei nimmt heutzutage das Streben, die einhergehenden Prozesse im Sinne einer nachhaltigen Entwicklung zu effizienten, konsistenten und suffizienten Prozessen zu überführen, einen zunehmend bedeutsamen Stellenwert ein. So ist auch im Bauingenieurwesen die ökologische Bewertung von Prozessen zur Gewinnung von Rohstoffen, zur Produktion von Bauprodukten, zur Konstruktion sowie Fertigung von Bauteilen und zur Errichtung von Gebäuden und sonstigen Bauwerken vermehrt in den Vordergrund gerückt - wobei gleichsam auch die Beseitigungs- und Verwertungsprozesse von Bauprodukten und Bauteilen sowie der Abriss von Gebäuden und die Prozesse zur Rückführung der in diesem Zuge anfallenden Stoffe in den Stoffkreislauf eine Rolle spielen.

Innovative Produkte aufgrund emissionsarmer, umweltschonender oder effizienter Herstellprozesse als „ökologisch nachhaltig“ auszuzeichnen, ist ein beliebtes Marketinginstrument. Für die zugrunde gelegten Berechnungsverfahren zur Bewertung der ökologischen Nachhaltigkeit werden Kenntnisse benötigt, welche über die Ingenieurwissenschaften hinausgehen - so sind neben der Erfassung und Analyse der technischen Systeme auch Informationen über die Einflüsse auf die Umwelt notwendig, welche wiederum z. B. klimatologische Kenntnisse erfordern. Zusätzlich müssen mögliche Änderungen der Systeme in der Zukunft und deren Einflüsse und Wechselbeziehungen auf die Umweltwirkungen prognostiziert werden. Dies ist insbesondere im Bauwesen von Bedeutung, da aus den vergleichsweise langen Nutzungszeiten entsprechend große Betrachtungszeiträume resultieren.

Die Aussagen zur ökologische Nachhaltigkeit beruhen in der Regel auf den Ergebnissen von Ökobilanzen, deren exakt angegebene Werte eine hohe Genauigkeit suggerieren. Tatsächlich können diese jedoch nach wie vor von einer Vielzahl von mitunter signifikanten Unsicherheiten geprägt sein. Bis heute fehlt ein ingenieurgerechtes, praxistaugliches und allgemeingültiges Verfahren, mit dem die Unsicherheiten umfassend berücksichtigt und die Größe der Unsicherheiten eingeordnet werden können. Es bedarf auch einer transparenten Darstellung unsicherer Resultate, aus denen robuste Aussagen über einzelne potentielle Umweltwirkungen abgeleitet werden können.

1.2 Zielsetzung

Das in dieser Arbeit vorgeschlagene Verfahren beruht auf der These, dass eine intervallbasierte Methodik zur Ökobilanzierung eine praxisgerechte Lösung darstellt, um Unsicherheiten in der Ökobilanzierung umfassend und einheitlich zu berücksichtigen und so durch die in Form von Intervallen erfassten unsicheren Daten robuste Er-

gebnisse zu generieren. Dabei stellen die Unter- und Obergrenzen der Intervalle die Grenzen zum Unmöglichen oder Irrelevanten dar. Dieser Ansatz fundiert auf der Überzeugung, dass sich Wissenschaft und mathematisch fundierte, aber vage und in gewisser Weise als unsicher eingestufte Ergebnisse nicht ausschließen, sondern bei bestimmten Unsicherheiten und bei Einhaltung der wissenschaftlichen Regeln sogar notwendig sind.

Ziel dieser Arbeit ist daher die Entwicklung und Implementierung eines Berechnungsverfahrens, welches für intervallbasierte Ökobilanzen verwendet werden kann. Es soll ein praxistaugliches Verfahren zur ökologischen Bewertung bereitgestellt werden, dass nicht vermeidbare Unsicherheiten in der Ökobilanzierung umfassend berücksichtigt, damit robuste Ergebnisse berechnet werden können. Bei der Entwicklung werden Kenntnisse aus dem Bereich der Informatik, der Mathematik und den Ingenieurwissenschaften verknüpft.

Ergebnis und Schwerpunkt der im Folgenden vorgestellten Arbeit ist die Entwicklung des intervallarithmetisch basierten Berechnungsverfahrens zur ökologischen Bilanzierung und dessen numerische Umsetzung. Die Entwicklung des Berechnungsverfahrens baut auf der bestehenden Methodik der Ökobilanzierung sowie dem bereits vor Jahren formulierten Vorschlag auf, Ökobilanzen intervallbasiert durchzuführen. Bei der Entwicklung des Berechnungsverfahrens werden umfassend alle möglichen Arten von Produktsystemen sowie die daraus resultierenden Typen von Produktmatrizen berücksichtigt, die auftreten können.

Die Forschungsfragen, durch welche diese Dissertation angeregt wurde, sind im Wesentlichen die folgenden:

- Welche Arten von Unsicherheit treten im Zuge der Ökobilanzierung auf und wie können diese Unsicherheiten berücksichtigt werden?
- Wie werden die technologischen Systeme modelliert und welche Besonderheiten lassen sich unter Berücksichtigung des graphentheoretischen Kontexts ableiten?
- Wie können Ökobilanzen intervallbasiert durchgeführt und Unsicherheiten eingeordnet werden? Welche Methoden zur Lösung der Intervallgleichungssysteme können hierfür verwendet werden?
- Wie können die Verfahren numerisch formuliert und die Algorithmen implementiert werden, sodass diese in der Praxis zur Anwendung kommen können?
- Wie geeignet sind die implementierten Algorithmen und welches Berechnungsverfahren lässt sich für intervallbasierte Ökobilanzen auf Basis der gewonnenen Erkenntnisse ableiten?

Als problematisch ist in der Ökobilanz die Vielzahl erforderlicher Informationen aus diversen Bereichen der Wissenschaft einzustufen, die mit ganz unterschiedlichen Arten von Unsicherheit einhergehen. Die möglichen Arten von Unsicherheit in der Ökobilanzierung müssen identifiziert und bei der Entwicklung eines allgemeingültigen Verfahrens zur Erfassung von Unsicherheit berücksichtigt werden.

Auch die Modellierung der Produktsysteme birgt Herausforderungen. Die Besonderheiten unterschiedlicher Systemstrukturen müssen erfasst und bei der Verfahrensentwicklung berücksichtigt werden. Durch die politisch angestrebte Schließung der Stoffkreisläufe sind in der Praxis der ökologischen Bilanzierung zunehmend Produktsysteme mit komplizierter Systemstruktur zu erwarten.

Ein wesentliches Ziel dieser Arbeit ist es, ein Verfahren zur intervallbasierten Ökobilanzierung zu entwickeln, mit dem möglichst schmale Ergebnisintervalle berechnet werden können, ohne dass die Rechenzeit die Praxistauglichkeit des Verfahrens eingeschränkt wird. Grund hierfür ist, dass ein zu großes Aufweiten des Ergebnisses dessen Aussagekraft reduziert. Zusätzlich ist ein Unsicherheitsindex erforderlich, mit dem die generierten Ergebnisse gemäß des vergleichenden Ansatzes der Ökobilanzierung hinsichtlich ihrer Unsicherheit - auch im Vergleich zu anderen Ökobilanzergebnissen - eingeordnet werden können. Auf methodischer Ebene ist dabei eine besondere Schwierigkeit, die Hülle der Lösungsmenge von intervallbasierten Gleichungssystemen zu ermitteln. Dies wird

als Problem mit hoher Komplexität eingeordnet. So kann das mehrfache Einsetzen von Intervallen zu einem zu großen Aufweiten von Ergebnisintervallen führen; dies wird in der Intervallarithmetik auch als Abhängigkeitsproblem bezeichnet. Es existieren verschiedene Verfahren, mit denen Näherungslösungen berechnet werden können. Diese sind mitunter aufgrund ihres Rechenzeitaufwandes als wenig praktikabel einzustufen. Daher ist eine weitere Fragestellung dieser Arbeit, ob die Produktmatrizen in kleinere Matrizen unterteilt werden können, um so die Rechenzeit zu reduzieren.

Zur Entwicklung des Berechnungsverfahrens müssen verschiedene Berechnungsalgorithmen zur Lösung der Intervall-Gleichungssysteme analysiert und modifiziert werden. Um diese objektorientiert zu programmieren, ist es notwendig, Klassen zu erstellen, mit denen intervallbasierte Skalare, Vektoren und Matrizen erzeugt werden können. Damit kleinere Matrizen einer übergeordneten Produktmatrix gesondert berechnet werden können, müssen weitere Klassen und Methoden implementiert werden.

Die Verwendbarkeit des intervallbasierten Ansatzes wird anhand von Parameterstudien fiktiver Produktsysteme erprobt und mit vollständigen Szenarioanalysen verifiziert. Dabei werden die möglichen Strukturen der Produktsysteme als auch diverse Matrixtypen der zugehörigen Produktmatrizen berücksichtigt. Aus den daraus gewonnenen Erkenntnissen soll ein Verfahren abgeleitet werden, mit dem intervallbasierte Ökobilanzen zur Berücksichtigung von Unsicherheiten durchgeführt werden können.

Das entwickelte Berechnungsverfahren wurde als Java-Paket *Ivari* („**I**nter**v**all**ari**thmetik“) implementiert. *Ivari* kann als Paket in das Ökobilanzierungsprogramm *MultiVaLCA* („**Multi Va**lue **L**ife **C**ycle **A**ssessment“), das derzeit am Institut für Werkstoffe im Bauwesen entwickelt wird, eingebettet werden. *MultiVaLCA* enthält eine grafische Benutzeroberfläche, die es dem Nutzer ermöglicht, Produktsysteme beliebiger Art und Komplexität einzugeben. Es unterstützt dabei den intervallbasierten Ansatz der Ökobilanzierung, sodass unsichere Werte in Form von Intervallen eingegrenzt werden können. Die auf Basis von *Ivari* berechneten Ergebnisintervalle zeigen durch die Intervallgrenzen die Grenze zum Unmöglichen oder Irrelevanten auf. Daneben soll ein geeigneter Unsicherheitsindex aufzeigen können, ob die generierten Ergebnisse im Vergleich zu anderen Ökobilanzen sicher genug sind, um aufschlussreiche Aussagen über die untersuchten Umweltwirkungen des Produktsystems zu treffen oder ob weitere Informationen für schmalere Ergebnisintervalle gesammelt werden sollten.

1.3 Aufbau der Arbeit

In Abbildung 1.1 ist der Aufbau der Doktorarbeit nach thematischem Schwerpunkt in Theorie, Analyse und Umsetzung, den einzelnen Kapiteln sowie der Untergliederung in „Stand der Technik“ (in weißem Hintergrund) und „Forschungsbeitrag“ (in grauem Hintergrund) dargestellt. Das aktuelle Kapitel enthält die Einleitung und bildet zusammen mit dem letzten Kapitel einen umfassenden Rahmen um die Arbeit.

Die ökologische Bilanzierung basiert auf der mathematischen Formulierung und Modellierung technischer Systeme und den damit verbundenen potentiellen Umweltwirkungen. Die mathematische Formulierung der Ökobilanzierung beruht auf linearer Algebra. Das im Zuge der Arbeit entwickelte Verfahren zur Ökobilanzierung basiert auf der Intervallarithmetik. Die hierfür notwendigen *theoretischen* Grundlagen werden in drei Kapiteln behandelt (vgl. Kapitel im linken oberen Bereich der Abbildung 1.1). In Kapitel 2 werden zunächst die notwendigen Grundlagen zur Mathematik und Informatik dargelegt. Das daran anschließende Kapitel 3 stellt gesondert die Grundlagen zur Intervallarithmetik sowie bekannter numerischer Methoden dar. Die Modellierung der Produktsysteme folgt im Wesentlichen der Graphentheorie, weswegen in Kapitel 4 die Modellierung von Systemen auf Basis der Graphentheorie thematisiert wird.

Zur Erweiterung der Methodik der Ökobilanzierung um ein allgemeingültiges Verfahren zur Berücksichtigung

von Unsicherheiten müssen die aktuell verfügbaren Methoden zur ökologischen Bilanzierung und zum Umgang mit Unsicherheiten analysiert werden. Dabei wird auch die Modellbildung der Produktsysteme im graphentheoretischen Kontext untersucht. Diese *Analysen* erfolgen in drei weiteren Kapiteln (vgl. oben rechts in Abbildung 1.1).

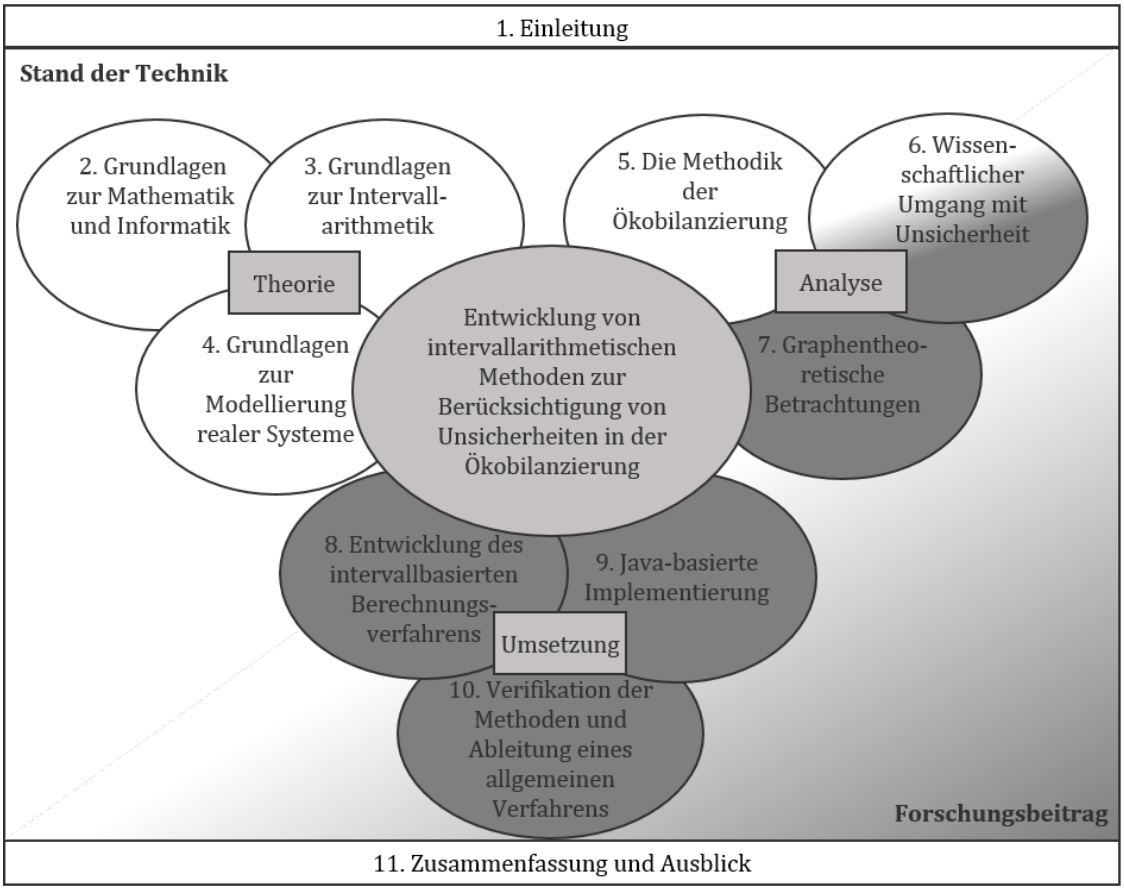


Abbildung 1.1: Gliederung der Doktorarbeit nach thematischem Schwerpunkt, Kapiteln sowie dem Stand der Technik und eigener Forschungsarbeit.

Hierfür werden die wesentlichen Begrifflichkeiten und Berechnungsmethoden der herkömmlichen Methodik der Ökobilanzierung in Kapitel 5 vorgestellt. Die Problematik mit den einhergehenden Unsicherheiten, die Unterscheidung der möglichen Typen von Unsicherheiten sowie der Stand der Wissenschaft zum Umgang mit diesen wird in Kapitel 6 erörtert. In diesem Zusammenhang wird auch der intervallbasierte Ansatz zur ökologischen Bilanzierung hergeleitet. Somit beginnt in diesem Kapitel auch der Forschungsbeitrag dieser Dissertation, der in Abbildung 1.1 durch einen grauen Hintergrund hervorgehoben wird. Die graphentheoretische Betrachtung der untersuchten Methodik erfolgt in Kapitel 7. Es werden verschiedene Produktsystemstrukturen identifiziert und einhergehende Besonderheiten der Produktmatrizen erläutert.

In den folgenden drei Kapiteln wird die *Umsetzung* des zu entwickelnden Verfahrens thematisiert (vgl. Abbildung 1.1). Dafür werden die Anforderungen an das zu entwickelnde Verfahren sowie die Vorgehensweise bei der Entwicklung in Kapitel 8 erläutert. Einige Aspekte der wesentlichen Implementierungsarbeiten werden in Kapitel 9 vorgestellt. Auf Basis der Ergebnisse der durchgeführten Parameterstudien werden in Kapitel 10 die implementierten Methoden verifiziert, analysiert und schließlich ein geeignetes Verfahren abgeleitet. Abschließend werden in Kapitel 11 die Ergebnisse diskutiert und ein Ausblick auf zukünftige Entwicklungsmöglichkeiten gegeben.

Kapitel 2

Grundlagen zur Mathematik und Informatik

Die mathematische Modellierung der Ökobilanzierung basiert auf linearer Algebra. Die numerische Entwicklung des in dieser Arbeit angestrebten Berechnungsverfahrens erfordert Kenntnisse aus verschiedenen Bereichen der Mathematik und Informatik. Für diese Arbeit relevante Aspekte dieser Themenfelder werden im Folgenden behandelt. In Kapitel 2.1 werden die Grundlagen zur linearen Algebra thematisiert, daran schließen in Kapitel 2.2 die Grundlagen zur Numerik und Informatik an. In Kapitel 2.3 werden relevante Aspekte zu linearen Gleichungssystemen erläutert und in Kapitel 2.4 numerische Lösungsverfahren vorgestellt. Abschließend werden in Kapitel 2.5 wesentliche Kenntnisse zur diskreten Mathematik und Stochastik aufgeführt.

2.1 Grundlagen zur linearen Algebra

Die lineare Algebra befasst sich mit Vektorräumen als algebraische Strukturen und linearen Abbildungen. Dabei stellen die Quellen [1], S. 513-867, [2] S. 143-215, [3], [4] eine breite Grundlagenbasis zur linear-algebraischen Mathematik bereit. Ein gebräuchliches Konzept zur Berechnung, Lösung und Darstellung linearer Gleichungssysteme ist die Verwendung von Matrizen. Die Zeilenanzahl m und die Spaltenanzahl n einer Matrix $A \in \mathbb{R}^{mxn}$ mit den Elementen a_{ij} legen den Matrixtyp mit der Dimension mxn fest. Die Dimension wird durch den Vektorraum $\mathbb{R}^{mxn}$ bestimmt. Der Körper des Vektorraums sind die reellen Zahlen. Eine Matrix $\in \mathbb{R}^{mx1}$ bzw. $\in \mathbb{R}^{1xn}$ wird als Zeilen- bzw. Spaltenvektor bezeichnet, eine Matrix $\in \mathbb{R}^{1x1}$ als Skalar. Matrizen $\in \mathbb{R}^{mxn}$, bei denen m gleich n ist, werden als quadratische Matrizen bezeichnet. In dem in dieser Arbeit entwickelten Verfahren werden Skalare, Vektoren sowie quadratische Matrizen zur Bilanzierung verwendet. Im Folgenden werden spezielle Formen quadratischer Matrizen sowie einhergehende Eigenschaften erläutert, die für das Verständnis dieser Arbeit relevant sind und im Zuge der Entwicklung des intervallbasierten Berechnungsverfahrens verwendet worden sind.

2.1.1 Diagonal- und Einheitsmatrix

Eine quadratische Matrix $A \in \mathbb{R}^{mxn}$ ist eine Diagonalmatrix $D \in \mathbb{R}^{nxn}$, wenn für deren Komponenten a_{ij} gilt (vgl. z. B. [1], S. 574):

$$a_{ij} = 0,\ wenn\ i \neq j$$

Eine besondere Form stellt die Einheitsmatrix E dar, die vorliegt, wenn zusätzlich gilt (vgl. z. B. [1], S. 575):

$$a_{ij} = 1,\ wenn\ i = j$$

2.1.2 Dreiecksmatrix

Eine Matrix $A \in \mathbb{R}^{nxn}$ ist eine obere Dreiecksmatrix, wenn für ihre Komponenten a_{ij} gilt (vgl. z. B. [3], S. 233):

$$a_{ij} = 0\ ,\ für\ i > j$$

Analog hierzu ist eine Matrix $A \in \mathbb{R}^{nxn}$ eine untere Dreiecksmatrix, wenn für ihre Komponenten a_{ij} gilt (vgl. z. B. [3], S. 233):

$$a_{ij} = 0,\ für\ i < j$$

2.1.3 Matrix der Cofaktoren

Die Cofaktoren $\tilde{a}_{ij}$ einer Koeffizientenmatrix $A \in \mathbb{R}^{nxn}$ zu den jeweiligen Indexpaaren ij ergeben sich durch die vorzeichenbehafteten Unterdeterminanten, den sog. Minoren. Die Minoren bilden die Determinante $det(A_{ij})$ der Untermatrix $A_{ij} \in \mathbb{R}^{(n-1)x(n-1)}$, welche sich durch Streichung der Zeile i und der Spalte j aus $A \in \mathbb{R}^{nxn}$ ergibt (vgl. z. B. [5], S. 150):

$$\tilde{a}_{ij} = (-1)^{i+j} \cdot det(A_{ij}) \tag{2.1}$$

Alle Cofaktoren $\tilde{a}_{ij}$ der Matrix A bilden die Matrix der Cofaktoren $\tilde{A}$.

2.1.4 Adjunkte

Die adjunkte Matrix oder kurz Adjunkte $adj(A)$ der Matrix A ergibt sich durch Transposition der Matrix der Cofaktoren $\tilde{A}$ der Matrix A (vgl. z. B. [5], S. 150):

$$adj(A) = \tilde{A}^T \tag{2.2}$$

Hierbei ist anzumerken, dass mitunter die Bezeichnung „adjungierte Matrix“ verwendet wird (vgl. z. B. [5], S. 150). Daneben wird auch die Bezeichnung „adjunkte Matrix“ (vgl. z. B. [3], S. 239) oder „Adjunkte“ (vgl. z. B. [1], S. 609) verwendet. Die Begriffe „adjunkten Matrix“ und „Adjunkten“ werden in dieser Arbeit bevorzugt verwendet.

2.1.5 Inverse und invers-positive Matrix

Eine quadratische Matrix $A \in \mathbb{R}^{nxn}$ gilt als regulär und damit invertierbar, wenn ihre Determinante $det(A) \neq 0$ ist (vgl. z. B. [5], S. 129). Die Multiplikation einer Inversen A^{-1} mit der zugehörigen Matrix A ergibt die Einheitsmatrix E (vgl. z. B. [2], S. 202, Anmerkung: hier wird die Einheitsmatrix angelehnt an den englischen Begriff „identity matrix“ mit I abgekürzt):

$$E = AA^{-1}$$

Eine invers-positive Matrix ist dabei eine Matrix, deren Inverse ausschließlich Elemente ≥ 0 aufweist.

2.1.6 Vergleichsmatrix

Als Vergleichsmatrix M(A) (englisch: „comparison matrix“) einer Matrix $A \in \mathbb{R}^{mxn}$ kann eine Matrix bezeichnet werden, die wie folgt definiert wird (vgl. z. B. [6], S. 2359 f.):

$$M(A) = \alpha_{ij},\ mit: \begin{cases} \alpha_{ij} = -|A_{ij}| \ \forall\ i \neq j \\ \alpha_{ij} = +|A_{ij}| \ \forall\ i = j \end{cases} \tag{2.3}$$

2.1.7 Z-Matrix

Eine Matrix $A \in \mathbb{R}^{nxn}$ kann als eine Z-Matrix bezeichnet werden, wenn deren Komponenten $a_{ij} \leq 0$ für alle $i \neq j$ sind (vgl. z. B. [7], S. 386).

2.1.8 P-Matrix

Eine Matrix $A \in \mathbb{R}^{nxn}$, deren Minoren > 0 sind, wird auch als P-Matrix bezeichnet (vgl. z. B. [8], S. 164).

2.1.9 M-Matrix

M-Matrizen stellen eine Subklasse von Z-Matrizen dar (vgl. z. B. [9], S. 176). Eine Z-Matrix $A \in \mathbb{R}^{nxn}$ kann den M-Matrizen zugeordnet werden, wenn u. a. gilt:

- Es existiert eine Inverse $A^{-1} \in \mathbb{R}^{nxn}$ und diese ist ≥ 0 (vgl. z. B. [10], S. 15).

Somit stellen M-Matrizen auch eine Teilmenge der P-Matrizen und invers-positiven Matrizen dar. Des Weiteren treffen folgende Punkte auf M-Matrizen zu (vgl. z. B. [9], S. 179 f.):

- A kann in eine untere Dreiecksmatrix L und eine obere Dreiecksmatrix R mit positiven Hauptdiagonalen zerlegt werden, sodass gilt: A = LR,
- Die Minoren der Hauptdiagonalen einer M-Matrix sind positiv,
- Mit einem Vektor $u > 0 \in \mathbb{R}^n$ gilt $Au > 0$.

Hierbei ist zu beachten, dass nicht alle invers-positiven Matrizen M-Matrizen sind; jedoch können invers-positive Z-Matrizen den M-Matrizen zugeordnet werden.

2.1.10 H-Matrix

Eine Matrix $A \in \mathbb{R}^{nxn}$ wird als nichtsinguläre H-Matrix deklariert, wenn ihre Vergleichsmatrix M(A) einer nichtsingulären M-Matrix entspricht (vgl. z. B. [11], S. 120). Die Vergleichsmatrizen M(A) von H-Matrizen können daher allgemein auch als Z-Matrizen bezeichnet werden.

2.1.11 Elementare Matrixumformungen

Zwei elementare Zeilen- und Spaltenumformungen einer Matrix sind die folgenden (vgl. z. B. [4], S. 112 f.):

- Addition von Zeilen bzw. Spalten oder deren Vielfaches zu einer Zeile bzw. Spalte,
- Vertauschung zweier Zeilen bzw. Spalten.

2.1.12 Determinante

Die Determinante $det(\mathrm{A})$ einer quadratischen Matrix $\mathrm{A} \in \mathbb{R}^{nxn}$ ist ein Skalar, der aus den Elementen von A berechnet wird (vgl. z. B. [2], S. 206). Wenn A die lineare Abbildung $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, $f(x) = \mathrm{A}x$ beschreibt, kann $det(\mathrm{A})$ als ein Verzerrungsfaktor der Volumenänderung betrachtet werden (vgl. z. B. [12], S. 33). Die Determinante $det(\mathrm{A}^{-1})$ der Inversen A^{-1}, welche die Umkehrabbildung der Vektoren w_1 bis w_n auf die Vektoren v_1 bis v_n des Vektorraums V darstellen, ist der Kehrwert von $det(\mathrm{A})$ (vgl. z. B. [5], S. 143):

$$det(\mathrm{A}^{-1}) = \frac{1}{det(\mathrm{A})} \tag{2.4}$$

Des Weiteren gilt, dass sich die Determinante bei den in Kapitel 2.1.11 genannten elementaren Matrixumformungen nicht ändert (vgl. z. B. [3], S. 233) - unter Berücksichtigung der Vorzeichenwechsel bei der Vertauschung zweier Zeilen bzw. Spalten. Eine weitere Matrixumformung stellt die Multiplikation einer Zeile oder Spalte mit einer Zahl i dar. Bei Anwendung dieser Matrixumformung ändert sich der Betrag der Determinante um den Faktor i (vgl. z. B. [3], S. 233). Für die Berechnung der Determinanten existieren verschiedene Methoden. Ein bekanntes Verfahren ist die Determinantenberechnung über den Gauß-Algorithmus (Gaußsches Eliminationsverfahren vgl. Kapitel 2.4.2). Alternativ kann die Determinante über den Entwicklungssatz von Laplace berechnet werden (Entwicklungssatz von Laplace vgl. Kapitel 2.4.1). Daneben gibt es Ansätze für mitunter effizientere Algorithmen zur Berechnung der Determinanten (vgl. z. B. [13]).

2.2 Grundlagen zur Numerik und Informatik

Zur Lösung (mathematischer) Probleme - wie z. B. die Lösung von Intervall-Gleichungssystemen - können Verfahren in Form von Algorithmen verwendet werden. Dabei ist die Entwicklung, Bewertung und Implementierung von Algorithmen Teil der Numerik. Die Quellen [10], [14], [15],[16] und [17] bieten eine breite Grundlagenbasis zur numerischen Mathematik, während weiterführende Informationen zur Informatik in [18], [19], [20], [21], [22] und [23] bereitgestellt werden. Numerische Verfahren kommen in der Regel dann zur Anwendung, wenn das Problem nicht analytisch lösbar ist oder die analytische Lösung zu viel Zeit in Anspruch nehmen würde. Auch lineare Gleichungssysteme werden heute in der Regel numerisch gelöst, wobei mittels geeigneter Algorithmen exakte oder approximierte Lösungen von Problemen generiert werden. In diesem Zusammenhang spielen Daten eine wesentliche Rolle, die zur Übertragung von Informationen dienen (vgl. Kapitel 2.2.1 Informationstheorie). Die Anforderungen an Algorithmen sind zum einen, möglichst genaue Größen zu ermitteln, was auch als Stabilität bezeichnet wird (vgl. Kapitel 2.2.2 Stabilität von Algorithmen). Zum anderen sollen die Algorithmen eine möglichst geringe Komplexität hinsichtlich des Rechenzeitaufwands oder des Speicherplatzbedarfs aufweisen (vgl. Kapitel 2.2.3 Komplexitätstheorie und Effizienz). In Kapitel 2.2.4 werden notwendige Grundlagen und Begrifflichkeiten zum Verständnis der im späteren Verlauf dieser Arbeit beschriebenen Implementierungsarbeiten erläutert.

2.2.1 Informationstheorie

In der Informatik spielt die mathematisch fundierte Informationstheorie eine wesentliche Rolle. Sie behandelt die Nachrichtenübertragung zur Übermittlung von Informationen.

Informationen

Eine Information ist eine Kenntnis, die vermittelt wird. Dies erfolgt durch die Übermittlung einer Nachricht von einem Sender zu einem Empfänger in Form von bspw. Signalen oder Codes, wobei die Information auch verschlüsselt vorliegen kann (vgl. z. B. [19], S. 37 f.). Die Nachricht muss interpretiert werden, um die Information zu erhalten - diese Interpretation ist mitunter subjektiv vom Empfänger geprägt.

Daten und Datensätze

Daten stellen Zustände oder Sachverhalte in digitaler Form dar, aus denen Informationen gewonnen werden können. Datensätze enthalten mehrere Angaben eines Objektes. Ihre Struktur erlaubt es, auf die Daten zuzugreifen, sie zu speichern und zu verändern (vgl. z. B. [23], S. 109).

2.2.2 Stabilität von Algorithmen

In der Numerik werden Algorithmen als stabil bezeichnet, wenn die durch Störungen von Daten eingebrachten Abweichungen in den numerisch ermittelten Ergebnissen im Vergleich zur natürlichen Lösung mit denselben eingebrachten Störungen des mathematischen Problems gering sind (vgl. z. B. [24], S. 4 ff.).

2.2.3 Komplexitätstheorie und Effizienz

Die Komplexitätstheorie dient zur Beurteilung der Effizienz von Algorithmen, welche zur Lösung von Problemen eingesetzt werden. Durch sie soll eingeordnet werden, wie aufwändig es ist, ein Problem hinsichtlich der Rechenzeit bzw. dem benötigten Speicherplatzbedarf algorithmisch zu lösen.

Eine Bestimmung der im schlechtesten Fall notwendigen Rechenzeit zur Lösung eines Problems ist über das O-Kalkül bzw. die O-Notation möglich; eine tiefere Einführung hierzu ist u. a. in [25], S. 439 ff. sowie [18], S. 65 ff. zu finden. Die Aufwandsabschätzung im schlechtesten Fall kann z. B. anhand der Anzahl der Zuweisungen und Berechnungsschritte bei zwei verschachtelten Schleifen, die n-mal durchlaufen werden, $O(n^2)$ betragen (vgl. [26], S. 16).

Daneben gibt es die Möglichkeit, zwischen den Komplexitätsklassen P und NP zu unterscheiden. Diese Komplexitätsklassen werden ausführlicher z. B. in [25], S. 446 ff. sowie [18], S. 68 ff. erläutert. Dabei beschreibt die Komplexitätsklasse P die Mengen von Problemen, die in polynomieller Zeit lösbar sind, während NP die Probleme umfasst, welche nichtdeterministisch in polynomieller Zeit gelöst werden können. Letztere Komplexitätsklasse kann zusätzlich in NP-schwer und NP-vollständig differenziert werden.

Tabelle 2.1: Reelle Laufzeit T(n) von Algorithmen unterschiedlicher Komplexität.

n	$T(n) = n$	$T(n) = n^2$	$T(n) = n^3$	$T(n) = 2^n$	$T(n) = n!$
5	$5 \cdot 10^{-9}$ sec	$2{,}5 \cdot 10^{-8}$ sec	$1{,}25 \cdot 10^{-7}$ sec	$3{,}2 \cdot 10^{-8}$ sec	$1{,}2 \cdot 10^{-7}$ sec
10	$1 \cdot 10^{-8}$ sec	$1 \cdot 10^{-7}$ sec	$1 \cdot 10^{-6}$ sec	$\approx 1{,}02 \cdot 10^{-6}$ sec	$\approx 3{,}6 \cdot 10^{-3}$ sec
20	$2 \cdot 10^{-8}$ sec	$4 \cdot 10^{-7}$ sec	$8 \cdot 10^{-6}$ sec	$\approx 1{,}05 \cdot 10^{-3}$ sec	≈ 77 Jahre
30	$3 \cdot 10^{-8}$ sec	$9 \cdot 10^{-7}$ sec	$2{,}7 \cdot 10^{-4}$ sec	$\approx 1{,}07$ sec	$\approx 8{,}4$ Brd. Jahre
40	$4 \cdot 10^{-8}$ sec	$1{,}6 \cdot 10^{-6}$ sec	$6{,}4 \cdot 10^{-5}$ sec	$\approx 18{,}3$ Minuten	∞
50	$5 \cdot 10^{-8}$ sec	$2{,}5 \cdot 10^{-6}$ sec	$1{,}25 \cdot 10^{-4}$ sec	≈ 13 Tage	∞
60	$6 \cdot 10^{-8}$ sec	$3{,}6 \cdot 10^{-6}$ sec	$2{,}16 \cdot 10^{-4}$ sec	$\approx 36{,}6$ Jahre	∞

Zusätzlich kann die reelle Laufzeit eines Algorithmus anhand der Rechenleistung des Computers ermittelt werden. Die Frequenz eines Prozessors wird z. B. in einem Gigahertz angegeben, was einer Milliarde Takte pro Sekunde entspricht. Anhand dieser kann die reelle Laufzeit berechnet werden, die ein Algorithmus bspw. bei einer Matrixdimension von n benötigt. In Tabelle 2.1 sind verschiedene Laufzeiten diverser Algorithmen mit unterschiedlicher Komplexität und einer Frequenz des Prozessors von einem Gigahertz abgebildet (angelehnt an [26], S. 17).

2.2.4 Objektorientierte Programmierung

Die objektorientierte Programmierung ist ein Programmierstil, der mit der Idee entwickelt wurde, komplexe Probleme leichter modellieren zu können (vgl. z. B. [27], Kapitel 1.2.5). Demnach lassen sich reale Systeme durch konkrete Objekte, abstrakte Typen und deren Wechselwirkungen und Beziehungen untereinander abbilden (vgl. z. B. [19], S. 415 f.). Dabei wird unterstellt, dass die Denkweise des Menschen objektorientiert ist (vgl. z. B. [27], Kapitel 1.2.5). Die mathematische Modellierung der realen Systeme in der Ökobilanzierung entspricht im Wesentlichen diesem objektorientierten Ansatz. Dabei können die Wirkungskategorien und Prozessmodule gewissermaßen als abstrakte Typen aufgefasst werden, welche erst durch die Festlegung von Mengen zu konkreten Objekten ausgebildet werden. Diese Objekte sind über Flüsse miteinander verbunden und stehen so in Wechselwirkung miteinander.

Programmiersprache Java

Java ist eine bekannte objektorientierte Programmiersprache, die seit den 1990er Jahren kontinuierlich weiterentwickelt wird (vgl. z. B. Kapitel [19], S. 418). In der Java-basierten Implementierung kann in vielen Fällen auf bestehende Pakete zurückgegriffen werden. So stellt bspw. das *JAMA Package* („**JA**va **MA**trix Package“) die Basis für eine matrixbasierte Notation in Java sowie einige Algorithmen zur Berechnung bereit (vgl. [28]). Intervallarithmetische Methoden sind im *IAMath Package* (vgl. [29]) implementiert. Die Implementierung des in dieser Arbeit vorgestellten Entwicklungsverfahrens beruht auf der Programmiersprache Java.

Software-Test-Verfahren

Im Zuge der Entwicklung einer Software kommen verschiedene Testverfahren zum Einsatz, mit denen kleine Programmeinheiten, Schnittstellen sowie das gesamte Programm erprobt werden. Während der Implementierung werden Modultests durchgeführt, mit denen einzelne Methoden oder Klassen getestet werden (vgl. z. B. [19], S. 699 f.). Weitere Tests im Laufe der Softwareentwicklung fokussieren die Funktionsfähigkeit von Schnittstellen sowie Tests am gesamten System, um bspw. die Eignung und Qualität der Software festzustellen. Daneben gibt es Tests, die von der Zielgruppe des Programms, den Anwendern, durchgeführt werden. Ziel ist es, zu prüfen, ob die Software die Anforderungen der Nutzer erfüllt.

Vor- und Nachbedingungen

Durch eine formale Sprache können Methoden und Programme präzise beschrieben werden. Mit Vorbedingungen können Voraussetzungen definiert werden, für welche die Methode oder das Programm spezifiziert ist. Nachbedingungen sind Kontrollinstrumente, mit denen die Funktionsfähigkeit des Programms verifiziert werden kann. (vgl. z. B. [22], S. 196 ff.)

Exceptions

Bedingungen, die den Programmablauf stören, können mit Ausnahmen, den sog. „Exceptions“, abgefangen, gemeldet und behandelt werden (vgl. z. B. [22], S. 323 ff.).

2.2.5 Teile-und-herrsche-Verfahren

Das Teile-und-herrsche-Verfahren (englisch: „divide and conquer“) ist ein Programmierprinzip, welches zur Entwicklung effizienter Suchalgorithmen dient. Ausführliche Informationen hierzu finden sich z. B. in [30], S. 313-334, [26], S.141-164. Der Ansatz beruht darauf, rekursiv ein komplexes Problem in immer kleinere Teilprobleme zu teilen. Dies erfolgt solange, bis die Teilprobleme lösbar sind. Aufbauend aus den Teillösungen wird dann die Gesamtlösung des komplexen Problems abgeleitet. Das Programmierprinzip wurde in dieser Arbeit bei der Entwicklung des Berechnungsverfahrens angewandt.

2.3 Lineare Gleichungssysteme

Lineare Gleichungssysteme (LGS) bestehen aus einer Menge linearer Gleichungen. Durch die Lösung(en) dieser LGS sollen alle Gleichungen gleichzeitig erfüllt sein. Quadratische Koeffizientenmatrizen mit vollem Rang sind eindeutig lösbar und weder über- noch unterbestimmt (vgl. auch [4], S. 201 ff.). Zum Lösen linearer Gleichungssysteme gibt es eine Reihe effizienter Algorithmen und eine Vielzahl von Optimierungsstrategien. Eine ausführliche Übersicht über bekannte und optimierte Algorithmen sind z. B. in [14], S. 115-221, [16] und [31] zu finden.

2.3.1 Darstellungsform

Unter Verwendung der Matrixnotation können lineare Gleichungssysteme in der folgenden Form dargestellt werden (vgl. z. B. [16], S. 1 ff.):

$$\mathrm{A}x = b \tag{2.5}$$

Die Koeffizientenmatrix $\mathrm{A} \in \mathbb{R}^{nxn}$ beschreibt die Koeffizienten des Gleichungssystems mit den Elementen a_{ij}, der Spaltenvektor x mit den Elementen x_i die Unbekannten und der Spaltenvektor b mit den Elementen b_i die Konstanten bzw. Absolutglieder (vgl. z. B. [1] S. 514). Ist $\mathrm{b} \neq 0$, ist das Gleichungssystem inhomogen, anderenfalls homogen.

2.3.2 Lösbarkeit quadratischer Gleichungssysteme

Die Lösbarkeit eines quadratischen, linearen Gleichungssystems mit der Koeffizientenmatrix A ist eindeutig, wenn A einen vollen Rang aufweist und damit die Determinante $det(\mathrm{A}) \neq 0$ ist (vgl. z. B. [4], S. 231). Die Lösungsmenge linearer Gleichungssysteme ändert sich auch nicht durch Matrixumformungen; die Systeme bleiben lösungsäquivalent (vgl. z. B. [4], S.118). Bei einer invertierbaren Koeffizientenmatrix A ergibt sich die Lösung zu:

$$x = \mathrm{A}^{-1} b \tag{2.6}$$

Numerisch werden die LGS jedoch aufgrund des Aufwands in der Regel meist nicht über die Inverse, sondern mittels alternativer Methoden gelöst (vgl. z. B. [10], S. 19).

2.4 Numerische Lösungsverfahren

Die numerischen Lösungsverfahren linearer Gleichungssysteme können in direkte und iterative Verfahren unterschieden werden. Mit direkten Verfahren kann - mit Ausnahme von Rundungsfehlern - die exakte Lösung ermittelt werden; bei iterativen Methoden wird die Lösung approximiert. Direkte Verfahren gelten in der Numerik als stabiler (Erläuterung zur Stabilität vgl. Kapitel 2.2.2), dafür eignen sich iterative Methoden bei sehr großen Gleichungssystemen aufgrund ihres geringeren Rechenzeitaufwands. Direkte Methoden basieren häufig auf dem Prinzip des sog. Faktorisierens, d. h. die Koeffizientenmatrix A $\in \mathbb{R}^{nxn}$ des zu lösenden Gleichungssystems wird in eine obere Dreiecksmatrix R $\in \mathbb{R}^{nxn}$ und eine untere Dreiecksmatrix L $\in \mathbb{R}^{nxn}$ zerlegt. Es gilt mit der Matrix M und MA = R sowie dem Vektor $c = \mathrm{M}b$:

$$\mathrm{A}x = b \Longrightarrow \mathrm{R}x = c \tag{2.7}$$

Dabei stellt x mit $x = (x_1 ... x_n)^{\mathrm{T}}$ die eindeutige Lösung des linearen Gleichungssystems dar.

Im folgenden Kapitel 2.4.1 wird der Entwicklungssatz von Laplace zur Berechnung der Determinanten vorgestellt. Daran anschließend wird in Kapitel 2.4.2 das Gaußsche Eliminationsverfahren erläutert. Kapitel 2.4.3 thematisiert das direkte Adjunkten-Verfahren, bei dem das Gleichungssystem über die Inverse gelöst wird. Abschließend wird in Kapitel 2.4.4 die Präkonditionierung erläutert - eine Methode, mit der die Stabilität verbessert werden kann. Zum Lösen linearer Gleichungssysteme gibt es eine Reihe effizienter Algorithmen und eine Vielzahl von Ansätzen zur Effizienzsteigerung dieser. Eine ausführliche Übersicht über bekannte und modifizierte Algorithmen sind z. B. in [14], S. 115-221, [16] und [31] zu finden.

2.4.1 Entwicklungssatz von Laplace

Bei Anwendung des Entwicklungssatzes von Laplace entspricht die Determinante $det(\mathrm{A})$ der Summe von denjenigen Produkten, die mit den Elementen a_{ij} und den zugehörigen Cofaktoren $\tilde{a}_{ij}$ (Cofaktoren vgl. Kapitel 2.1.3) berechnet werden können. Demnach ergibt sich die Determinante entwickelt nach der i-ten Zeile zu (vgl. z. B. [3], S. 231):

$$det(\mathrm{A}) = \Sigma_{j=1}^{n} a_{ij}\tilde{a}_{ij}; \tag{2.8a}$$

Analog kann die Determinante nach der j-ten Spalte entwickelt werden:

$$det(\mathrm{A}) = \Sigma_{i=1}^{n} a_{ij}\tilde{a}_{ij} \tag{2.8b}$$

In der Numerik gilt dieses Verfahren mit einer Komplexität der Größenordnung von O(n!) insbesondere bei größeren Matrixdimensionen als ineffizient (O-Notation abgeleitet aus berechnetem Aufwand in [13], S. 28 f.).

2.4.2 Gaußsches Eliminationsverfahren

Das Gaussche Eliminationsverfahren weist gemäß der O-Notation eine Komplexität von bis zu $\mathrm{O}(n^3)$ auf, abhängig von der Durchführung (O-Notation abgeleitet aus dem berechneten Aufwand, Aufwand vgl. z. B. [32] S. 37). Beim Gaußschen Eliminationsverfahren wird das zu lösende Gleichungssystem $\mathrm{A}x = b$, $\mathrm{A} \in \mathbb{R}^{nxn}$, $b \in \mathbb{R}^{nx1}$ in ein äquivalentes Gleichungssystem $\dot{\mathrm{A}}x = \dot{b}$, $\dot{\mathrm{A}} \in \mathbb{R}^{nxn}, \dot{b} \in \mathbb{R}^{nx1}$ überführt, sodass $\dot{\mathrm{A}}$ einer oberen Dreiecksmatrix entspricht (vgl. z. B. [14], S. 136 ff.):

$$\mathrm{A}x = b \Longrightarrow \dot{\mathrm{A}}x = \dot{b}$$

Hierfür werden Variablen mittels elementarer Matrixumformungen eliminiert, was auch als Vorwärtselimination bezeichnet wird. Der Aufwand der Vorwärtselimination beträgt $O(n^2)$ (vgl. z. B. [32] S. 28). Da $\dot{A}$ dann einer oberen Dreiecksmatrix entspricht, kann die Bestimmung der Lösung x des Gleichungssystems $\dot{A}x = \dot{b}$, welche gleichsam die Lösung des Gleichungssystems $Ax = b$ darstellt, durch Rückwärtssubstitution von der letzten Zeile an durch rückwärtiges Einsetzen in die darüber liegenden Zeilen ermittelt werden. Der Aufwand der Rückwärtselimination beträgt $O(n^2)$ (vgl. z. B. [32] S. 29). Ohne Pivotisierung kann die Lösung x des Gleichungssystems wie folgt ermittelt werden (vgl. z. B. [10], S. 97 f.):

$$x_i = \frac{1}{\dot{a}_{ii}}(\dot{b}_i - \sum_{k=i+1}^{n} \dot{a}_{ik} x_k) \tag{2.9}$$

$$mit\ \frac{1}{\dot{a}_{ii}} \neq 0, i = n, ..., 1,$$

Die zeilenweisen Umformungen der Elemente von A und b zur Überführung in das äquivalente Gleichungssystem $\dot{A}x = \dot{b}$ ergeben sich dabei mit $0 \notin a_{kk}^k$ wie folgt (vgl. [33], S. 8 ff.):

$$a_{ij}^{k+1} = a_{ij}^k - \frac{a_{ik}^k}{a_{kk}^k} a_{kj}^k,\ für\ i, j \geq k+1, ..., n$$

$$b_i^{k+1} = b_i^k - \frac{a_{ik}^k}{a_{kk}^k} b_k^k,\ für\ i = k+1, ..., n$$

$$mit\ k = 1, ..., n-1\ sowie\ i, j = k+1, ..., n$$

Kann die Bedingung $0 \notin a_{kk}^k$ nicht erfüllt werden, sind Umformungen der Matrix notwendig, damit der Algorithmus durchgeführt werden kann. Durch Pivotisierung können Zeilen und ggfs. Spalten vertauscht werden. Bei Zeilenvertauschungen wird dies als Spaltenpivotsuche oder teilweise bzw. partielle Pivotisierung bezeichnet. Werden auch Spaltenvertauschungen durchgeführt, liegt eine vollständige Pivotisierung vor (vgl. auch [10], S. 97 f.). Die O-Notation des Gauß-Algorithmus mit Spaltenpivotisierung beträgt $O(n^3)$ (vgl. z. B. [32] S. 37). Bei Anwendung der Pivotisierung wird ein sog. Pivotelement a_{pivot} einer Spalte oder aber der gesamten Matrix gesucht. Häufig wird das wird das betragsmäßig größte Element detektiert. Auch die Anfälligkeit für Rundungsfehler kann durch Pivotisierung reduziert werden.

Im Zuge der numerischen Umsetzung des Gaußschen Algorithmus bei teilweiser Pivotisierung wird die Matrix $A \in \mathbb{R}^{nxn}$ spaltenweise nach einem Pivotelement durchsucht. Im Zuge einer Spaltenpivotsuche ergibt sich in einer Spalte k das Pivotelement a_{pivot} als das betragsmäßig größte Element mit:

$$a_{pivot} = max|a_{ik}|,$$
$$mit\ i = k, ..., n$$

Nach erfolgter Durchsuchung einer Spalte wird die jeweilige Zeile p, in der sich das Pivotelement a_{pivot} befindet, mit der k-ten Zeile in der Matrix A sowie des Vektors b vertauscht; die Elemente dieser umgeformten Matrix $\overline{A}$ bzw. dieses umgeformten Vektors $\overline{b}$ werden nach [33], S. 12 ff. wie folgt angegeben:

$$\overline{a}_{ij}^k = \begin{cases} \overline{a}_{kj}^k,\ falls\ i = p \\ \overline{a}_{pj}^k,\ falls\ i = k \\ \overline{a}_{ij}^k,\ falls\ i \neq p, k \end{cases}$$

$$\overline{b}_i^k = \begin{cases} \overline{b}_k^k, \ falls\ i = p \\ \overline{b}_p^k, \ falls\ i = k \\ \overline{b}_i^k, \ falls\ i \neq p,k \end{cases}$$
$$mit\ i,j = 1,...,n$$

Die Nichtnullelemente unterhalb des Elements $\overline{a}_{kk}$, d. h. die Elemente $\overline{a}_{k(k+1)}$ bis $\overline{a}_{kn}$ der Matrix $\overline{\mathrm{A}}$ werden nach Identifikation des Pivotelements der Spalte k durch Multiplikation und Zeilenaddition eliminiert. Die Elemente der im nächsten - und damit $(k+1)$-ten - Schritt der unterhalb der Diagonalen umgeformten Matrix A^{k+1} werden im Folgenden als a_{ij}^{k+1} angegeben (vgl. [10], S. 98 ff.):

$$a_{ij}^{k+1} = \overline{a}_{ij}^k - c_{i(k+1)} \cdot \overline{a}_{(k+1)j}^k,$$
$$mit\ c_{i(k+1)} = \frac{\overline{a}_{i(k+1}^k}{\overline{a}_{(k+1)(k+1}^k}$$
$$und\ i,j = k+2,...,n$$

Ein Java-basierter Gauß-Algorithmus ist z. B. in [34] zu finden.

2.4.3 Adjunkten-Verfahren

Für die Berechnung der Inversen A^{-1} gibt es verschiedene Methoden; eine Möglichkeit ist die Verwendung der adjunkten Matrix bzw. der Adjunkten $adj(\mathrm{A})$ (vgl. Kapitel 2.1.4) einer Matrix $\mathrm{A} \in \mathbb{R}^{nxn}$ (vgl. z. B. [35], S. 201 ff. und [3], S. 239 f.):

$$\mathrm{A}^{-1} = \frac{1}{det(\mathrm{A})} adj(\mathrm{A}) \tag{2.10}$$

Die adjunkten Matrix ergibt sich durch Transposition der Matrix der Cofaktoren $Cof(\mathrm{A})$:

$$adj(\mathrm{A}) = Cof(\mathrm{A})^{\mathrm{T}} \tag{2.11}$$

Die Cofaktoren Cof_{ij} werden durch Berechnung der Minoren bzw. Unterdeterminanten $\mathrm{D}_{ij}(\mathrm{A})$ derjenigen Matrix ermittelt, die sich ergibt, wenn die Zeile i und die Spalte j der Matrix A entfernt werden:

$$Cof_{ij} = (-1)^{i+j} \cdot \mathrm{D}_{ij}(\mathrm{A}) \tag{2.12}$$

Ein Java-basierter Algorithmus zur Berechnung der adjunkten Matrix ist z. B. in [36] zu finden.

2.4.4 Präkonditionierung linearer Gleichungssysteme

Die Präkonditionierung linearer Gleichungssysteme beschreibt ein Verfahren, welches ein Gleichungssystem in ein äquivalentes Gleichungssystem überführt. Es wird insbesondere bei iterativen Verfahren angewandt, um die Konvergenz des Gleichungssystems zu verbessern. Bei der sog. Linkspräkonditionierung wird das lineare Gleichungssystem von links mit einer regulären Matrix R multipliziert. R sollte dabei eine bestmögliche Annäherung an die Inverse von A^{-1} darstellen.

$$\mathrm{A}x = b \Longrightarrow \mathrm{RA}x = \mathrm{R}b \tag{2.13}$$

Dadurch ergibt sich mit RA = M und Rb = r das präkonditionierte Gleichungssystem, durch welches dann die Lösung ermittelt wird (vgl. z. B. [16], S. 250 ff.):

$$\mathrm{RA}x = \mathrm{R}b \Longrightarrow \mathrm{M}x = r \tag{2.14}$$

2.5 Grundlagen zur diskreten Mathematik und Stochastik

Im Folgenden werden relevante Aspekte der diskreten Mathematik sowie der Stochastik erläutert. Ausführliche Informationen zu Grundlagen der diskreten Mathematik finden sich bspw. in den Quellen [37], [38], [39]; weitere Informationen zur Stochastik z. B. in [40], [41]. Die zur diskreten Mathematik ebenfalls zuordenbare Graphentheorie wird im Kapitel 4.2 gesondert behandelt.

2.5.1 Kardinalität von Mengen

Die Anzahl von Elementen einer Menge M wird als Kardinalität K(M) bezeichnet. Sie kann wie folgt angegeben werden (vgl. auch [37], S. 13 ff.):

$$\mathrm{K(M)} = |\mathrm{M}|$$

2.5.2 Kartesisches Produkt

Das kartesische Produkt der Mengen M_1 bis M_k kann definiert werden als (vgl. [37], S. 12):

$$\mathrm{M}_1 x \mathrm{M}_2 x ... x \mathrm{M}_{k-1} x \mathrm{M}_k = \prod_{i=1}^{k} \mathrm{M}_i$$

Die Anzahl der möglichen Tupel von $\prod_{i=1}^{k} \mathrm{M}_i$ ergibt sich aus dem Produkt der Kardinalitäten der Mengen M_1 bis M_k (vgl. [37], S. 22):

$$|\mathrm{M}_1 x \mathrm{M}_2 x ... x \mathrm{M}_{k-1} x \mathrm{M}_k| = \prod_{i=1}^{k} |\mathrm{M}_i|$$

Die Anzahl möglicher Tupel des kartesischen Produkts von n Mengen mit einer Kardinalität von 2 beträgt 2^n.

2.5.3 Permutation

Die Anordnung von k Elementen einer Menge M in einer bestimmten Reihenfolge wird als Permutation $\mathrm{P}er(\mathrm{M})$ bezeichnet, wenn für die Kardinalität K(M) gilt ([37], S. 32):

$$\mathrm{K(M)} = |\mathrm{M}| = n = k$$

Die Anzahl möglicher Permutationen $|\mathrm{P}er(\mathrm{M})|$ der Menge M und Kardinalität K(M) = n bei einer Permutation ohne Wiederholung ist die Fakultät (vgl. z. B. [38], S. 10 ff. sowie [39], S. 526 f.):

$$|\mathrm{P}er(\mathrm{M})| = n!$$

Die Anzahl möglicher Zeilenvertauschungen einer Matrix $\mathrm{A} \in \mathbb{R}^{mxn}$ beträgt $m!$, die Anzahl möglicher Spaltenvertauschungen beträgt $n!$. Ein Java-basierter Algorithmus zur Permutation ist bspw. in [42] zu finden.

2.5.4 Variation

Die Anordnung von k Elementen einer Menge M in einer bestimmten Reihenfolge wird als Variation $Var(M)$ bezeichnet (vgl. z. B. [39], S. 528 f.), wenn für die Kardinalität K(M) gilt:

$$K(M) = |M| = k < n$$

Bei einer Variation mit k-facher Wiederholung ist die Anzahl möglicher Variationen $|Var(M)|$ der Menge M und Kardinalität $K(M) = n$ das kartesische Produkt mit der k-maligen Menge M selbst (vgl. z. B. [43], S. 20):

$$|Var(M)| = n^k$$

Die Anzahl möglicher Variationen $|Var(M_n)|$ ergibt sich bei m Mengen mit jeweils genau n Elementen, von denen jeweils k Elemente ausgewählt werden, durch das kartesische Produkt zu:

$$|Var(M_n)| = (n^k)^m$$

2.5.5 Stochastik

Die Stochastik befasst sich mit zufälligen Ereignissen, die nicht (ohne Unsicherheit) vorhersehbar sind und beinhaltet die schließende Statistik und die Wahrscheinlichkeitstheorie. Während die Wahrscheinlichkeitstheorie die mathematische Theorie zur Modellierung und Analyse zufälliger Ereignisse beschreibt, können mit den Methoden der schließenden Statistik aus beobachteten Daten geeignete Modelle abgeleitet werden, mit denen die Eintrittswahrscheinlichkeit bestimmter Größen bestimmt werden kann (vgl. z. B. [41], S. 183 ff.).

Schließende Statistik

Statistische Methoden dienen zur Abschätzung zufälliger Abweichungen. Sie sind geeignet, um von den durch Wiederholungen gewonnenen Messwerten einer Stichprobe auf die Grundgesamtheit zu schließen. Voraussetzung hierfür ist, dass die Messungen unbeeinflusst voneinander durchgeführt werden und bei der Messung die gleichen Randbedingungen vorliegen (vgl. z. B. [40], S. 265 ff., [41], S. 183 ff.). Mittels der Methoden der schließenden Statistik ist es möglich, dass auf Grundlage der vorliegenden Daten Wahrscheinlichkeiten bzw. Wahrscheinlichkeitsverteilungen bestimmt werden können (vgl. z. B. [41], S. 185).

Wahrscheinlichkeitstheorie

In der Wahrscheinlichkeitstheorie werden Ereignisse als Mengen definiert, welchen Eintrittswahrscheinlichkeiten zwischen 0 und 1 zugeordnet werden (vgl. Kolmogorov-Axiome, z. B. in [40], S. 176). Ein Ereignis, dem eine Eintrittswahrscheinlichkeit von 1 zugeordnet wird, ist demnach ein sicher eintretendes Ereignis - analog hierzu ist ein Ereignis, dem eine Eintrittswahrscheinlichkeit von 0 zugeordnet wird, demnach ein unmögliches Ereignis (vgl. [40], S. 176 unter Beachtung von [40], S. 191).

Kapitel 3

Grundlagen zur Intervallarithmetik

Die Intervallarithmetik ist ein Teilgebiet der Mathematik, mit welchem Kenngrößen als Intervalle definiert und Lösungen auf Basis dieser berechnet werden. Der grundlegende Gedanke zur Nutzung der Intervallarithmetik in der Ökobilanzierung ist es, durch die Definition von Intervallen unsichere Bereiche von Kenngrößen vollständig zu berücksichtigen. Dies betrifft alle Werte, die nicht exakt bestimmbar sind - was in der Realität der Regelfall ist. Die Grundsätze der Intervallarithmetik werden ausführlich u. a. in der Literatur [44], [45], [46], [47], [48], [49], [50] beschrieben. Das in dieser Arbeit entwickelte Berechnungsverfahren beruht auf der Intervallarithmetik. Im Folgenden werden diejenigen Aspekte der Intervallarithmetik thematisiert, welche im Zuge des in dieser Arbeit entwickelten und implementierten Berechnungsverfahrens als relevant zu betrachten sind. Hierfür werden in Kapitel 3.1 Intervallzahlen thematisiert und in Kapitel 3.2 die einfache Intervallarithmetik von der erweiterten Intervallartihmetik abgegrenzt. In Kapitel 3.3 werden relevante Intervall-Funktionen vorgestellt und in Kapitel 3.4 analog relevante Vergleichsoperationen. Wesentliche Intervall-Matrizen werden in Kapitel 3.5 behandelt. Aspekte zu Intervall-Gleichungssystemen werden schließlich in Kapitel 3.6 erläutert. In diesem Zusammenhang werden auch in Kapitel 3.7 diverse Berechnungsverfahren zur Lösung von Intervall-Gleichungssystemen erläutert.

3.1 Intervallzahlen und Rechengesetze

Eine Intervallzahl wird in dieser Arbeit als $x^{\mathrm{I}} = [\underline{x}, \overline{x}]$ dargestellt und repräsentiert ein geschlossenes Intervall (vgl. z. B. [48], S. 5 ff.), wobei $\underline{x}$ die Untergrenze des Intervalls darstellt und $\overline{x}$ die Obergrenze. Die mit reellen Zahlen gebildeten Intervalle sowie die einhergehenden Verknüpfungen werden Intervallräume bezeichnet und im Folgenden als $\mathbb{IR}$ abgekürzt. Es handelt sich um geordnete Paare, womit $\underline{x} \leq \overline{x}$ gilt. Die Intervallzahl x^{I} beschreibt somit eine Menge von all denjenigen reellen Zahlen, welche Teil des Intervalls sind:

$$x^{\mathrm{I}} = \{x \mid \underline{x} \leq x \leq \overline{x}\},\ x^{\mathrm{I}} \in \mathbb{IR} \tag{3.1}$$

Bei $\underline{x} = \overline{x}$ fällt die Obergrenze mit der Untergrenze von x^{I} zusammen, was auch als Punktintervall bezeichnet wird (vgl. z. B. [47], S. 316, Anmerkung: hier wird der englische Begriff „point interval“ verwendet). Punktintervalle, deren Wert null ist, werden in dieser Arbeit als Nullintervalle bezeichnet. Die Intervalle sollen den Körper $\mathbb{R}$ unter Berücksichtigung der Unsicherheiten darstellen. In der Intervallarithmetik gilt zwar das Assoziativ- und das Kommutativgesetz, die Rechenregeln des Distributivgesetzes treffen jedoch nicht (vollständig) zu (vgl. z. B. [50], S. 12, [51], S. 91) - die Intervallarithmetik entspricht so keiner algebraischen Struktur (vgl. z. B. [51], S. 90-92).

3.2 Einfache und erweiterte Intervallarithmetik

Der symbolische Operator • deklariert eine der vier Rechenoperationen Addition, Subtraktion, Multiplikation und Division, deren Operation in der Intervallarithmetik wie folgt auf die beiden Intervallzahlen $x^{\mathrm{I}} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ und $y = [\underline{y}, \overline{y}] \in \mathbb{IR}$ angewandt wird (vgl. z. B. [48], S. 6):

$$x \bullet y = [\, min(\underline{x} \bullet \underline{y}, \underline{x} \bullet \overline{y}, \overline{x} \bullet \underline{y}, \overline{x} \bullet \overline{y}) \,, max(\underline{x} \bullet \underline{y}, \underline{x} \bullet \overline{y}, \overline{x} \bullet \underline{y}, \overline{x} \bullet \overline{y},) \,] \tag{3.2}$$

Für die Addition, Subtraktion sowie Multiplikation und Division ergeben sich konkret die folgenden Rechenoperationen (vgl. z. B. [51], S. 90):

$$[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \tag{3.3}$$

$$[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \tag{3.4}$$

$$[\underline{x}, \overline{x}] \cdot [\underline{y}, \overline{y}] = [\, min(\underline{x} \cdot \underline{y}, \underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}), max(\underline{x} \cdot \underline{y}, \underline{x} \cdot \overline{y}, \overline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y})] \tag{3.5}$$

$$\frac{[\underline{x}, \overline{x}]}{[\underline{y}, \overline{y}]} = [\underline{x}, \overline{x}] \cdot [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}] \tag{3.6}$$

Die Gleichungen 3.2 bis 3.6 sind Teil der einfachen bzw. elementaren Intervallarithmetik, bei der die Division durch Null ausgeschlossen ist, d. h. das Intervall $y^{\mathrm{I}} = [\underline{y}, \overline{y}]$ impliziert nicht die Null (vgl. z. B. [48], S. 6). In der erweiterten Intervallarithmetik ist dies zulässig; für diesen Sonderfall gibt es festgelegte Regeln (vgl. z. B. [46], S. 21). Die erweiterte Intervallarithmetik ist für die Berechnung ökologischer Bilanzierungen nicht praktikabel, da die auf dieser Basis ermittelten Ergebnisintervalle mitunter unendliche Intervallweiten aufweisen können (vgl. z. B. [47], S. 238 f.). Die Division durch Intervalle, welche die Null implizieren oder Grenzwerte von null aufweisen, ist bei dem in dieser Arbeit entwickelten Berechnungsverfahren nicht vorgesehen.

3.3 Relevante Intervall-Funktionen

Im Folgenden werden Funktionen zur Anwendung auf Intervalle erläutert, die für das Verständnis der hier vorgestellten Arbeit als relevant zu betrachten und im Zuge der Entwicklung des intervallbasierten Berechnungsverfahrens verwendet worden sind.

3.3.1 Mittelpunkt eines Intervalls

Der Mittelpunkt $\check{x} = m(x)$ eines Intervalls $x^{\mathrm{I}} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ wird definiert als (vgl. z. B. [50], S. 10):

$$m(x) = \check{x} = \frac{\underline{x} + \overline{x}}{2} \tag{3.7}$$

3.3.2 Weite eines Intervalls

Die Weite $w(x)$ eines Intervalls $^{\mathrm{I}} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ wird definiert als (vgl. z. B. [46], S. 23):

$$w(x) = \overline{x} - \underline{x} \tag{3.8}$$

3.3.3 Radius eines Intervalls

Der Radius $rad(x)$ eines Intervalls $^{\mathrm{I}} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ wird definiert als (vgl. z. B. [48], S. 5):

$$rad(x) = \frac{\overline{x} - \underline{x}}{2} \tag{3.9}$$

3.3.4 Mignitude eines Intervalls

Die Mignitude $\langle x \rangle$ eines Intervalls $^{\mathrm{I}} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ ist das betragsmäßige Minimum der beiden Intervallgrenzen (vgl. z. B. [48], S. 6).

3.3.5 Magnitude eines Intervalls

Die Magnitude $|x|$ eines Intervalls $^{\mathrm{I}} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ ist das betragsmäßige Maximum der beiden Intervallgrenzen (vgl. z. B. [48], S. 6).

3.4 Relevante Vergleichsoperatoren

Der Vergleich von Intervallen weist Besonderheiten auf. Daher werden die Vergleichsoperatoren im Folgenden erläutert (vgl. [48], S. 6):

- „Kleiner als“-Vergleichsoperation: Es gilt $x^{\mathrm{I}} < y^{\mathrm{I}}$, wenn $\overline{x} < \underline{y}$.
- „Größer als“-Vergleichsoperation bei Intervallen: Es gilt $x^{\mathrm{I}} > y^{\mathrm{I}}$, wenn $\underline{x} > \overline{y}$.
- „Gleich“-Vergleichsoperation: Es gilt $x^{\mathrm{I}} = y^{\mathrm{I}}$, wenn $\underline{x} = \underline{y}$ und $\overline{x} = \overline{y}$.
- „Kleiner gleich“-Vergleichsoperation: Es gilt $x^{\mathrm{I}} \leq y^{\mathrm{I}}$, wenn $\overline{x} = \underline{y}$.
- „Größer gleich“-Vergleichsoperation: Es gilt $x^{\mathrm{I}} \geq y^{\mathrm{I}}$, wenn $\underline{x} = \overline{y}$.

3.5 Definition relevanter (Intervall-)Matrizen

Im Folgenden werden spezielle (Intervall-)Matrixtypen erläutert, die für das Verständnis der hier vorgestellten Arbeit als relevant zu betrachten sind.

3.5.1 A^{I}-Matrix

Die Intervallmatrix $\mathrm{A}^{\mathrm{I}} \in \mathbb{IR}^{nxn}$ mit den Komponenten $a^{\mathrm{I}}_{ij} = [\underline{a}_{ij}, \overline{a}_{ij}] \in \mathbb{IR}$ kann definiert werden als (vgl. z. B. [48], S. 77 f.):

$$\mathrm{A}^{\mathrm{I}} = [\underline{\mathrm{A}}, \overline{\mathrm{A}}] \tag{3.10}$$

Sie stellt eine Koeffizientenmatrix dar, deren Koeffizienten Intervalle sind.

3.5.2 C-Matrix

Die Matrix $C \in \mathbb{R}^{nxn}$ ergibt sich aus den Mittelpunkten $m(a^I_{ij})$ der Komponenten $a^I_{ij} = [\underline{a}_{ij}, \overline{a}_{ij}] \in \mathbb{IR}$ einer Intervallmatrix $A^I = [\underline{A}, \overline{A}] \in \mathbb{IR}^{nxn}$ (vgl. z. B. [52], S. 1494):

$$C(A^I) = m(A^I) \tag{3.11}$$

3.5.3 Z-Matrix

Die Intervallmatrix $A^I \in \mathbb{IR}^{nxn}$ ist eine Z-Matrix, wenn alle $\overline{A} \in A^I$ einer Z-Matrix (Definition von Z-Matrix vgl. Kapitel 2.1.7) entsprechen (vgl. z. B. [53], S. 878).

3.5.4 P-Matrix

Die Intervallmatrix $A^I \in \mathbb{IR}^{nxn}$ ist eine P-Matrix, wenn alle $A \in A^I$ einer P-Matrix (Definition von P-Matrix vgl. Kapitel 2.1.8) entsprechen (vgl. z. B. [54], S. 33).

3.5.5 M-Matrix

Die Intervallmatrix $A^I \in \mathbb{IR}^{nxn}$ ist eine M-Matrix, wenn alle $A \in A^I$ einer M-Matrix (Definition von M-Matrix vgl. Kapitel 2.1.9) entsprechen (vgl. [55], S. 155).

3.5.6 H-Matrix

Die Intervallmatrix $A^I \in \mathbb{IR}^{nxn}$ ist eine H-Matrix (Definition von H-Matrix vgl. Kapitel 2.1.10), wenn ihre Vergleichsmatrix $\langle A^I \rangle$ (Definition von Vergleichsmatrix vgl. Kapitel 3.5.7) eine M-Matrix ist (vgl. z. B. [48], S. 111).

3.5.7 Vergleichsmatrix $\langle A^I \rangle$

Die Vergleichsmatrix $\langle A^I \rangle$ (englisch: „comparison matrix“) einer Intervallmatrix $A^I \in \mathbb{IR}^{nxn}$ ist eine Matrix, die wie folgt definiert werden kann (vgl. z. B. [56], S. 1290):

$$\langle A^I \rangle = \alpha_{ij},\ mit: \begin{cases} \alpha_{ij} = \langle A_{ij} \rangle\ \forall\ i = j \\ \alpha_{ij} = -|A_{ij}|\ \forall\ i \neq j \end{cases} \tag{3.12}$$

Die Vergleichsmatrix $\langle A^I \rangle$ enthält keine Intervalle, sondern die Untergrenzwerte der intervallbasierten Vergleichsmatrix $M(A^I)$. A^I wird den H-Matrizen zugeordnet, wenn $\langle A^I \rangle$ einer M-Matrix entspricht (vgl. [56], S. 1290).

3.5.8 Inverse

Die Inverse $inv(A^I) = B^I$ der Intervallmatrix $A^I \in \mathbb{IR}^{nxn}$ stellt ebenfalls eine Intervallmatrix dar (vgl. [57], S. 864). Die Inverse $inv(B^I)$ der Inversen B^I einer Intervallmatrix A^I stellt nicht die Intervallmatrix A^I dar.

3.6 Intervall-Gleichungssysteme

Lineare Intervall-Gleichungssysteme mit der Intervallmatrix $\mathrm{A}^{\mathrm{I}} = [\underline{\mathrm{A}}, \overline{\mathrm{A}}] \in \mathbb{IR}^{nxn}$ und dem Intervallvektor $b^{\mathrm{I}}=[\underline{b}, \overline{b}] \in \mathbb{IR}^n$ in der Form:

$$\mathrm{A}^{\mathrm{I}} x = b^{\mathrm{I}} \tag{3.13}$$

bestehen aus der Menge aller linearen Gleichungssysteme (vgl. z. B. [46], S. 85, [58]); eine Lösung der Gleichungssysteme wird daher durch eine Lösungsmenge s definiert:

$$s = \{x \mid \exists\, \mathrm{A} \in \mathrm{A}^{\mathrm{I}}, b \in b^{\mathrm{I}} : \mathrm{A}\, x = b\} \tag{3.14}$$

3.6.1 Hülle der Lösungsmenge und Oberhüllen

Diese Lösungsmenge s enthält alle möglichen Lösungen, ist aber „im Allgemeinen nicht bestimmbar“ (vgl. [44], S. 44). Stattdessen wird daher der Intervallvektor $x^{\mathrm{I}} \in \mathbb{IR}^n$ gesucht. Dieser „umhüllt“ einen größeren Lösungsbereich, als in der tatsächlichen Lösungsmenge s enthalten sind. D. h. es gilt:

$$x^{\mathrm{I}} \supset s \tag{3.15}$$

Der Lösungsvektor $x^{\mathrm{I}} \in \mathbb{IR}^n$, dessen Intervallweiten für x_i mit i von 1 bis n minimal werden, wird auch als sog. „hull of the solution set“ ([46], S. 103 f.) bezeichnet, was im Deutschen „Hülle der Lösungsmenge“ (vgl. z. B. [55], S. 151) genannt wird. Doch auch die Hülle der Lösungsmenge s zu finden, wird den NP-schweren Problemen zugeordnet (vgl. z. B. [59], [60]). Daher wird häufig auf Verfahren zurückgegriffen, die nicht auf das Finden der Hülle der Lösungsmenge abzielen, sondern auf Lösungen, welche diese umhüllen. Diese werden mitunter auch als Oberhüllen bezeichnet (vgl. [44], S. 17 f., [61], S. 234). Bei Oberhüllen spielt das im folgenden Kapitel 3.6.2 erläuterte Problem zu weiter (Ergebnis-)Intervalle durch sog. „abhängige Intervalle“ eine Rolle.

3.6.2 Problem abhängiger Intervalle

Ein Kernproblem in der Intervallarithmetik ist das Problem mit „abhängigen Intervallen“. Ebendieses Problem beschreibt ein zu großes Aufweiten von (Ergebnis-)Intervallen (vgl. z. B. [44], S. 20 ff.). Dieses Problem tritt auf, wenn Intervalle mehrfach eingesetzt werden und ist darauf zurückzuführen, dass in der Intervallarithmetik das Grundrechengesetz der Distributivität nicht vollständig zutrifft (vgl. z. B. [51], S. 90 ff.). Des Weiteren wird der Einhüllungseffekt (engl. „Wrapping-Effect“) bei nichtlinearen Gleichungen als problematisch beschrieben (vgl. z. B. [46], S. 266). Dieser Effekt besagt, dass der Intervallvektor neben der Lösungsmenge s weitere Bereiche umhüllt (vgl. Kapitel 3.6).

3.7 Numerische Lösungsverfahren

Es existieren verschiedene Methoden,mit denen der Intervallvektor x^{I} berechnet werden kann. Dabei gibt es auch Verfahren, mit denen für bestimmte - mathematisch klassifizierte - Matrixtypen die Hülle der Lösungsmenge berechnet werden kann. Manche der in dieser Arbeit vorgestellten Verfahren bedienen sich der Präkonditionierung. Es existieren auch Verfahren, mit denen über die Inverse die intervallbasierten Gleichungssysteme gelöst werden.

Im Folgenden werden einige Verfahren diskutiert, die aus der Literatur [46], [52], [56], [62], [63] stammen und dort jeweils ausführlicher behandelt werden. Daneben wird der Intervall-Gauß-Algorithmus und die intervallba-

sierte Inversenberechnung über die adjunkten Matrix erläutert. In dieser Arbeit werden iterative Verfahren nicht weiter behandelt. Der Grund hierfür ist, dass der iterative Ansatz auf intervallarithmetischer Ebene in dieser Arbeit als ungeeignet anzunehmen ist, da die Ergebnisintervalle bei Anwendung iterativer Methoden große Intervallweiten aufweisen können (vgl. z. B. [44], S. 50 f., [64], S. 102 f., [65], S. 539). Dies steht im Widerspruch zu dem in dieser Arbeit verfolgten Ziel.

3.7.1 Intervall-Gauß-Algorithmus

Eine Möglichkeit, das Intervall-Gleichungssystem A^{I} zu lösen, basiert auf dem Gaußschen Eliminationsverfahren (Erläuterung zum Gaußschen Eliminationsverfahren vgl. Kapitel 2.4.2), welches auf die intervallarithmetische Ebene übertragen werden kann. Der Intervall-Gauß-Algorithmus ist bei der Lösung von intervallbasierten Gleichungssystemen weitläufig bekannt (vgl. z. B. [46], S. 89-91, [66], S. 408), es werden aber im Allgemeinen nicht die Hülle der Lösungsmenge s ermittelt, sondern Oberhüllen (Definition von Oberhülle vgl. auch. Kapitel 3.6.1). Grund hierfür können „abhängige Intervalle“ sein (vgl. z. B. [46], S. 90).

Bei Anwendung des Intervall-Gauß-Algorithmus auf intervallbasierte Gleichungssysteme gibt es keine eindeutige Lösung. So kann das Ergebnis des Lösungsvektors x^{I} durch vorherige Matrixvertauschungen und/oder Pivotisierung beeinflusst werden. Bis heute ist eine optimale Methodik zur Pivotisierung für Intervallmatrizen nicht bekannt (vgl. z. B. [67]). Die Intervall-Gauß-Elimination kann u. a. durchgeführt werden, wenn $\mathrm{A} \in \mathrm{A}^{\mathrm{I}}$ den H-Matrizen angehört (vgl. z. B. [66], S. 414 ff., [48], S. 152).

3.7.2 Intervall-Adjunkten-Verfahren

Intervall-Gleichungssysteme können auch über die Inversen mit der intervallbasierten adjunkten Matrix gelöst werden. Das lineare Verfahren zur Berechnung der Adjunkten kann auf Intervallmatrizen übertragen werden (vgl. z. B. [63], S. 616 f.). Die Ergebnisse des Lösungsvektors x^{I} können durch Zeilenvertauschungen der Intervallmatrix A^{I} beeinflusst werden.

3.7.3 Verfahren von Beeck

Stellt die Intervallmatrix $\mathrm{A}^{\mathrm{I}} = [\underline{\mathrm{A}}, \overline{\mathrm{A}}]$ eine M-Matrix dar und ist damit invers-positiv, ist die Berechnung der Hülle mit der rechten Seite b wie folgt möglich (vgl. [61], S. 110 f., [56], S. 1291):

$$x^{\mathrm{I}} = \begin{cases} [\overline{\mathrm{A}}^{-1} \cdot \underline{b},\ \underline{\mathrm{A}}^{-1} \cdot \overline{b}], \ f\ddot{\mathrm{u}}r\ b \geq 0 \\ [\underline{\mathrm{A}}^{-1} \cdot \underline{b},\ \underline{\mathrm{A}}^{-1} \cdot \overline{b}], \ f\ddot{\mathrm{u}}r\ b \in 0 \\ [\underline{\mathrm{A}}^{-1} \cdot \underline{b},\ \overline{\mathrm{A}}^{-1} \cdot \overline{b}], \ f\ddot{\mathrm{u}}r\ b \leq 0 \end{cases} \tag{3.16}$$

3.7.4 Verfahren von Hansen

Hansen veröffentlichte 1992 ein Verfahren zur Berechnung der Hülle der Lösungsmenge (vgl. auch [52]) durch Präkonditionierung der Matrix (Präkonditionierung vgl. auch Kapitel 2.4.4), welches er später noch modifizierte (vgl. auch [64]). Zur Präkonditionierung wird die Inverse B der Mittelpunkts-Matrix $\mathrm{C} \in \mathrm{A}^{\mathrm{I}}$ verwendet:

$$\mathrm{B} = m(\mathrm{A}^{\mathrm{I}})^{-1} \tag{3.17}$$

Die präkonditionierte Gleichung lautet:

$$\mathrm{M}^{\mathrm{I}} \cdot x = r^{\mathrm{I}},\ mit: \begin{cases} \mathrm{M}^{\mathrm{I}} = \mathrm{B}\mathrm{A}^{\mathrm{I}} \\ r^{\mathrm{I}} = \mathrm{B}b^{\mathrm{I}} \end{cases} \tag{3.18}$$

mit $\mathrm{M}^{\mathrm{I}} = [\underline{\mathrm{M}}, \overline{\mathrm{M}}]$ sowie $r^{\mathrm{I}} = [\underline{r}, \overline{r}]$. Dabei repräsentiert $\underline{\mathrm{M}}$ eine M-Matrix, womit deren Inverse $\mathrm{P} = \underline{\mathrm{M}}^{-1}$ somit positiv ist (vgl. [52], S. 1499). Die Obergrenze $\overline{x}$ des Intervallvektors x^{I} berechnet sich daraus wie folgt:

$$\overline{x} = \underline{\mathrm{M}}^{-1}\overline{r} \tag{3.19}$$

Die Untergrenzwerte $\underline{x}_i$ des Intervallvektors x^{I} berechnen sich wie folgt:

$$\underline{x_i} = \begin{cases} c_i z_i,\ wenn\ z_i > 0 \\ z_i,\ wenn\ z_i \leq 0 \end{cases} \tag{3.20}$$

Dabei sind c_i sowie z_i mit e_i^{T} (wobei e_i^{T} die i-te und transponierte Spalte der Einheitsmatrix E darstellt):

$$c_i = \frac{1}{2 \cdot \mathrm{P}_{ii} - 1} \tag{3.21}$$

$$z_i = (\underline{\mathrm{M}}_i x\ +\ \overline{\mathrm{M}}_i x)\mathrm{P}_{ii}\ -\ e_i^{\mathrm{T}}\mathrm{P}\overline{r}. \tag{3.22}$$

Die Methode wurde im Jahr 2000 modifiziert. Dies betrifft die Bestimmung exakter und nicht nur angenäherter Grenzwerte der M-Matrix, als auch die Beschreibung eines effizienteren, praktischen Verfahrens (vgl. [64], S. 98 f.). Die Untergrenzwerte der M-Matrix werden wie folgt definiert:

$$\underline{\mathrm{M}}_{ij} \qquad \begin{cases} min[\underline{\mathrm{M}}_{ij}, -\overline{\mathrm{M}}_{ij}],\ wenn\ i \neq j \\ min[\underline{\mathrm{M}}_{ij}, 2 - \overline{\mathrm{M}}_{ij}],\ wenn\ i = j. \end{cases}$$

3.7.5 Verfahren von Rohn

Die Berechnung des Intervallvektors x^{I} baut auf der Präkonditionierung mit der Inverse $\mathrm{P} = \underline{\mathrm{M}}^{-1}$ sowie dem Parameter c_i der Hansen-Methode auf (Verfahren von Hansel vgl. Kapitel 3.7.4). Der Intervallvektor x^{I} kann wie folgt berechnet werden (vgl. [68], S. 15 und [56], S. 1292):

$$x^{\mathrm{I}} = [\underline{x}, \overline{x}] \qquad \begin{cases} \underline{x} = min\{x_u,\ c_i \cdot x_u\} \\ \overline{x} = max\{x_o,\ c_i \cdot x_o\} \end{cases} \tag{3.23}$$

Die Variablen x_u und x_o ergeben sich zu:

$$x_u = -\dot{x} + \mathrm{P}_{ii} \cdot (\check{r} + |\check{r}|) \tag{3.24}$$

$$x_o = +\dot{x} + \mathrm{P}_{ii} \cdot (\check{r} - |\check{r}|) \tag{3.25}$$

Dabei ist $\dot{x}$ definiert als:

$$mit\ \dot{x} = (\mathrm{P} \cdot (|\check{r}| + rad(r)))_i \tag{3.26}$$

Die theoretische Methode stellt eine Verbesserung der Hansen-Methode dar, da nur eine Matrix invertiert werden muss (vgl. [68], S. 14), hinsichtlich der numerischen Umsetzung sind beide Methoden etwa gleich effizient (vgl. [64], S. 96). Auch sind die Ergebnisse beider numerischen Algorithmen gleich.

3.7.6 Verfahren von Ning

Ning et al. diskutierten 1997 verschiedene Berechnungsverfahren (vgl. [56]). Dabei wurden verschiedene Theoreme aufgestellt. Ein Theorem bezieht sich auf H-Matrizen (vgl. auch Theorem 2.2 in [56], S. 1294 ff.). Wenn die Intervallmatrix $\mathrm{A}^{\mathrm{I}} = [\underline{\mathrm{A}}, \overline{\mathrm{A}}]$ den H-Matrizen zugeordnet werden kann und damit ihre Vergleichsmatrix $\langle \mathrm{A}^{\mathrm{I}} \rangle$ invers-positiv ist, kann die Berechnung von x^{I} nach [56] wie folgt berechnet werden:

$$x^{\mathrm{I}} = \frac{b_i + [-\beta_i, \beta_i]}{\mathrm{A}_{ii} + [-\alpha_i, \alpha_i]} \tag{3.27}$$

Dabei werden α_i und β_i definiert zu:

$$\alpha_i = \langle \mathrm{A}_{ii} \rangle - \frac{1}{d_i} \tag{3.28}$$

$$\beta_i = \frac{u_i}{d_i} - |b_i| \tag{3.29}$$

Die Parameter u und d_i werden beschrieben als:

$$u = \langle \mathrm{A}^{\mathrm{I}} \rangle^{-1} |b| \tag{3.30}$$

$$d_i = (\langle \mathrm{A}^{\mathrm{I}} \rangle^{-1})_{ii} \tag{3.31}$$

Der Algorithmus weist nach [56] einen geringen Rechenzeitaufwand auf, berechnet in vielen Fällen jedoch Oberhüllen, die weiter sind als die des Intervall-Gauß-Verfahrens. Das von Ning et al. vorgestellte Theorem 2.5 (vgl. [56], S. 1298 f.) basiert auf dem Verfahren von Beeck (Verfahren von Beeck vgl. Kapitel 3.7.3) und ist auf M-Matrizen anwendbar, das Theorem 2.6 (vgl. [56], S. 1299 f.) gilt verallgemeinernd bei invers-positiven Matrizen. Das Theorem 2.6 lautet wie folgt (vgl. [56], S. 1299):

$$\mathrm{A}^{(1)}_{ik} = \overline{\mathrm{A}}_{ik},\ falls\ \underline{x}_k \geq 0\ und\ \mathrm{A}^{(1)}_{ik} = \underline{\mathrm{A}}_{ik},\ falls\ \underline{x}_k < 0 \tag{3.32}$$

$$\mathrm{A}^{(2)}_{ik} = \overline{\mathrm{A}}_{ik},\ falls\ \overline{x}_k \leq 0\ und\ \mathrm{A}^{(2)}_{ik} = \underline{\mathrm{A}}_{ik},\ falls\ \overline{x}_k > 0 \tag{3.33}$$

Die Hülle der Lösungsmenge mit dem Intervallvektor x^{I} ergibt sich zu (vgl. nach [56], S. 1299):

$$x^{\mathrm{I}} = [\,(\mathrm{A}^{(1)})^{-1}\ \underline{b}, (\mathrm{A}^{(2)})^{-1}\ \overline{b}] \tag{3.34}$$

3.7.7 Verfahren von Nirmala

Im Zuge der Recherche mit den Schlagworten „Inverse“ sowie „Interval Matrix“ wurde u. a. eine im Jahr 2011 veröffentlichte Berechnungsmethode gefunden (vgl. [63]). Sie wird im weiteren Verlauf als „Verfahren von Nirmala“ bzw. „Nirmala-Methode“ bezeichnet. In der genannten Veröffentlichung werden Beispiele aufgeführt, auf Basis derer das Verfahren von Nirmala mit diversen Methoden verglichen wird. Dabei wird auch Bezug genommen auf die im vorigen Kapitel 3.7.6 vorgestellte Studie von Ning et al. (vgl. [56]). Der Studie in [63] zufolge werden mit der Nirmala-Methode Intervalle mit schmaleren Intervallweiten erzielt als bei den vorab vorgestellten Verfahren.

Kapitel 4

Grundlagen zur Modellierung realer Systeme

Um die Auswirkungen von technologischen Systemen auf ökologische Systeme berechnen zu können, bedarf es einer Modellierung dieser Systeme. In der Ökobilanzierung werden technologische Systeme modelliert, wobei auf weitere Modelle zur Beschreibung der potentiellen Umweltwirkungen dieser Systeme zurückgegriffen wird. Hierfür werden in Kapitel 4.1 diejenigen Bereiche zur Modellbildung realer Systeme näher erläutert, die in Bezug zur Methodik der Ökobilanzierung als relevant eingestuft werden. In Kapitel 4.2 werden relevante Begriffe der Graphentheorie behandelt, welche zur Darstellung mathematischer Modelle verwendet werden kann. Abschließend wird in Kapitel 4.3 auf die Modellierung in der Ökobilanzierung eingegangen.

4.1 Modellierung realer Systeme

Zur Analyse realer Systeme bedarf es Modelle, durch die abstrahiert und vereinfacht die Realität widergespiegelt werden kann. Eine umfangreiche Basis zur allgemeinen Modellbildung und Simulation ist bspw. in [69, 70, 71] zu finden. Detaillierte Informationen zur objektorientierten Modellierung technischer Systeme bietet z. B. [72]. Umfassende Informationen zur Modellierung ökologischer Systeme stellt z. B. [73] sowie [74], S. 198-206 zur Verfügung. Grundlagen zur Modellbildung im Kontext der ökologischen Bilanzierung liefert z. B. [75], S. 78-94.

4.1.1 Systeme der Öko- und Technosphäre

Systeme sind geordnete Strukturen der realen Welt, welche sich aus Komponenten und deren Beziehungen auszeichnen (vgl. z. B. [76], S. 9 f. und [69], S. 34 f.). Diese können differenziert werden in „natürliche Systeme“ und „künstliche Systeme“. „Natürliche Systeme“ sind dabei z. B. ökologische Systeme (vgl. z. B. [76], S. 12), während „künstliche Systeme“ auf menschliche Handlungen zurückzuführen sind. Ein weitgehend von der Umwelt abgrenzbarer Lebensraum, das Biotop, mit der darin lebenden Gemeinschaft von Lebewesen diverser Art, der Biozönose, wird als „Ökosystem“ bezeichnet. Die Summe aller globalen Ökosysteme bildet die „Ökosphäre“. (Definition von Biotop, Biozönose und Ökosystem vgl. z. B. [77], S. 43, S. 44 und S. 204)

Durch die Entwicklung seiner Fähigkeiten begann der Mensch aktiv seine Umwelt nach seinem Willen zu verändern, zu benutzen und umzugestalten, wie z. B. durch Waldrodung zur landwirtschaftlichen Nutzung (vgl. z. B. [78], S. 214 f.), wodurch Ökosysteme zunehmend in Technologiesysteme überführt werden. Die Technosphäre

stellt die Summe aller Technologiesysteme dar (vgl. z. B. [77], S. 289). Ihre Flüsse können vorwiegend der materiellen Technik (z. B. die Herstellung eines Produkts) oder der immateriellen Soziologie (z. B. das Unternehmen selbst) und somit den technischen oder sozialen Systemen zugeordnet werden. Die Gemeinschaft der Bewohner dieses Lebensraums werden angelehnt zur Biozönose auch als Technozönose bezeichnet (vgl. z. B. [77], S. 289).

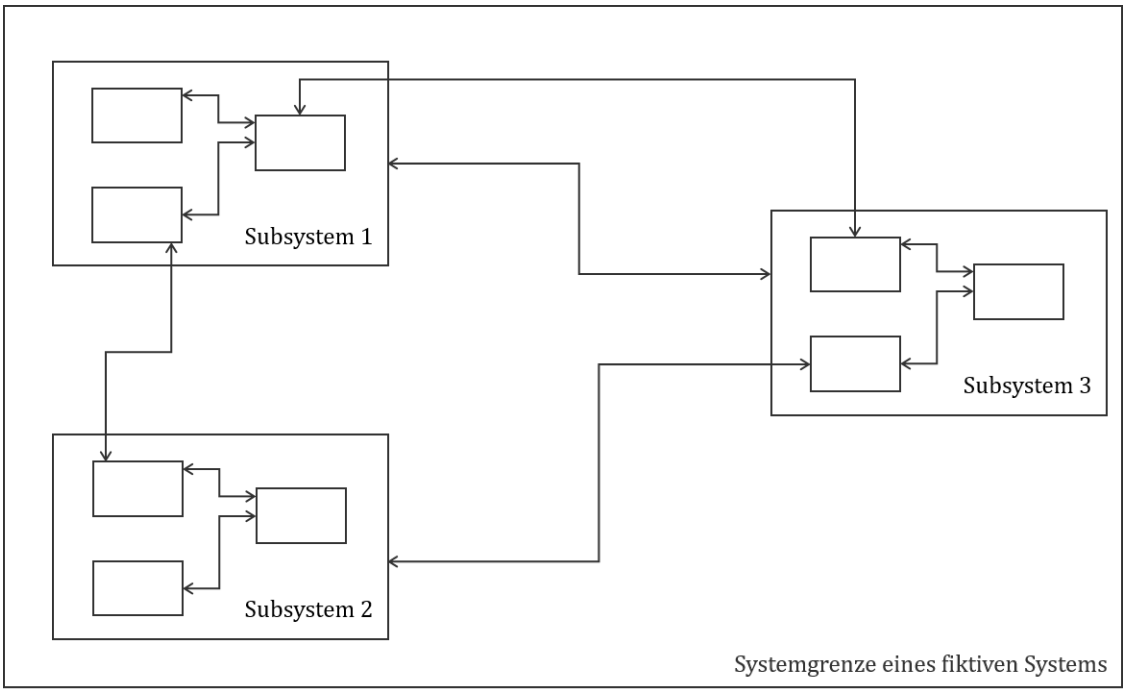


Abbildung 4.1: Fiktives System, bestehend aus drei Subsystemen, welche miteinander in Verbindung stehen.

.

In der heutigen Welt sind viele Mischsysteme vorzufinden, bei denen die Grenzen zwischen technologischem System und ökologischem System nicht mehr eindeutig gezogen werden bzw. die Festlegung dieser erschwert ist. So gibt es bspw. biotechnische Systeme, wenn sie der Nutzung von biologischen Organismen in technischen Systemen (wie bspw. die Milchproduktion mithilfe von Kühen) oder technischen Produkten in biologischen Organismen (wie bspw. ein Herzschrittmacher im menschlichen Körper) dienen. So existieren auch eine Reihe von sozioökologischen Systemen (vgl. z. B. [79], S. 19 f.). Sie umfassen sowohl weitgehend natürliche Systeme wie Schutzgebiete des Regenwaldes, als auch durch den Menschen, d. h. „künstlich “ entstandene, jedoch unter Naturschutz stehende Gebiete wie die Lüneburger Heide sowie auch stärker beeinflusste Kulturlandschaften wie Ackerflächen, Siedlungs- oder Industriegebiete.

Systeme können hinsichtlich ihrer Wechselwirkungen unterschieden werden in offene, geschlossene und abgeschlossene Systeme (Definition von abgeschlossenem System, geschlossenem System und offenen System vgl. z. B. [77], S. 1, S. 104, S. 197). Offene Systeme tauschen demnach sowohl Energie als auch Materie mit anderen Systemen bzw. ihrer Umwelt aus, geschlossene Systeme nur Energie und abgeschlossene Systeme weder Materie noch Energie. In der Regel sind Ökosysteme wie auch Technologiesysteme offene Systeme, d. h. neben den internen Stoff- und Energiekreisläufen innerhalb des Systems gibt es über dessen Grenzen hinweg Energie- und Stoffflüsse zu anderen Systemen, durch die sich die Systeme gegenseitig beeinflussen. Systeme können Subsysteme implizieren (Definition „Subsystem“ vgl. z. B. auch [77], S. 283). So stellt bspw. ein Tümpel inmitten eines Waldes ein eigenes Ökosystem dar, dieser kann ein Subsystem des Waldes sein, der wiederum ein Subsystem des Ökosystems Erde dargestellt (Anmerkung des Autors: die Erde wird z. B. in [80], S. 347 als „Ökosystem“ bezeich-

net). Ökosysteme können auch Technologiesysteme als Subsysteme beinhalten; wobei auch Technologiesysteme ökologische Subsysteme beinhalten können. In Abbildung 4.1 ist ein fiktives System dargestellt, das drei Subsysteme beinhaltet. Die drei Subsysteme stehen in Wechselwirkung zueinander, wobei das übergeordnete System als abgeschlossenes System dargestellt ist.

4.1.2 Stoff- und Energiekreisläufe

Die Biozönose eines Ökosystems kann unterschieden werden in Produzenten, Konsumenten und Destruenten. Produzenten wie Pflanzen erzeugen aus energiearmen, anorganischen Stoffen energiereiche, organische Substanzen, welche Konsumenten - wie bspw. Menschen – nutzen. Destruenten formen diese Stoffflüsse zu Kreisläufen, indem sie die organischen „Abfälle“ wieder in anorganische Substanzen überführen (vgl. z. B. [81], S. 298, [82], S. 233 sowie sowie [77], S. 204 f.); mit diesen Kreisläufen verbunden ist der Energiefluss. In Abbildung 4.2 ist vereinfacht der Stoffkreislauf und die Wechselwirkungen zwischen Produzenten, Konsumenten und Destruenten dargestellt. Der dargestellte Kreislauf entspricht vereinfacht auch dem Kohlenstoffkreislauf.

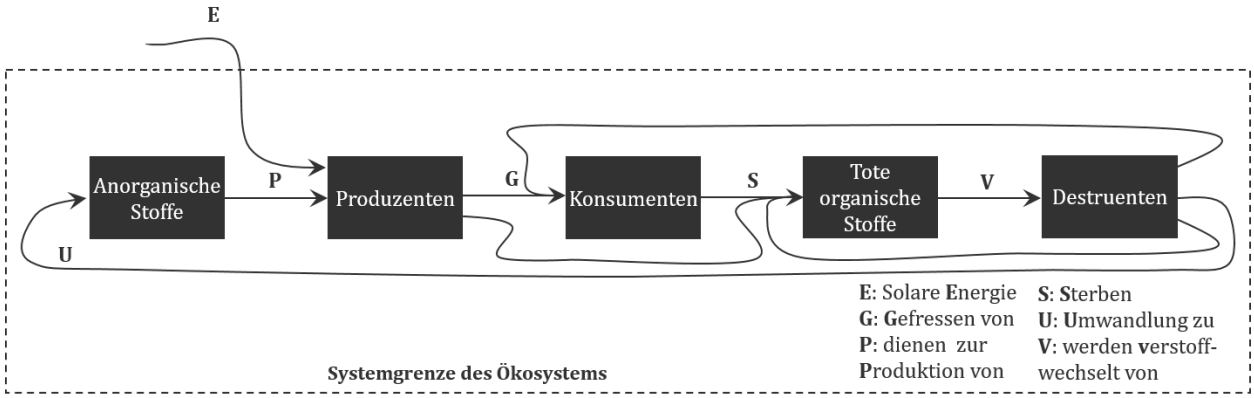


Abbildung 4.2: Energiefluss und Kohlenstoffkreislauf eines Ökosystems, angelehnt an [81], S. 298, [82], S. 233, [77], S. 205.

.

Funktionierende Ökosysteme sind in der Lage, durch das Zusammenspiel von Produzenten, Konsumenten und Destruenten ihre kreisläufigen Eigenschaften beizubehalten, um sich selbst zu regulieren, wobei ihr Zustand sich nicht signifikant ändert (vgl. Definition von „Selbstregulation“ z. B. in [77], S. 265). Die Flüsse in Technologiesystemen sind im Gegensatz zu funktionierenden Ökosystemen (noch) nicht (vollständig) kreisläufig, sondern linear, da es ihnen im Verhältnis zu den anfallenden Emissionen und Reststoffen an Destruenten mangelt, welche die entstandenen Emissionen binden und erzeugte Abfälle in abiotische Stoffe umwandeln. Durch die Kreislaufwirtschaft soll das Prinzip der Ökosysteme von den technologischen Systemen adaptiert werden (vgl. z. B. [83], S. 147 f. sowie Definition von „Stabilität“ in [77], S. 273 f.).

4.1.3 Steuerung von Systemen

Die materiellen und energetischen Kreisläufe von Systemen werden über Informationen gesteuert (vgl. auch „Ökosystem“ in [77], S. 204 f.), welche zwischen den Organismen mittels chemischer Stoffe übertragen werden (vgl. z. B. „chemische Informationsübertragung“ in [77], S. 55). Es ist daher anzunehmen, dass die Informationsübertragung in ökologischen Systemen weitgehend parallel zu materiellen und energetischen Flüssen verläuft (vgl. auch Stoffflüsse und Informationsflüsse in Abb. 41 in [77], S. 205). In anthropogenen Systemen nimmt der Informationsfluss

eine zunehmend bedeutende Rolle ein, wobei dieser Informationsfluss mitunter auch als der bedeutendste Fluss im heutigen Zeitalter des „Dataismus“ bezeichnet wird (vgl. [84], S. 563 ff.). Die Digitalisierung hat zur Folge, dass der Rohstoff- und Energieverbrauch zur Abnahme, Speicherung, Aufnahme und Verarbeitung von Informationen zunehmend komplexer verläuft. Beispielsweise kann ein Dokumentarfilm über ein Smartphone durch einen Streaming-Anbieter übermittelt werden. Hierfür ist Energie für das Betreiben des Smartphones notwendig. Ein relevanter Anteil des Energieverbrauchs ist für den Nutzer des Smartphones vermutlich weniger ersichtlich; dieser Anteil des Energieverbrauchs ist beim Betrieb des Netzwerks und des Rechenzentrums des Streamingdienstes vorzufinden. Insgesamt wurde in einer Studie abgeschätzt, dass Endgeräte, Rechenzentren und Netze 2020 für ca. 1,8 Prozent bis 3,2 Prozent der globalen Treibhausgasemissionen verantwortlich sind; Rechenzentren und Netze tragen hierzu ca. ein Fünftel bei (vgl. [85], S. 45). Zur vollständigen Abbildung technologischer Systeme ist es daher wichtig, auch den Informationsfluss, d. h. die Kommunikation zwischen den Systemen, zu berücksichtigen.

4.1.4 Herangehensweise zur Modellbildung

Die Modellbildung realer Systeme kann in verschiedene Teilschritte untergliedert werden. Eine vereinfachte Vorgehensweise hierzu ist in Abbildung 4.3 dargestellt. Zunächst ist die Basis eines Modells durch ein entsprechendes Konzept festzulegen. Die Systemstruktur von Ökosystemen kann z. B. als Wirkungsgraph erfasst werden (vgl. z. B. [73], S. 10 ff.). Bei technologischen Systemen können zunächst die relevanten Prozesse und deren Abfolge identifiziert und als Prozessmodule und Flüsse dargestellt werden.

Abbildung 4.3: Darstellung der vereinfachten Vorgehensweise bei der Modellbildung.

Auch die mathematische Formulierung der Modelle lässt mehrere Möglichkeiten zu. Grundlegend kann zwischen statischen und dynamischen Modellen unterschieden werden. Mit ersteren kann der Systemzustand vor und nach Störungen beschrieben werden, letztere ermöglichen auch die Beschreibung während dem Einwirken der Störungen (vgl. z. B. [72], S. 16). Statische Modelle weisen somit konstante Eigenschaften auf, dynamische Modelle verfügen über veränderliche Eigenschaften. Es gibt verschiedene Ansätze, ein System und dessen Beziehungen zwischen oder innerhalb von Systemen zu beschreiben. Festgelegte, wachsende oder zeitabhängige Vorgänge lassen sich bspw. mit (Differential-)Gleichungen beschreiben; stochastische Prozesse mit Wahrscheinlichkeitsverteilungen, Übergänge von Zuständen zu Aktionen mit der Automatentheorie und netzartige Strukturen mit der Graphentheorie (vgl. z. B. [71], S. 1-12). Die numerische Übertragung und Implementierung der Modelle (vgl. auch Abbildung 4.3, „Computermodell“) stellt ein hilfreiches Werkzeug zur Modellierung sowie Analyse und Bewertung von Systemen dar.

4.1.5 Schnittstellen von Modellen

Eine Schnittstelle ist ein Objekt, das auf Informationen diverser Modelle zugreifen kann. Sie dient zur Kommunikation und Verknüpfung verschiedener Modelle. In Abbildung 4.4 sind exemplarisch das System A, System B und System C dargestellt. Informationen können über Schnittstellen an den Systemen übermittelt und weitergegeben werden und z. B. in einem Modell zur Anwendung kommen.

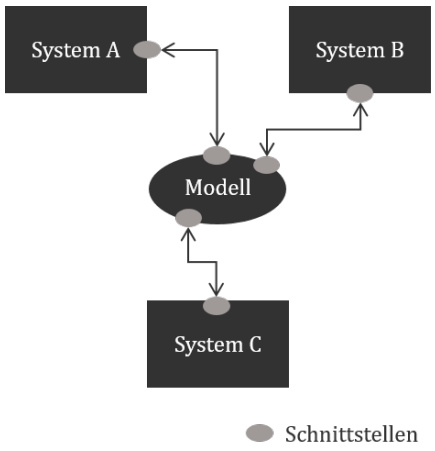
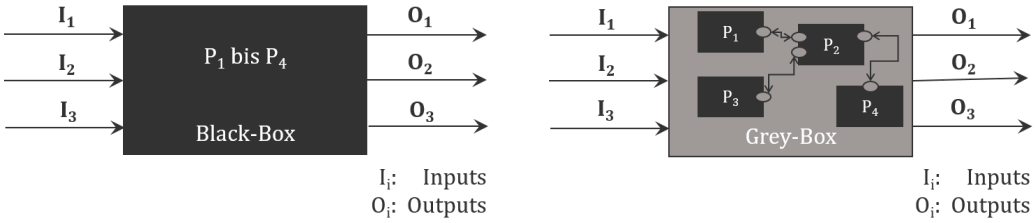


Abbildung 4.4: Schnittstelle als Kommunikations- und Verbindungsmittel von Systemen angelehnt an [72], S. 12.

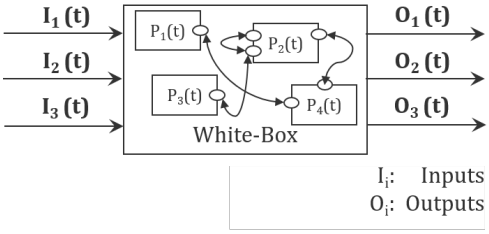
4.1.6 Box-Modelle

Systeme können als Boxen betrachtet werden, wobei unterschiedliche Boxmodelle möglich sind. So kann zwischen Black-, White- oder Grey-Box-Modellen differenziert werden (vgl. [72], S. 17). Dies hängt u. a. von der Komplexität, dem Informationsgrad oder dem Anwendungszweck der Modelle ab. Als White-Boxen werden Modelle bezeichnet, bei denen auch die innere Systemstruktur und das dynamische Verhalten bekannt sind und diese nachgebildet werden. Dieses Systemverständnis ermöglicht es, dass Auswirkungen durch Systemänderungen simuliert werden können. Bei Black-Box-Modellen hingegen ist die innere Systemstruktur und Wechselwirkungen zwischen einzelnen Komponenten nicht bekannt. Diese Art von Modellen können Auswirkungen durch Systemänderungen nur begrenzt simulieren. Grey-Boxen stellen eine Mischform von Black- und White-Boxen dar. So ist bei dieser Art von Modellen in der Regel die Systemstruktur bekannt, es liegt aber kein vollständiges Systemverständnis vor (vgl. z. B. [72], S. 17).



(a) : Black-Box-Modell.

(b) : Grey-Box-Modell.



(c) : White-Box-Modell.

Abbildung 4.5: Diverse Modelle eines Systems mit den Prozessen P_1 bis P_4, den Inputs I_i und den Outputs O_i.

In Abbildung 4.5a ist ein System mit vier Prozessen P_1 bis P_4 dargestellt, das als Black-Box modelliert wird und drei Inputs sowie Outputs aufweist. In Abbildung 4.5b ist ein System mit vier Prozessen P_1 bis P_4 dargestellt, das als Grey-Box modelliert wird und drei Inputs sowie Outputs beinhaltet. Die Systemstruktur dieses Systems ist bekannt: so besteht es aus vier Komponenten, die miteinander in Beziehung stehen. Es liegt jedoch kein vollständiges Systemverständnis vor. In Abbildung 4.5c ist ein System mit vier Prozessen P_1 bis P_4 dargestellt, das als White-Box modelliert wird mit den zeitabhängigen Inputs $I_i(t)$ und den Outputs $O_i(t)$. Die Systemstruktur des Systems ist bekannt: so besteht es aus vier Komponenten, die miteinander in Beziehung stehen. Es liegt umfassendes Systemverständnis vor; somit ist das Verhaltens über die Zeit abbildbar.

4.2 Graphentheorie

Eine Möglichkeit zur Darstellung mathematischer Modelle ist die Verwendung von Graphen. Sie werden sowohl in der Mathematik als auch in der Informatik zur Darstellung von Strukturen verwendet. Die Grundsätze zur Graphentheorie und Repräsentationsformen von Graphen werden ausführlich z. B. in [21], S. 201-248, [37], S. 77-106, [86] und [87] beschrieben. Im Folgenden werden diejenigen Begriffe zur Graphentheorie definiert, welche im Zuge des in dieser Arbeit entwickelten und implementierten Berechnungsverfahrens als relevant zu betrachten sind.

4.2.1 Eigenschaften von Graphen(typen)

Graphen sind abstrakte Konstrukte, deren Knoten die Komponenten eines Systems repräsentieren und deren Kanten die Beziehungen des Systems untereinander beschreiben (vgl. z. B. [21], S. 201).

Gerichtete Graphen

Graphen, deren Kanten Richtungen aufweisen, werden auch als gerichtete Graphen bezeichnet (vgl. z. B. [21], S. 202 f.).

Zugeordneter ungerichteter Graph

Ein ungerichteter Graph, der dieselben Kantenverbindungen wie ein gerichteter Graph *G* aufweist, nur ohne Kantenrichtung, ist der dem Graphen *G* zugeordnete ungerichtete Graph (vgl. z. B. [89], S. 15).

Gewichtete Graphen

Graphen, deren Kanten Gewichtungsfaktoren aufweisen, werden auch als gewichtete Graphen bezeichnet (vgl. z. B. [38], S. 128 sowie [23], S. 251 f.).

Weg

Ein Weg bezeichnet die Verbindung zweier Knoten über eine oder mehrere Kanten (vgl. z. B. [38], S. 112 f.). Bei gerichteten Graphen ist die Wegrichtung über die Kantenrichtung vorgegeben (vgl. z. B. [38], S. 115 f.).

Adjazenz

Zwei Knoten sind zueinander adjazent, wenn sie über eine Kante miteinander verbunden sind (vgl. z. B. [38], S. 108 sowie [86], S. 13).

Inzidenz

Eine Kante ist inzident zu zwei Knoten, wenn sie die Knoten miteinander verbindet (vgl. z. B. [38], S. 108 sowie [86], S. 12).

Zusammenhängende Graphen

Graphen sind zusammenhängend, wenn zwischen zwei beliebigen Knoten des Graphen ein Weg existiert (vgl. z. B. [37], S. 85). Ein gerichteter Graph gilt als zusammenhängend, wenn der ihm *zugeordnete ungerichtete Graph* zusammenhängend ist.

Zyklus

Ein Weg, dessen Startknoten mit dem Endknoten zusammenfällt und aus mindestens drei Knoten besteht, wird als Zyklus bezeichnet (vgl. z. B. [21], S. 202 f. und S. 239 f.).

Grad

Der Grad eines Knoten bezeichnet die Anzahl der angrenzenden Knoten; der Grad von Knoten gerichteter Graphen wird unterschieden in Eingangsgrad und Ausgangsgrad (vgl. z. B. [21], S. 202 f.). Ersterer beschreibt die Anzahl der auf den Knoten hinweisenden Kanten, letzterer die Anzahl der vom Knoten abweisenden Kanten.

Quellen und Senken

Ein Knoten, dessen Ausgangsgrad *0* ist, wird Senke genannt; ein Knoten, dessen Eingangsgrad *0* ist, wird als Quelle bezeichnet (vgl. z. B. [86], S. 133 f.). Senken gerichteter Bäume bzw. Knoten vom Grad *1* allgemeiner Bäume werden als Blätter bezeichnet (vgl. z. B. [86], S. 109); Quellen gerichteter Bäume bzw. Bäume, von denen aus ein Weg zu jedem anderen Knoten des Baumes existiert, als Wurzel (vgl. z. B. [21], S. 244).

Bäume

Zusammenhängende Graphen sind Bäume, wenn sie keine Zyklen enthalten; ein ungerichteter Baum mit n Knoten besitzt dabei *n-1* Kanten (vgl. z. B. [86], S. 21 f. sowie [37], S. 89).

Innere Knoten

Knoten von Bäumen, die keine Blätter darstellen, werden als innere Knoten bezeichnet (vgl. z. B. [21], S. 95). Bei einer Kante von Knoten v zu Knoten w gilt Knoten v als direkter Vorgänger und Knoten w als direkter Nachfolger (vgl. z. B. [23], S. 134). Innere Knoten von Bäumen weisen nur einen direkten Vorgänger, aber mitunter mehrere direkte Nachfolger auf.

Lineare Graphen

Eine besondere Form von Bäumen stellen gerichtete Graphen dar, deren innere Knoten genau einen Vorgänger sowie Nachfolger aufweisen und damit der Eingangs- und Ausgangsgrad dieser Knoten *1* beträgt. Derlei Graphen werden in dieser Arbeit als lineare Graphen bezeichnet.

Teilgraphen

Ein Graph, der einen Teil der Knoten und der Kanten eines anderen Graphen G abbildet, wird als Teilgraph von G bezeichnet (vgl. z. B. [89], S. 11 f.).

Untergraphen

Ein Graph, der einen Teil der Knoten und der zwischen diesen Knoten existierenden Kanten eines anderen Graphen G vollständig abbildet, wird als Untergraph von G bezeichnet (vgl. z. B. [87], S. 7).

4.2.2 Komplizierte Graphen

Als kompliziert werden Graphen bezeichnet, die spezielle Kantenverbindungen aufweisen. Sie werden in dieser Arbeit unterschieden in Hypergraphen und Multigraphen.

Hypergraphen

Graphen, die mindestens eine Kante beinhalten, welche *mehr als zwei* Knoten miteinander verbindet, indem sich diese verzweigt oder zusammenfließt, können als Hypergraphen bezeichnet werden. Die sich verzweigende(n) oder zusammenfließende(n) Kante(n) werden analog als Hyperkanten bezeichnet (vgl. z. B. [90], S. 178).

Multigraphen

Graphen, in denen mindestens zwei parallel verlaufende Kanten vorliegen, d. h. wenn *mehrere* Kanten *dieselben* Knoten miteinander verbinden, werden als Multigraphen bezeichnet (vgl. z. B. [87], S. 2 sowie [91], S. 54), die parallel verlaufenden, mehrfachen Kanten analog in dieser Arbeit als Multikanten.

4.2.3 Einfache Graphen

Als einfache Graphen werden angelehnt an Fáry (vgl. [88], S. 230) Graphen genannt, die keine Multikanten aufweisen, wobei in dieser Arbeit gilt, dass einfache Graphen auch keine Hyperkanten beinhalten.

4.2.4 Tabellarische Darstellungsformen

Bekannte tabellarische Repräsentationsformen von Graphen sind Adjazenz- oder Inzidienzmatrizen, die u. a. ausführlicher in [38] S. 109-112, [23], S. 223-225, [86], S. 29-33 sowie [89], S. 18-22 beschrieben werden.

Adjazenzmatrizen

Adjazenzmatrizen stellen als sog. Knoten-Knoten-Matrix die Kantenverbindungen zwischen den Knoten eines Graphen dar. Die Elemente der Matrix sind Einsen und Nullen, wobei eine Eins eine Kantenverbindung zwischen zwei Knoten repräsentiert; bei gerichteten Graphen muss die Kantenorientierung von Knoten v zu Knoten w entsprechen (vgl. z. B. [23], S. 223 ff.).

Inzidenzmatrizen

Inzidenzmatrizen stellen als sog. Knoten-Kanten-Matrizen die Zusammenhänge zwischen Knoten und Kanten dar. Die Elemente der Matrix bestehen aus *+1*,*-1* sowie *0*, wobei *+1* die Präsenz einer vom Knoten abgehenden Kante darstellt; eine *-1* hingegen eine ankommende und eine *0* besagt, dass zwischen der entsprechenden Kante und dem zugehörigen Knoten keine Verbindung besteht (vgl. z. B. [89], S. 19 ff.)

4.3 Modellierung in der Ökobilanzierung

In der Ökobilanzierung werden die Prozesse von Technologiesystemen analysiert. Zur ökologischen Bilanzierung werden Charakterisierungsmodelle integriert, mit denen die Abschätzung der Umweltwirkungen möglich ist. Die modellierten Technologiesysteme wie auch die durch Wirkungsindikatoren modellierten Umweltwirkungen können als Graphen dargestellt werden.

4.3.1 Grundlegendes Prinzip

In der Ökobilanzierung wird das zu analysierende Technologiesystem als Produktsystem modelliert. Die Umweltwirkungen werden durch Wirkungskategorien klassifiziert und mithilfe von Wirkungsindikatoren berechnet (Definition von „Wirkungskategorien“ sowie „Wirkungsindikatoren“ vgl. auch Kapitel 5.2.5).

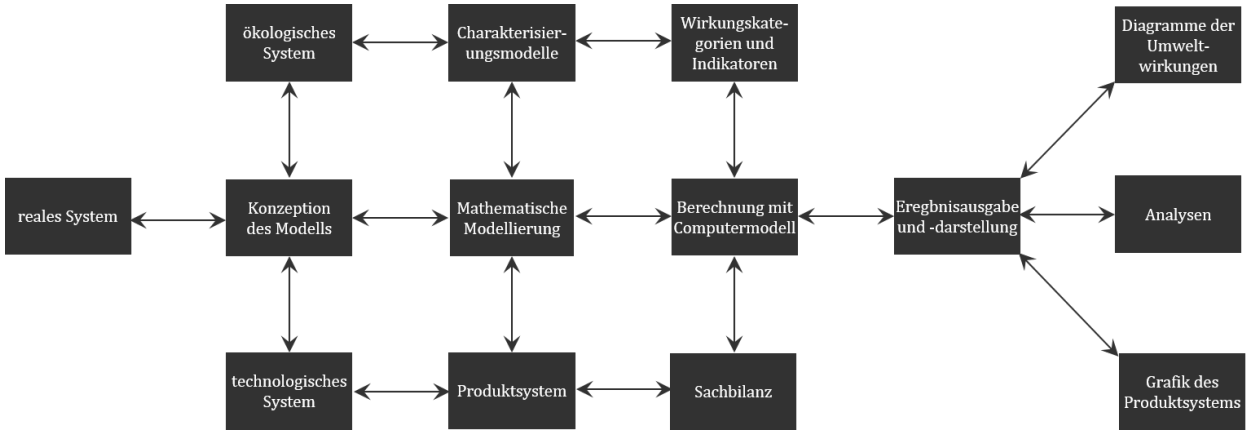


Abbildung 4.6: Modellierung in der Ökobilanzierung.

In Abbildung 4.6 ist eine vereinfachte Herangehensweise zur Modellierung in der Ökobilanzierung dargestellt. Die Ergebnisse der Produktsysteme stellen die Sachbilanz dar (Definition „Sachbilanz“ vgl. auch Kapitel 5.2.4). Auf Basis dieser Ergebnisse können in der Wirkungsabschätzung die potentiellen Umweltwirkungen abgeschätzt werden. Hierfür sind Charakterisierungsmodelle notwendig, welche in die Modelle integriert werden. Sie enthalten konstante Charakterisierungsfaktoren zur Abschätzung der Umweltwirkungen (Definition „Charakterisierungsfaktor“ und „Charakterisierungsmodell“ vgl. auch Kapitel 5.2.5). Diese werden zwar auf Basis von mitunter äußerst komplexen, dynamischen Modellen wie bspw. Klimamodellen bestimmt, die Ergebnisse fließen aber als konstante Parameter in die Modellierung ein. Die „Konsistenz“ des Produktsystems kann dann vergleichend hinsichtlich seiner potentiellen Umweltwirkungen und unter Berücksichtigung von bspw. der Sensitivität und Unsicherheit analysiert werden.

4.3.2 Normative Modellierungsansätze

Normativ werden für die Ökobilanzierung zwei grundlegende Modellierungsansätze festgelegt (vgl. [92], S. 30 f.): Zum einen können die Umweltwirkungen des Ist-Zustands eines Produktsystems bilanziert werden, welcher sich aus den erfassten Inputs und Outputs eines bestimmten Zeitpunkts ergibt. Da die Berechnung erst nach der Erfassung erfolgt, liegt dieser Zeitpunkt genau genommen in der Vergangenheit, womit eine *vergangenheitsbezogene* Abbildung des Produktsystems erfolgt. Zum anderen können mögliche, „zukünftige“ Abweichungen eines Produktsystems bilanziert werden, indem alternative Produktsysteme modelliert werden und die Bilanzierungsergebnisse miteinander verglichen werden. Dafür werden die potentiellen Abweichungen in der Regel mittels Abschätzungen und Annahmen festgelegt.

4.3.3 Klassifikation der (integrierten) Modelle

Das Konzept der Ökobilanzierung basiert auf einem linearen, statischen Modellansatz, wobei lediglich aus den Kenngrößen der „Modellhülle“ der betrachteten Systeme ein mathematisches Modell abgeleitet wird. Dieses Vorgehen wird in auch als Black-Box-Methode bezeichnet (vgl. [93], S. 29). Bei der Modellierung von Technologiesystemen werden in der Regel Stoff- und Energiebilanzen verwendet. Diese gelten für diese Systeme gemeinhin als ausreichend für eine vollständige Systembeschreibung (vgl. z. B. [94], S. 268). Zur Berechnung der potentiellen Umweltwirkungen werden geeignete Charakterisierungsmodelle integriert. Diese enthalten eine Reihe von konstanten Charakterisierungsfaktoren, welche in regelmäßigen Abständen aktualisiert werden. Zur Modellierung der Kreisläufe in Ökosystemen kommen statistische oder dynamische Modelle zum Einsatz. Die komplexe Modellierung des Klimas bspw. als Teil des Ökosystems Erde beruht auf komplexen dynamischen Klimamodellen, welche aus physikalischen Gesetzen abgeleitet werden und so auch die inneren Zusammenhänge und Wechselwirkungen der Systeme abbilden können (vgl. z. B. [95], S. 596).

Der statische Ansatz der Ökobilanzierung ist geeignet, Aussagen über den Zustand des Systems zum Zeitpunkt der Messungen oder Beobachtungen abzuleiten. Bei gleichbleibenden bzw. weitgehend konstanten Bedingungen kann er auch verwendet werden, um die Einflüsse auf die Umwelt des möglichen, zukünftigen Systems zu prognostizieren. Im Gegensatz zu dynamischen Modellen sind Modelle auf Basis dieses Ansatzes jedoch nicht geeignet, Auswirkungen bei sich stark ändernden Randbedingungen zu simulieren, da die innere Struktur des Systems, d. h. die einzelnen Systemelemente und deren Verknüpfungen, nicht abgebildet werden.

Kapitel 5

Die Methodik der Ökobilanzierung

Die Ökobilanzierung gilt als eine wissenschaftliche Methodik zur Modellierung und Abschätzung der potentiellen Umweltwirkungen technischer Systeme. Die Literaturquellen [75], [93], [96] und [97] bieten eine breite Grundlagenbasis zur ökologischen Bilanzierung; die europäische Union hat einige sehr ausführliche Handbücher zu verschiedenen Themen des „Life Cycle Assessments“ herausgegeben (vgl. z. B. [98], [99] und [100]). Die Ökobilanzierung kommt bei der Beurteilung nachhaltiger Aspekte zur Anwendung, was in Kapitel 5.1 thematisiert wird. Relevante Begriffe zur Ökobilanzierung werden in Kapitel 5.2 behandelt. Grundsätzliche Annahmen und Voraussetzungen zur ökologischen Bilanzierung folgen schließlich in Kapitel 5.3. Die zwei grundlegenden Berechnungsansätze werden in Kapitel 5.4 vorgestellt. Abschließend werden ausgewählte Datenbanken und Programme in Kapitel 5.5 aufgeführt, die zur ökologischen Bilanzierung verwendet werden können.

5.1 Einordnung der Methodik

Mit einer Ökobilanzierung können Prozesse bspw. zur Herstellung, Instandsetzung und Entsorgung von Produkten, Bauteilen oder Gebäuden von der Gewinnung der benötigten Ausgangsstoffe bis zur Aufbereitung abgebildet werden und deren potentielle Umweltwirkungen abgeschätzt werden. Ökobilanzen werden zunehmend bei der Bewertung von Nachhaltigkeit herangezogen. Daher wird in Kapitel 5.1.1 kurz auf die Ökobilanzierung im Kontext der Nachhaltigkeit eingegangen sowie in Kapitel 5.1.2 die Bewertungsgrenzen der Ökobilanzierung aufgezeigt.

5.1.1 Ökobilanzierung im Kontext der Nachhaltigkeit

Die Ökologie gilt als ein wesentlicher Baustein, wenn es darum geht, Aussagen über die Nachhaltigkeit zu treffen (Anmerkung des Autors: in dieser Arbeit wird der Begriff „Nachhaltigkeit“ synonym verwendet mit der Bezeichnung „nachhaltige Entwicklung“). Nachhaltigkeit wird häufig als interdisziplinäres Konstrukt, bestehend aus den Bereichen Ökologie, Ökonomie und Soziologie, beschrieben (vgl. z. B. [101], S. 24 f. [102], S. 682, [103], S. 99-115). Doch es gibt unterschiedliche Auffassungen von Nachhaltigkeit, und diese befinden sich zudem in einer stetigen Entwicklung. Manche Ansätze weisen in der Nachhaltigkeit einzig einen ökologischen Schwerpunkt auf, andere halten die Bereiche Ökologie, Ökonomie und Soziologie für unzureichend und ergänzen diese um weitere, wie bspw. um kulturelle Aspekte (vgl. z. B. [104], S. 27 f.).

Auch beim dreidimensionalen Modell wird der Stellenwert von ökologischen, wirtschaftlichen und sozialen Zielen diskutiert. Eine Darstellungsform der Nachhaltigkeit stellt die Ökologie, Soziologie und Wirtschaft als eigenständige Säulen dar (vgl. Abbildung 5.1a). Diese Darstellungsform wird jedoch häufig kritisiert (vgl. z. B. [105],

S. 100), da sie die Wechselwirkungen ökologischer, sozialer und ökonomischer Aspekte vernachlässigen. Daher werden mittlerweile eher Darstellungsformen bevorzugt, bei denen sich der wechselseitige Einfluss deutlicher ableiten lässt, wie bspw. das in Abbildung 5.1b dargestellte Modell mit drei gleich großen und sich überschneidenden Kreisen (vgl. z. B. [103], S. 100).

(a) : 3-Säulen-Modell. (b) : Venn-Diagramm. (c) : Prioritätenmodell.

Abbildung 5.1: Verschiedene Modelle zur Darstellung von Nachhaltigkeit.

Neben dieser Darstellungsart existieren auch sogenannte Prioritätenmodelle (vgl. Abbildung 5.1c), bei denen den drei Bereichen zusätzlich eine Rangordnung zugeteilt wird (vgl. z. B. [102], S. 682). Dabei nimmt bspw. die Ökologie die bedeutendste Stellung ein, mit dem Argument, ein „intakter“ Lebensraum sei Voraussetzung dafür, dass der Mensch überhaupt existieren kann, und so erst zu wirtschaftlichem Handeln fähig ist. Es gibt damit auch keine eindeutige Definition des Begriffs „Nachhaltigkeit“ (vgl. z. B. [106], S. 22 f.). Grunwald zufolge ist dies nicht möglich (vgl. z. B. [106], S. 86 f.), da Nachhaltigkeit ihm zufolge neben den notwendigen Entscheidungen auch Offenheit für Neues erfordert. Im Brundtland-Bericht wurde 1987 zur nachhaltigen Entwicklung folgendes festgehalten: „Die Menschheit hat die Möglichkeit, die Entwicklung nachhaltig zu gestalten, um sicherzustellen, dass sie die Bedürfnisse der Gegenwart erfüllt, ohne die Fähigkeit künftiger Generationen zu gefährden, ihre eigenen Bedürfnisse zu befriedigen.“ (vgl. [101], S. 24, Anmerkung: vom Autor aus dem Englischen übersetzt).

Aufgrund des unklaren Stellenwerts der einzelnen Dimensionen und insbesondere den Wechselwirkungen und Interessenkonflikten zwischen diesen scheint es umso wichtiger, bei der Bewertung von Nachhaltigkeit wissenschaftliche Methoden bereitzustellen, mit denen transparent die möglichen sozialen, wirtschaftlichen oder ökologischen Folgen dargestellt werden können. Zur Bewertung des *ökologischen* Teils der Nachhaltigkeit sollen mit der Ökobilanzierung die *potentiellen* Umweltwirkungen technologischer Systeme unter Berücksichtigung existierender Unsicherheiten ermittelt werden, wobei insbesondere auch die enthaltenen Unsicherheiten zu berücksichtigen sind. Die *Ergebnisse* der Ökobilanzierung können dann *zusammen* mit den Resultaten sozialer und wirtschaftlicher Resultate abgewogen werden. So entsteht eine *ganzheitliche* Grundlage, auf der dann geeignete Maßnahmen bestimmt und Entscheidungen getroffen werden, die tatsächlich und nicht nur vordergründig zu einer *nachhaltigen* Entwicklung beitragen können.

5.1.2 (Sozio)ökologische Systeme und Bewertungsgrenzen

Ein „ökologisches“ System muss kein „geschütztes“ System darstellen – und anlog dazu führt ein „künstliches“, technologisches System nicht unweigerlich zu „den Menschen störenden“ Umweltbelastungen, sondern ist in erster Linie ein vom Menschen entwickeltes System zur Erfüllung eines bestimmten Zwecks – mit einer Reihe erwünschter Effekte, aber eben auch potentieller, unerwünschter „Nebenwirkungen“. Welches weitgehend „natürli-

che“ als auch welches „künstliche“ und „trotzdem geschützte“ System gegen welche unerwünschten Veränderungen inwiefern geschont oder gänzlich verschont werden soll, basiert in der Regel auf politischen Entscheidungen. Aktuelle Leitstrategien zur Verhinderung von Umweltkrisen sind vorwiegend die Schonung von Ressourcen, die Effizienzsteigerung bei der Nutzung dieser sowie die Kreislaufwirtschaft (vgl. z. B. auch [83], S. 348 ff. und S. 338 ff.). Die Kreislaufwirtschaft verfolgt das Ziel, das Verhalten natürlicher Systeme auf technologische Systeme zu übertragen. So wird durch die politisch angestrebte Kreislaufwirtschaft versucht, dass die Stoffkreisläufe technologischer Systeme weitgehend geschlossen werden (vgl. z. B. [83], S. 338 ff.).

Bei sozioökologischen Systemen, bei denen nicht nur der rein ökologische Zustand, sondern auch der soziale Nutzen von Bedeutung ist, spielt die Resilienz eine Rolle (vgl. z. B. [83], S. 64 ff. und S. 205 f.). Resiliente Systeme können sich durch äußere Störungen verändern und wieder zum ursprünglichen Zustand zurückkehren oder einen anderen, stabilen Gleichgewichtszustand annehmen (vgl. z. B. [83], S. 105 f). Wird ein Ökosystem allerdings fortwährend gestört, kann es aus dem Gleichgewicht geraten, was zu irreversiblen Veränderungen oder gar zu dessen Zerstörung führen kann.

Klimawandel

Ein Beispiel für eine Störung des Ökosystems Erde ist der durch anthropogene Einflüsse verstärkte und beschleunigte Klimawandel als unerwünschte Wirkung der Technosphäre auf die Ökosphäre. In der Atmosphäre befinden sich von Natur aus Treibhausgase; sie bestehen zu ca. zwei Dritteln aus Wasserdampf und zu ca. einem Drittel aus Kohlenstoffdioxid. Die Treibhausgase ermöglichen die Bewohnbarkeit der Erde, da sie durch die Absorption und einer teilweisen Rückstrahlung der Wärme zur Erde die Auskühlung dieser vermeiden. Die *zusätzlichen*, durch den Menschen eingetragenen Treibhausgase führen hingegen zu einer erhöhten und beschleunigten Erwärmung der Erde. Diese anthropogen beschleunigte und verstärkte Erhöhung der Temperatur in der Atmosphäre stellt eine Sekundärwirkung bzw. „Midpoint“ (Information zu „Midpoint“ vgl. auch Kapitel 5.2.5) der erhöhten Strahlungsabsorption von zusätzlichen, in der Atmosphäre nicht natürlich vorhandenen Treibhausgasen dar.

Um eine irreversible Schädigung der Erde zu verhindern, wird eine Regulierung der Treibhausgasemissionen technologischer Systeme angestrebt, sodass die Bilanz der in die Systeme eingehenden sowie der ausscheidenden Ströme weitgehend neutral ist. Dies spiegelt sich auch in den Maßnahmen zum Erreichen nationaler Klimaschutzziele wider. Zum Beispiel zeigt sich dies in dem Bestreben, einen nahezu „klimaneutralen Gebäudebestand bis 2050“ erreichen zu wollen (vgl. z. B. [107], S. 52). Wo eine Minimierung bzw. der völlige Verzicht von klimarelevanten Emissionen nicht möglich ist, gibt es im Zuge der Modellierung eine faktische Vorgehensweise, um unerwünschte Elementarflüsse zu „neutralisieren“. Dabei werden rein theoretisch zwei in der Realität nicht zueinander gehörende Systeme zu einem System zusammengefasst, sodass „negative“ und „positive“ ein- und ausgehende Elementarflüsse der zusammengefassten Produktsysteme sich gegenseitig aufgeben bzw. „neutralisieren“.

Öko-Label und CO_2-Zerifikate

Es gibt Institutionen, die sich den Ansatz von „neutralisierten Flüssen“ mithilfe von Öko-Labeln zunutze machen. So können Unternehmen - aber auch Privatpersonen - Klimaschutz-Projekte unterstützen, z. B. zur Aufforstung in Deutschland oder auch in Nicaragua. Privatpersonen können auf Webseiten ihren jährlichen „Carbon Footprint“ (auf freiwilliger Basis) mit einem sog. CO_2-Rechner bestimmen (vgl. Internetauftritt [108]). Der „Carbon Footprint“ beruht auf Angaben wie dem jährlichen Verbrauch von Nahrungsmitteln, dem Heizenergiebedarf und der Nutzungsintensität von Verkehrsmitteln. Unternehmen können analog die Emissionen berechnen lassen, welche bspw. durch die Herstellungsprozesse eines Produktes entstehen. Im Zuge der Ergebnisdarstellung wird zu-

gleich angeboten, für Klimaschutzprojekte zu spenden, welche den berechneten „Carbon Footprint“ verringern oder gänzlich neutralisieren. Die Privatperson lebt dann z. B. ein Jahr lang „klimaneutral“ (vgl. z. B. [109]). Unternehmen, die für ihre Produktionsprozesse solche Öko-Label nutzen, können gleichsam ihre Produkte dann als „klimaneutral“ ausweisen; dies ist bspw. zunehmend auf Verpackungen von Pflegeprodukten zu finden.

Dieser Emissionshandel wird mit sog. CO_2-Zertifikaten durchgeführt. Die Anzahl der Zertifikate wird so berechnet, dass der aufgenommene Kohlenstoffdioxid derjenigen *äquivalenten Menge* der sog. „CO_2-Äquivalente“ (vgl. z. B. [110] sowie [111]) entspricht, die zuvor im CO_2-Rechner für die Lebensweise der Privatperson oder den Herstellungsprozess des Unternehmens bestimmt wurden. Auch wenn eine finanzielle Unterstützung von solchen Projekten zweifelsohne zu begrüßen ist, bleibt eine solche Vermarktung kritisch zu hinterfragen (vgl. auch [112], S. 13). Bei Waldprojekten muss bspw. gewährleistet sein, dass die Aufforstung zusätzlich und eigens durch das Projekt erfolgt. Hinzu kommt, dass z. B. eine verfrühte Freisetzung der Kohlenstoffdioxidemissionen durch illegale Waldrodung oder natürliche Waldbrände der aufgeforsteten Gebiete vermieden werden muss (vgl. auch [112], S. 6). Auch wenn die Bedingungen zu den Klimaschutzprojekten erfüllt werden können und wenn darüber hinaus angenommen wird, dass die klimarelevanten Emissionen sowohl im technologischen System exakt und in Gänze erfasst werden, bleiben weitere Aspekte zu berücksichtigen. So wird durch Aufforstungsprojekte in Nicaragua zwar unterstützt, dass dort zusätzlich gepflanzte Bambusplantagen Kohlenstoffdioxid zur Photosynthese aufnehmen. Im technologischen System eines hergestellten Produktes (oder einer Privatperson) sind jedoch weitere klimarelevante Emissionen zu erwarten, die in *anderen* Kreisläufen mit der Umwelt interagieren.

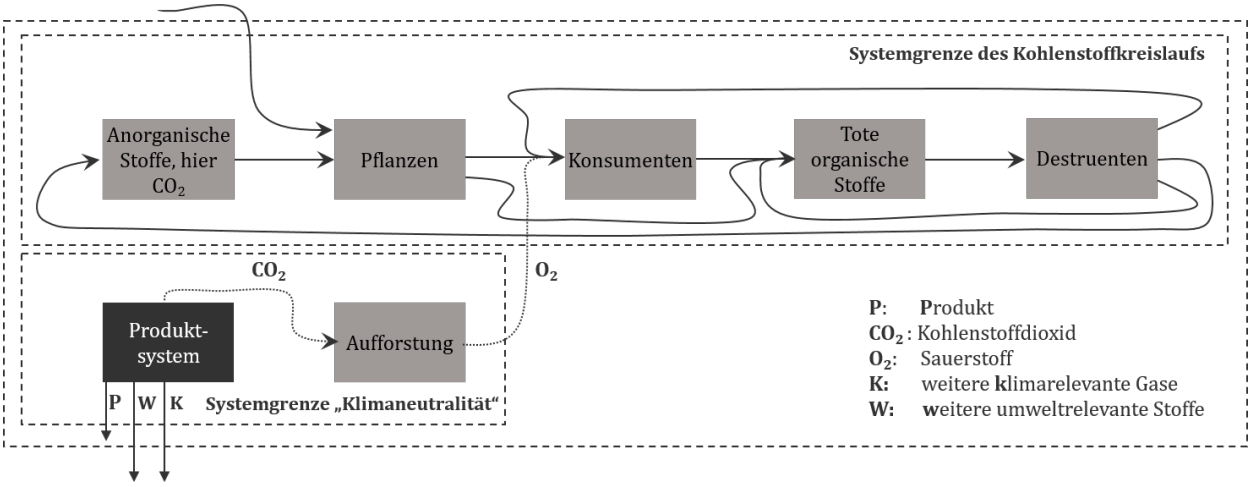


Abbildung 5.2: Produktsystem, das durch Einbezug eines Aufforstungs-Projekts als „klimaneutral“ gilt.

Hierfür ist der Kohlenstoffkreislauf (vgl. Kapitel 4.1.2, Abbildung 4.2) sowie ein fiktives Produktsystem, das durch Aufforstung als klimaneutral bezeichnet werden kann, vereinfacht in Abbildung 5.2 dargestellt. Mit dieser soll verdeutlicht werden, dass durch die Ergebnisse einer Ökobilanz keine Aussagen über die ökologischen Auswirkungen eines Produktsystems im Allgemeinen getroffen werden können – es können lediglich Aussagen über *einzelne* Umweltwirkungen, die das Produkt mit sich bringen kann, getätigt werden - und zwar über genau *diejenigen* Umweltwirkungen, welche durch die entsprechenden Wirkungsindikatoren in der Ökobilanz berücksichtigt werden. So kann das abgebildete Produktsystem weitere klimarelevante Gase *W* sowie andersartig umweltrelevante Stoffe *K* beinhalten, die durch den Kohlenstoffkreislauf nicht gebunden oder „neutralisiert“ werden können.

5.2 Relevante Normen und Begriffe

Die folgenden Begriffe sind für das Verständnis dieser Arbeit als relevant zu betrachten. Sie basieren auf den in Kapitel 5.2.1 genannten Regelwerken, und werden mit weiteren Definitionen ergänzt, um die in dieser Arbeit vorgenommenen Erläuterungen zu vereinfachen.

5.2.1 Normative Regelwerke

Wesentliche Grundsätze und Anforderungen zum Verfahren werden durch die international erstellten und in Deutschland geltenden Normen DIN EN ISO 14040 Umweltmanagement - Ökobilanz - Grundsätze und Rahmenbedingungen (ISO 14040:2006 + Amd 1:2020) (vgl. [92]) und DIN EN ISO 14044 Umweltmanagement - Ökobilanz - Anforderungen und Anleitungen (ISO 14044:2006 + Amd 1:2017 + Amd 2:2020) (vgl. [113]) vorgegeben.

Spezifische Rahmenbedingungen zur ökologischen Bewertung von Bauwerken werden in der DIN EN 15978 Nachhaltigkeit von Bauwerken - Bewertung der umweltbezogenen Qualität von Gebäuden - Berechnungsmethode (vgl. [114]) festgelegt. Die Grundregeln für Umweltkennzeichnungen und Umweltdeklarationen werden in weiteren Normen definiert. So sind die Grundsätze zur Bereitstellung quantifizierter, umweltbezogener Daten in Form von Umweltdeklarationen beispielsweise in der Norm DIN EN ISO 14025 Umweltkennzeichnungen und –deklarationen – Typ III Umweltdeklarationen – Grundsätze und Verfahren (vgl. [115]) geregelt.

Die Produktkategorieregeln für Bauprodukte werden spezifiziert in der Norm DIN EN 15804 Nachhaltigkeit von Bauwerken – Umweltproduktdeklarationen – Grundregeln für die Produktkategorie Bauprodukte (vgl. [116]) mit allgemeinen Grundregeln für Bauprodukte, die für einzelne Produktarten durch weitere Normen spezifiziert werden. Daneben ist auch die Auswahl und Verwendung generischer Daten geregelt in der CEN/TR 15941 Nachhaltigkeit von Bauwerken – Umweltproduktdeklarationen – Methoden für Auswahl und Verwendung von generischen Daten; Deutsche Fassung (CEN/TR 15941:2010) (vgl. [117]).

5.2.2 Phasen einer Ökobilanz

Eine Ökobilanz besteht allgemein aus vier Phasen, die iterativ durchlaufen werden. Die Phasen bestehen aus der Definition des Ziels und des Untersuchungsrahmens einer Ökobilanz, der Sachbilanz, der Wirkungsabschätzung und zuletzt der Auswertung (vgl. auch z. B. [113], S. 8). Diese Phasen werden in den folgenden Abschnitten kurz erläutert.

5.2.3 Ziel und Untersuchungsrahmen

Durch das Ziel und den Untersuchungsrahmen einer Ökobilanz sollen die Gründe für die Studie definiert und das zu analysierende Produktsystem, die verwendete funktionelle Einheit sowie die Vorgehensweise und die verwendeten Methoden beschrieben werden (vgl. z. B.[113], S. 17 f.).

5.2.4 Sachbilanz

Die Sachbilanz ist diejenige Phase, in der die für das Produktsystem notwendigen Informationen erhoben werden ([92], S. 22 f. sowie [113], S. 22 ff.). Die Erfassung der Inputs und Outputs erfolgt, sofern möglich, für jeden einzelnen als Prozessmodul abgebildeten Prozess des Technologiesystems. Anhand der erfassten Elementar- und Produktflüsse wird dann mit Bezug auf die funktionelle Einheit die Skalierung der einzelnen Prozessmodule berechnet. Darauf aufbauend werden alle Inputs und Outputs des ganzen Produktsystems kalkuliert.

5.2.5 Wirkungsabschätzung

Die Wirkungsabschätzung dient zur Abschätzung von potentiellen Umweltwirkungen, die relativ in Bezug zur funktionellen Einheit (Definition „funktionelle Einheit“ vgl. auch Kapitel 5.2.12) ermittelt werden (vgl. z. B. [92], S. 23 ff. sowie [113], S. 28 ff.). Hierfür müssen Wirkungskategorien sowie geeignete Wirkungsindikatoren als auch Charakterisierungsmodelle ausgewählt werden, wobei diese mit dem Untersuchungsrahmen und dem Ziel abgestimmt sein müssen. Diejenigen Umweltwirkungen, welche im Rahmen einer Ökobilanzierung adressiert werden sollen, werden im Bilanzierungsrahmen einer Ökobilanz in Form von Wirkungskategorien wie bspw. der Wirkungskategorie „Klimawandel“ festgelegt. Die Umweltwirkungen werden mittels Wirkungsindikatoren quantifiziert in Bezug zu einer Referenzemission.

Wirkungskategorien

Als Wirkungskategorien werden Kategorien bezeichnet, die gemäß [113], S. 15 „wichtige Umweltthemen“ repräsentieren. Ein Beispiel hierfür ist der Klimawandel (vgl. Kapitel 5.1.2, Abschnitt „Klimawandel“).

Wirkungs(kategorie)indikatoren

Mit Wirkungskategorieindikatoren (oder kurz häufig auch nur als „Wirkungsindikatoren“ bezeichnet) können Wirkungskategorien quantifiziert werden (vgl. z. B. [113], S. 15). Indikatoren mit Bezug auf Primär- oder ggfs. auch Sekundärwirkungen werden als „Midpoints“ bezeichnet (vgl. [93], S. 223 f.), während daraus resultierende Folgen wie z. B. Schäden am menschlichen Organismus, irreversible Störungen der Umwelt oder ein Verlust der Artenvielfalt „Endpoints“ darstellen; der Bezug auf solche sog. „Endpoints“ geht mit einem großen Anstieg an Unsicherheit einher [93], S. 224 f. sowie [98], S. 109). Daher werden stattdessen häufig Midpoint-Indikatoren verwendet. Die Temperaturerhöhung der Wirkungskategorie „Klimawandel“ kann beispielsweise als Sekundärwirkung eingestuft werden, während der Anstieg des Meeresspiegels hingegen eine Tertiärwirkung darstellt (vgl. [93], S. 224 f.). Die Referenzemission des Treibhauspotentials ist CO_2.

Charakterisierungsfaktoren

Mit den Charakterisierungsfaktoren können die zu einem konkreten Wirkungsindikator zugeordneten, umweltrelevanten Elementarflüsse auf eine gemeinsame Einheit umgerechnet werden (vgl. z. B. [92], S. 14). Sie beziehen sich auf einen festgelegten Referenzfluss und sind in Charakterisierungsmodellen zusammengefasst.

Charakterisierungsmodelle

Die Charakterisierungsmodelle enthalten die für ein konkretes Umweltthema relevanten Elementarflüsse, einen festgelegten Referenzfluss sowie die zugehörigen Charakterisierungsfaktoren. Für die Abbildung der Wirkungskategorie Klimawandel ist das Charakterisierungsmodell zum „Global Warming Potential“ des IPCC geeignet (vgl. auch [118], S. 663), wobei der Indikator z. B. einen Betrachtungszeitraum von 100 Jahren umfasst.

5.2.6 Auswertung

Mithilfe der Ergebnisse soll es möglich sein, Schlussfolgerungen zu definieren und Empfehlungen abzuleiten. In der Auswertung sollen hierfür die Ergebnisse mit verschiedenen Methoden überprüft und verständlich dargestellt werden (vgl. z. B. ([92], S. 26 f. sowie [113], S. 36 ff.).

5.2.7 Treibhauspotential

Mit dem Wirkungskategorieindikator „Treibhauspotential“ (englisch: „Global Warming Potential“, kurz *GWP*) können die potentiellen Umweltwirkungen von Systemen in Bezug auf den Klimawandel ermittelt werden (vgl. auch Abschnitt „Klimawandel“ in Kapitel 5.1.2). Der Indikator bezieht sich dabei auf die erhöhte Strahlungsabsorption, die potenziell zu einer Erderwärmung führt. Die Referenzemission des Treibhauspotentials ist Kohlenstoffdioxid (CO_2); der Einfluss der übrigen, klimarelevanten Gase wird im relativen Verhältnis zu Kohlenstoffdioxid bestimmt. Um lediglich den zusätzlichen Erwärmungseffekt durch die anthropogen verursachten Treibhausgase zu erfassen, erfolgt die Ermittlung in einem Strahlungsbereich von ca. 8 µm bis ca. 15 µm, in welchem das Absorptionsvermögen der natürlichen Treibhausgase CO_2 und Wasserdampf möglichst gering ist. Die Ermittlung der spezifischen Einflüsse durch die einzelnen, klimarelevanten Gase erfolgt mithilfe von komplexen, dynamischen Klimamodellen, aus denen sich - mit gewissen Unsicherheiten behaftete - Charakterisierungsfaktoren ableiten lassen. Sie werden vom *Intergovernmental Panel on Climate Change* (kurz *IPCC*, häufig auch als Weltklimarat bezeichnet) bereitgestellt. Der *GWP* wird durch Aufaddieren der einzelnen Stoffmengen m_i, die mit ihren spezifischen Charakterisierungsfaktoren multipliziert werden, bestimmt (vgl. [93], S. 319):

$$\mathrm{GWP} = \sum_i (m_i \cdot \mathrm{GWP}_i)$$

5.2.8 Produkt

Ein Produkt wird definiert als „jede Ware oder Dienstleistung“ ([92], S. 10 f.).

Zwischenprodukt

Ein Produkt, das innerhalb eines Produktsystems erzeugt wird und auch innerhalb desselben Produktsystems wieder verwendet wird, ist ein Zwischenprodukt (vgl. [92], S. 12).

Hauptprodukt

Das Produkt, das durch ein Prozessmodul oder Produktsystem erzeugt oder verbraucht werden soll, wird in dieser Arbeit als Hauptprodukt bezeichnet.

Endprodukt

Ein Produkt, das durch ein Produktsystem erzeugt oder verbraucht werden soll, wird in dieser Arbeit als Endprodukt bezeichnet (vgl. auch [92], S. 18).

5.2.9 Zusatzprodukt

Eine Substanz, die neben dem Hauptprodukt eines Prozessmoduls oder Produktsystems erzeugt wird und in weitere Prozessmodule oder Produktsysteme innerhalb oder außerhalb des Produktsystems fließt, wird in dieser Arbeit allgemein als Zusatzprodukt bezeichnet. Dies umfasst Koppelprodukte sowie Nebenprodukte. Produkte, bei denen noch nicht bekannt ist, ob sie das Ende der Abfalleigenschaften (Definition „Ende der Abfalleigenschaften“ vgl. Kapitel 5.2.10) erreichen, werden in dieser Arbeit ebenfalls allgemein als Zusatzprodukte bezeichnet.

Internes Zusatzprodukt

Ein Zusatzprodukt, das innerhalb des Produktsystems wieder verbraucht wird, wird in dieser Arbeit als internes Zusatzprodukt bezeichnet. Dabei kann es sich um ein intern verwendetes Koppel- oder Nebenprodukt handeln, welches zugleich ein Zwischenprodukt ist.

Externes Zusatzprodukt

Ein Produkt, das die Systemgrenze überschreitet oder durch eine Systemerweiterung im Produktsystem substituiert wird, wird in dieser Arbeit allgemein als externes Zusatzprodukt bezeichnet. Dabei kann es sich um ein extern verwendetes Koppel- oder Nebenprodukt handeln.

Koppelprodukt

Ein Zusatzprodukt, das einen Nutzen hat und ohne wesentliche, weitere Verarbeitung direkt verwendet werden kann, wird gemäß [92], S. 11 als Koppelprodukt (englisch: „Co-Products“) bezeichnet. Im chemischen Bereich werden nach vgl. [119], S. 12 Produkte, die neben dem „Wertprodukt“ entstehen, als Koppelprodukt bezeichnet. Dabei sind „Wertprodukte“ in dieser Arbeit zu übersetzen mit Hauptprodukten. Gemäß [96], S. 60 kann die Zuordnung mithilfe des Verkaufswerts in Relation zum Hauptprodukt erfolgen: Weist ein Produkt einen vergleichsweise hohen Marktwert auf, kann es demnach den Koppelprodukten zugeordnet werden.

Nebenprodukt

Ein Zusatzprodukt, das auf Basis des Verkaufswerts in Relation zum Hauptprodukt einen vergleichsweise niedrigen Marktwert aufweist, wird angelehnt an [96], S. 60 in dieser Arbeit als Nebenprodukt bezeichnet. Dies deckt sich weitgehend mit den Definitionen im chemischen Bereich, in denen „unerwünschte“ oder „wertlose“, nebenbei anfallende Substanzen als Nebenprodukte bezeichnet werden. Diese sollen soweit wie möglich vermieden oder umgewandelt werden ([119], S. 13). In der Abfallrahmenlichtlinie [120], S. 7 werden hingegen Substanzen, welche vom weitgehend direkten Nutzen sind, als „Nebenprodukte“ bezeichnet.

5.2.10 Abfall

In der Norm (vgl. [92], S. 13) wie auch in der „Abfallrahmenrichtlinie“ (vgl. [120], S. 3) wird Abfall als „Substanz“ bezeichnet, die entsorgt werden soll oder muss und deren Abfalleigenschaft nicht beendet werden kann. Gemäß [96], S. 60 kann eine Substanz mit negativem Verkaufswert, d. h. wenn dessen Abgabe kostenpflichtig ist, der Kategorie Abfall zugeordnet werden.

Ende der Abfalleigenschaft

In der Abfallrahmenrichtlinie (vgl. [120], Artikel 6, S. 7 f.) wird als „Ende der Abfalleigenschaft“ festgelegt, dass Abfälle, die rezykliert oder verwertet wurden, verwendbar sind und als unschädlich für die Umwelt und den Menschen gelten, nicht mehr als Abfall bezeichnet werden.

Abfallhierarchie

Gemäß der Abfallhierarchie (vgl. [120], S. 6) soll Abfall vorrangig vermieden werden. Ist dies nicht möglich, soll Abfall, wenn möglich, wiederverwendet werden. Alternativ sollen Rezyklierprozesse durchgeführt werden. Ist auch

dies nicht möglich, soll Abfall anderweitig verwertet werden. Die endgültige Entsorgung soll die letztmögliche Option sein.

5.2.11 Abfallbehandlung und -entsorgung

Beim Umgang mit Produkten oder Abfall gibt es verschiedene Verfahren, die nachfolgend kurz erläutert werden.

Wiederverwendung

Die Nutzung eines Produktes, welches nach einer vorangegangenen Nutzung und einer entsprechenden Aufbereitung wieder für dieselbe Funktion verwendet werden kann, wird [120], S. 5 zufolge als Wiederverwendung bezeichnet.

Behandlung

Die Verwertung oder Beseitigung von Abfällen wird nach [120], S. 6 allgemein als „Behandlung“ von Abfall bezeichnet.

Verwertung

Durch Verwertungsverfahren wird Abfall gemäß [120], S. 5 derart aufbereitet, dass er das Ende der Abfalleigenschaft erreicht und so nach der Behandlung wieder eine Funktion erhält.

Beseitigung

Alle weiteren Verfahren, durch welche Abfall nicht verwertet werden, werden nach [120], S. 6 als Beseitigung bezeichnet.

Rezyklierung

Die Rezyklierung ist gemäß [120], S. 5 ein Verwertungsverfahren, wobei die energetische Verwertung sowie die Aufbereitung zu Sekundärbrennstoffen ausgenommen ist.

5.2.12 Produktsystem

Gemäß Norm [113], S. 14 und [92], S. 18 ff. besteht ein Produktsystem aus Prozessmodulen und ein- sowie ausgehenden Flüssen, um ein Produkt unter Berücksichtigung der funktionellen Einheit darzustellen. Ein Produktsystem stellt „den Lebensweg(abschnitt) eines Produktes“ dar. Dabei sollen diejenigen Prozessmodule berücksichtigt werden, welche zur Erzeugung bzw. Verarbeitung des Endproduktes oder der dafür notwendigen Zwischenprodukte beitragen und innerhalb des Betrachtungsrahmens liegen.

Herstellendes Produktsystem

Ein Produktsystem, dessen Nutzen es ist, ein Produkt zu erzeugen, wird in dieser Arbeit auch als „herstellendes Produktsystem“ bezeichnet.

Verarbeitendes Produktsystem

Ein Produktsystem, dessen Nutzen es ist, ein Produkt zu verarbeiten (z. B. durch eine Abfallbehandlung), wird in dieser Arbeit auch als „verarbeitendes Produktsystem“ bezeichnet.

5.2.13 Prozessmodul

Das Prozessmodul ist nach [92], S. 13 definiert als die kleinste Einheit, für welche ein- und ausgehende Flüsse modelliert und quantifiziert werden können. Die Prozessmodule bilden die einzelnen Prozesse eines Technologiesystems ab.

Aggregiertes Prozessmodul

Als aggregiertes Prozessmodul wird angelehnt an [121], S. 110 in dieser Arbeit ein Prozessmodul bezeichnet, das ein Subsystem darstellt, welches als Black-Box verfügbar ist. Seine Flüsse sind kumuliert, sodass nur die eingehenden und ausgehenden Flüsse aus dem System bekannt sind.

Prozess

Ein Prozess besteht nach [92], S. 11 aus Vorgängen, durch welche Inputs in Outputs überführt werden.

Inputs und Outputs

Bei den Inputs und Outputs handelt es sich gemäß [113], S. 13 und S. 14 um quantifizierte und als Elementar- und Produktflüsse modellierte, ein- und ausgehende Ströme eines Prozesses.

5.2.14 Systemstruktur

Die Produktsysteme können unterschiedliche Systemstrukturen aufweisen. Die Systemstruktur ist gekennzeichnet durch die Anordnung der Prozessmodule und ihre Verknüpfung untereinander. Die Grundstruktur ist diejenige Struktur, die sich ergibt, wenn das Produktsystem keine Hyper- oder Multikanten enthält. Es handelt sich hierbei in der Regel um einfache Systemstrukturen von linearen oder baumartigen Produktsystemen. Mischstrukturen sind Strukturen, die sich ergeben, wenn mehrere Arten von Systemstrukturen ineinandergreifen und sich nicht in mehreren, voneinander getrennten Subsystemen abbilden lassen.

Subsystem

Ein Subsystem ist angelehnt an [77], S. 283 ein abgrenzbares Teilsystem eines übergeordneten Systems. In dieser Arbeit kann ein Produktsubsystem aus einem oder mehreren Prozessmodulen bestehen. Jedes Produktsystem kann ein Subsystem eines übergeordneten Systems darstellen (vgl. auch [122], S. 65).

Multifunktionales System

Als multifunktionales System wird ein System bezeichnet, das mehr als einen Zweck erfüllt. Dies kann der Fall sein, wenn durch ein System mehrere Produkte erzeugt werden.

5.2.15 Funktionelle Einheit

Die funktionelle Einheit stellt nach [92], S. 21 die messbare(n) Funktion(en) bzw. den quantifizierten Nutzen eines Produktsystems dar. Sie ist der Bezug der Inputs und Outputs des Produktsystems und ermöglicht den Vergleich mehrerer Produktsysteme mit gleicher Funktion.

5.2.16 Fluss

In der Methodik der Ökobilanzierung beschreibt ein Fluss allgemein einen als Produkt- oder Elementarfluss modellierten Produkt-, Stoff- oder Energiestrom.

Produktfluss

Ein Produktfluss beschreibt gemäß [92], S. 12 ein Produkt, welches als Input dem Produktsystem zugeführt wird oder als Output das Produktsystem verlässt. In dieser Arbeit werden Produktflüsse allgemein definiert als Produkte, die als Inputs oder Outputs einem Prozessmodul oder Produktsystem zugeführt werden oder ein Prozessmodul oder Produktsystem wieder verlassen. Der Produktfluss gilt als allgemeiner Begriff und kann einen Zwischen-, End-, Haupt-, Zusatz-, Neben-, oder Koppelproduktfluss repräsentieren.

Demnach kann es sich auch bei Abfall um ein Produkt handeln, dass an ein Produktsystem weitergegeben wird, welches den Abfall aufbereitet oder beseitigt. Daher wird Abfall in dieser Arbeit als Produktfluss erfasst und modelliert. Dies wird in bisherigen Ökobilanzprogrammen nicht einheitlich gehandhabt (weiterführende Informationen zu etablierten Ökobilanzprogrammen wie *openLCA* und *GaBi* vgl. auch Kapitel 5.5). Während bspw. im Ökobilanzprogramm *openLCA* die drei Flusstypen Produkt-, Elementar- und Abfallfluss existieren (vgl. [123], S. 36), wird im Ökobilanzprogramm *GaBi* grundsätzlich zwischen „valuable flows“ (deutsche Übersetzung: „wertvoller Fluss“) und Elementarflüssen unterschieden (vgl. [121], S. 50), wobei die „valuable flows“ differenziert werden in „valuable material flows“ (deutsche Übersetzung: „Wertstoffffluss“) oder „waste material flows“ (deutsche Übersetzung: „Abfallstofffluss“) (vgl. [121], S. 57 f.).

Zwischenproduktfluss

Ein Zwischenproduktfluss ist ein Produktfluss, der angelehnt an [92], S. 12 den Fluss eines Zwischenproduktes repräsentiert.

Elementarfluss

Ein Elementarfluss stellt einen Stoff- oder Energiestrom dar, der ohne zusätzlichen Prozessschritt aus der Umwelt einem Technologiesystem zugeführt wird oder ohne weiteren Prozessschritt von einem Technologiesystem „an die Umwelt abgegeben wird“ ([113], S. 12).

Referenzfluss

Ein Referenzfluss ist ein Produktfluss, auf den alle anderen Flüsse des Produktsystems bezogen werden, damit die Flüsse entsprechend der funktionellen Einheit quantifiziert werden (vgl. [92], S. 14).

5.2.17 Element

Als Elemente werden in dieser Arbeit allgemein Komponenten von Systemen bezeichnet; dies können Prozessmodule, Flüsse oder Subsysteme sein.

5.2.18 Gruppe

Eine Gruppe ist in dieser Arbeit eine Zusammenfassung mehrerer Elemente mit identischem Referenzfluss. Es können z. B. vergleichbare Prozessmodule, Subsysteme und Produktsysteme gruppiert werden zu Prozessmodulgruppen, Subsystemgruppen und Produktsystemgruppen. Es ist möglich, dass eine Gruppe nur aus einem Element besteht. Ein Produktsystem kann so bspw. nur aus einem Prozessmodul bestehen oder aus einer einzelnen Prozessmodulgruppe, bzw. aus einem Subsystem oder einer einzelnen Subsystemgruppe. Generische Datensätze repräsentieren häufig Gruppen, z. B. werden mehrere, ähnliche Produkte zu einer Produktgruppe zusammengefasst. In herkömmlichen Ökobilanzen werden für generische Datensätze häufig Durchschnittsdaten verwendet („Durchschnittsdaten“ vgl. Kapitel 5.2.20). Bei Anwendung des intervallarithmetischen Ansatzes ergeben sich die jeweiligen Untergrenzwerte der einzelnen Flüsse aus den geringsten Werten und die Obergrenzwerte analog aus den höchsten Werten (weiterführende Informationen sind in [122], S. 66 zu finden).

5.2.19 Lebensweg

Der Lebensweg stellt die „Stufen eines Produktsystems“ ([92], S. 9) dar vom Abbau der benötigten Rohstoffe bis zur vollständigen Beseitigung des im Zuge des Produktsystems dargestellten Produktes.

Lebenswegabschnitt

Der Lebenswegabschnitt ist eine einzelne Stufe des Lebenswegs, wie zum Beispiel die Rohstoffgewinnung, der Produktionsprozess, die Nutzungsphase, die Beseitigung sowie die Behandlung nach der Nutzung und/oder Wiederverwendung (vgl. z. B. [92], S. 49 und S. 51).

Informationsmodule

Gemäß [116], S. 20 ff. werden Lebenswegabschnitte zur Bewertung von Gebäuden in die Informationsmodule *A*, *B*, *C* und *D* untergliedert. Das Modul *A* bildet Informationen über die Herstellung und Errichtung bzw. den Einbau des betrachteten Produktes ab. Mit dem Modul *B* wird die Nutzungsphase modelliert, das Modul *C* dient zur Modellierung der Entsorgung bzw. Behandlung des betrachteten Produktes nach der Nutzung und das Modul *D* ist ergänzend zur Erfassung von Belastungen und Gutschriften außerhalb der Systemgrenze verfügbar, welches z. B. durch Substitution berücksichtigt werden kann.

5.2.20 Daten

Daten zur Durchführung einer Ökobilanzierung können gemäß [117], S. 5 in systemspezifische und generische Daten unterteilt werden.

Durchschnittsdaten

Durchschnittsdaten sind Daten, die in einer bestimmten Art und Weise gemittelt wurden und von verschiedenen Herstellern oder mehreren Produktionslinien stammen können (vgl. [117], S. 5).

Systemspezifische Daten

Systemspezifische Daten beruhen nach [117], S. 6 auf den Informationen eines konkret zu modellierenden Produktsystems. Systemspezifische Daten können auch Durchschnittsdaten sein (vgl. z. B. [117], S. 7).

Anlagenspezifische Daten

Anlagenspezifische Daten beschreiben [117], S. 6 zufolge die Informationen einer Anlage, welche verschiedene Produktionslinien aufweisen kann.

Generische Daten

Generische Daten sind nach [117], S. 5 „Ersatzdaten“, wenn keine spezifischen Informationen vorliegen; sie können anlagenspezifische Daten oder Durchschnittsdaten sein.

5.2.21 Unsicherheitsfluss

Der Begriff der „Fehlerfortpflanzung“ ist häufig im Zusammenhang mit der Messtechnik zu finden. Er beschreibt die Weiterreichung von Abweichungen im Zuge von Berechnungsschritten. Da mittlerweile in der Messtechnik zwischen „Fehlern“ und Abweichungen unterschieden wird (vgl. z. B. [124], S. 14), werden die Bezeichnungen „Fehlerrechnung“ und „Fehlerfortpflanzung“ als obsolet angesehen und daher in dieser Arbeit nicht verwendet. Stattdessen wird der Begriff „Unsicherheitsfluss“ verwendet. Gelangen unsichere Werte in die Bilanzierung, „fließen“ auch die enthaltenen „Unsicherheiten“ bis zu den Ergebnissen, wobei sie sich – in Wechselwirkung mit weiteren Unsicherheiten - vergrößern oder aber verringern können.

5.3 Grundsätzliche Annahmen und Voraussetzungen

Die herkömmliche Methodik der Ökobilanzierung bringt Bedingungen, Besonderheiten und Herausforderungen mit sich, die beachtet werden müssen. Diese werden in den folgenden Kapiteln erläutert.

5.3.1 Relevanz einer Trennlinie zwischen Öko- und Technosphäre

In der Ökobilanzierung wird zwischen der Ökosphäre und der Technosphäre unterschieden. Diese Abgrenzung ist notwendig, um überhaupt eine Bilanzierung vornehmen zu können. Die Herkunft des Begriffs „Bilanzierung“, ist im lateinischen Wort „bilanx“ zu finden, was übersetzt „zwei Waagschalen habend“ bedeutet (Übersetzung vgl. z. B. [125]). Bei einer Bilanzierung bedarf es zweier Seiten, die gegenübergestellt und verglichen werden. Ohne eine Unterscheidung zwischen zwei Seiten und der Festlegung dieser Systemgrenzen wäre eine Bilanzierung nicht durchführbar.

Auch die in dieser Arbeit weiterentwickelte Methodik zur Ökobilanzierung baut auf einer getrennten Betrachtung von Techno- und Ökosphäre auf. In der Praxis der Ökobilanzierung wird die Ökosphäre allgemein als der gesamte Bereich definiert wird, der nicht zur Technosphäre gehört (vgl. [93], S. 32). Eine klare „Entweder-Oder“-Trennung zwischen Ökosphäre und Technosphäre ist nicht uneingeschränkt möglich. Grund hierfür ist, dass Ökosysteme wie auch Technologiesysteme vorwiegend offene Systeme sind (Erläuterung zu „offenen Systemen“ vgl. Kapitel 4.1.1), die an den Grenzen über Stoff- und Energieströme miteinander in Verbindung stehen. So beeinflussen Veränderungen der Umwelt - wie z. B. die Erderwärmung - die Technosphäre. So werden z. B. in einigen Regionen Gebäude durch die Erderwärmung weniger geheizt und/oder vermehrt gekühlt. Dazu resultiert bspw.

aus den fehlenden Stoffkreisläufen von Technologiesystemen, dass die produzierten Abfälle und Edukte nicht vollständig verwertet werden können und sich in Deponien anreichern. So ist bspw. der Birkenkopf in Stuttgart ein über 40 Meter hoher Berg, der mit seiner Aussicht auf die Natur als beliebtes Ausflugsziel gilt. Nur noch vereinzelt erkennbar ist dabei sein „künstlicher“ Ursprung aus Trümmerschutt, weshalb der Berg im Volksmund auch als „Monte Scherbelino“ bezeichnet wird. In gewisser Weise entstehen dadurch Übergangsbereiche, die *weder* Ökosystem *noch* Technologiesystem sind, sondern vielmehr *sowohl* Ökosystem *als auch* Technologiesystem. Dies wird mitunter auch als „Hybride“ bezeichnet (Kontext zu „Hybride“ vgl. Kapitel 6.1.1).

5.3.2 Lineare Modellierung und Black-Box-Ansatz

Das in dieser Arbeit vorgeschlagene intervallarithmetische Verfahren baut auf der herkömmlichen Methodik der Ökobilanzierung auf. Da sich die einzelnen Phasen der Ökobilanzierung gegenseitig beeinflussen, müssen diese im Laufe einer Ökobilanz immer wieder neu betrachtet und gegebenenfalls angepasst werden. Die Ökobilanzierung ist somit eine *iterative* Methode, d. h. die durchgeführten Berechnungen, aber auch die getroffenen Annahmen, Methoden und ggfs. auch die gesetzten Systemgrenzen und Kriterien werden immer wieder geprüft und falls notwendig aktualisiert. Die Ökobilanzierung basiert dazu auf einem *relativen, vergleichenden* Ansatz. Dieser erfordert grundsätzlich Vergleiche, um Aussagen treffen zu können, da die bilanzierten Ergebnisse keine absoluten Umweltwirkungen darstellen.

Dies setzt gleichsam voraus, dass die gegenübergestellten Ökobilanzen vergleichbare Produktsysteme mit derselben funktionellen Einheit aufweisen und möglichst auf denselben Randbedingungen, Methoden und Systemgrenzen basieren. Die Methodik der Ökobilanzierung beruht auf einem statischen, linearen Ansatz zur Berechnung der Umweltwirkungen (vgl. auch Kapitel 4.3.3). Es gibt alternative Ansätze hierzu; ein solches wurde z. B. in [126] vorgestellt, bei dem in den Modellen auch chemische und physikalische Abhängigkeiten hinterlegt werden können. In [127] wird ein Verfahren vorgestellt, bei dem die Verknüpfung verschiedener Modelle erprobt wurde. So wurde ein Gebäudemodell zur thermischen Gebäudesimulation, ein Materialflussmodell zur energetischen Bilanzierung und ein Maschinenmodell zur Abbildung der Produktionsprozesse gekoppelt.

Diese Arbeit erweitert die bisherige Methodik der herkömmlichen Ökobilanzierung mit dem Ziel, Unsicherheiten umfassend berücksichtigen zu können. Alternative Ansätze zur konzeptuellen Modellbildung werden daher in dieser Arbeit nicht thematisiert.

5.3.3 Subsysteme von Produktsystemen

Subsysteme von Produktsystemen spielen bei der Abbildung vor- und nachgelagerter Produktionsprozesse eine wichtige Rolle. Bei einer Betrachtung des gesamten Lebenswegs und einer vollständigen Datenerhebung aller Vorgänge inklusive der vor- und nachgelagerten Prozesse (die sog. „Up-“ und „Downstreams“) beginnen und enden die modellierten Produktsysteme mit Prozessmodulen, die als Inputs bzw. Outputs keine Produktflüsse, sondern nur Elementarflüsse aufweisen. Da die vollständige Erfassung aller Informationen der vor- und nachgelagerten Prozesse oftmals nicht möglich ist, wird meistens auf generische Datensätze (Definition „generische Daten“ vgl. Kapitel 5.2.20) zurückgegriffen. Diese Daten beruhen auf anderen Produktsystemen – und sind damit Untergraphen bzw. Subsysteme im zu modellierenden Produktsystem.

In Abbildung 5.3 ist zur Erläuterung exemplarisch das Produktsystem PS_3 abgebildet, welches das Produkt P_2 benötigt. Da jedoch nicht genügend spezifische Informationen über die Herstellung des Produktes zur Verfügung stehen, wird zur Abbildung des Produktes P_2 ein generischer Datensatz verwendet. In diesem ist die Herstellung des Produktes P_2 durch die Modellierung des Produktsystems PS_2 abgebildet. Im Produktsystem PS_2 wird wieder-

um zur Abbildung des Produktes P_1 auf das Produktsystem PS_1 zurückgegriffen. In der Praxis ist es der Regelfall, dass zur Modellierung eines Produktsystems ab einer bestimmten Ebene auf bereits modellierte Subsysteme vergleichbarer Produkte zurückgegriffen wird.

Der Nutzer verfügt in der Regel über zusätzliche Informationen zum Bilanzierungsrahmen; d. h. es werden die verwendeten Methoden und Rahmenbedingungen wie die Systemgrenzen, die Bewertungsmethoden und die hinterlegten Charakterisierungsmodelle, die Allokationsregeln und die Abschneidekriterien aufgezeigt.

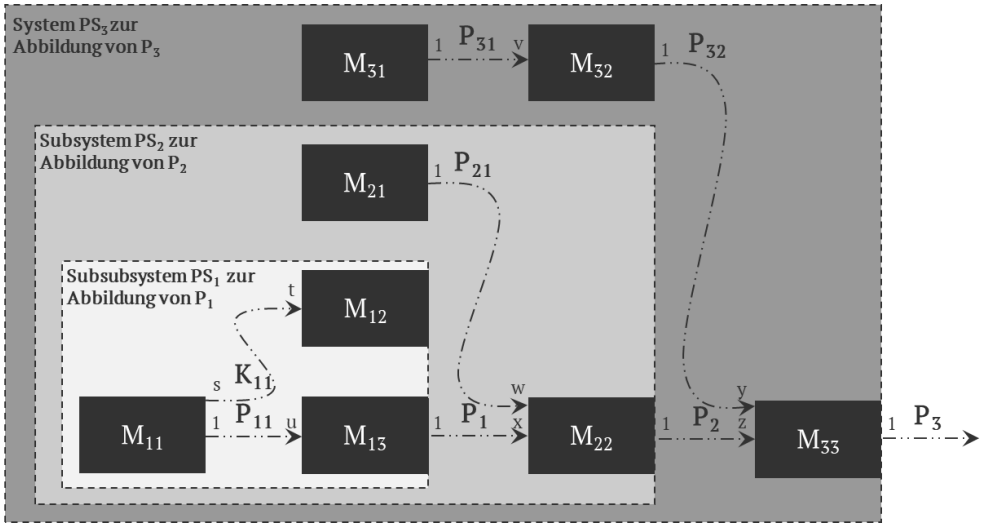


Abbildung 5.3: Produktsystem PS_3, das generische Daten verwendet, die als Subsysteme integriert werden.

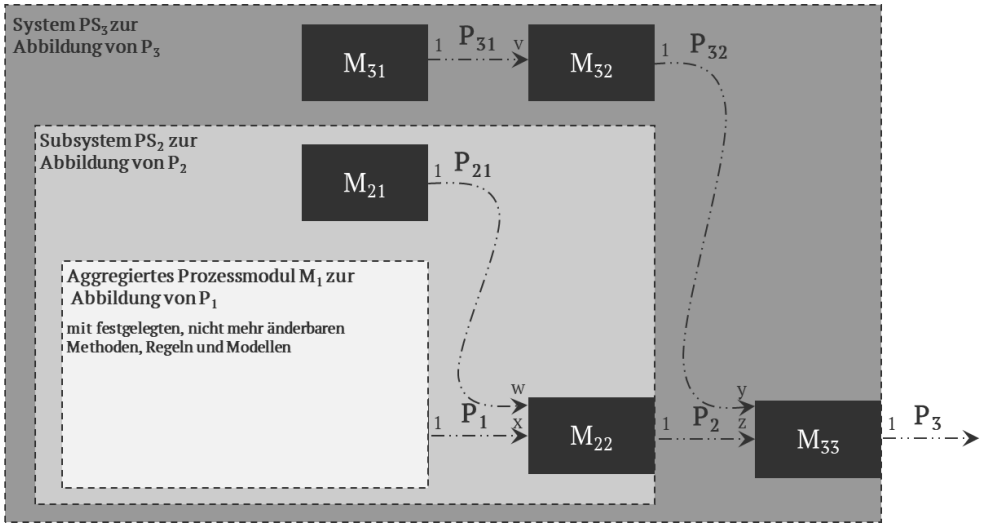


Abbildung 5.4: Generischer Datensatz mit Informationen des Produktes P_1 als aggregiertes Prozessmodul.

Dabei ist jedoch die Vielzahl verfügbarer Datensätze nur als „Black-Box“ erhältlich, was exemplarisch in Abbildung 5.4 dargestellt ist. Der Nutzer erhält zwar Informationen über die auf die funktionelle Einheit bezogenen Ergebnisse der Sachbilanz bzw. der Wirkungsabschätzung sowie den zugrunde gelegten Bilanzierungsrahmen, das modellierte Produktsystem sowie dessen Subsysteme sind jedoch nicht einsehbar.

5.3.4 Ökobilanzierung aus Ingenieurssicht

Die Tätigkeit eines Ökobilanzierenden gleicht einer Schnittstelle: er benötigt für eine Ökobilanz Informationen aus verschiedenen (Wissenschafts-)Bereichen. Jeder Bereich verfügt über einen eigenen, in sich geschlossenen Kompetenzbereich. Innerhalb eines Bereiches können Daten eingelesen, neue Informationen erzeugt sowie Werte bearbeitet und ausgegeben werden. Der Ökobilanzierende kann auf Daten unterschiedlicher Bereiche zugreifen, diese verarbeiten und daraus die zu erwartenden Umweltwirkungen ermitteln. Die Daten kann er jedoch in vielen Fällen nicht oder nur sehr begrenzt überprüfen. Dies gilt auch über Informationen zur Unsicherheit der übermittelten Daten. In Abbildung 5.5 sind die Datenquellen aus diversen Modellen dargestellt, die ein Ökobilanzierender benötigt und die häufig nur als Black-Boxen (vgl. Kapitel 4.1.6) verfügbar sind.

Sind die einzelnen Prozessschritte der Produktsysteme identifiziert und als Prozessmodule abbildbar, können derart modellierte Produktsysteme als Grey-Boxen bezeichnet werden (vgl. z. B. auch [75], S. 83). So sind die wesentlichen Komponenten und linearen Zusammenhänge des betrachteten Technologiesystems bekannt; sie können als Gleichungssysteme dargestellt und gelöst werden (vgl. z. B. auch [126], S. 18 f.). Die Charakterisierungsmodelle zur Abschätzung von Umweltwirkungen sowie die Sach- und Ökobilanzen zur Abbildung vor- und nachgelagerter Technologiesysteme fließen in der Regel als Black-Boxen in die Modellbildung ein (vgl. z. B. auch [126], S. 18), obgleich die Informationen der Charakterisierungsmodelle tatsächlich auf komplexen (White-Box)-Modellen basieren (z. B. auf dynamischen Klimamodellen).

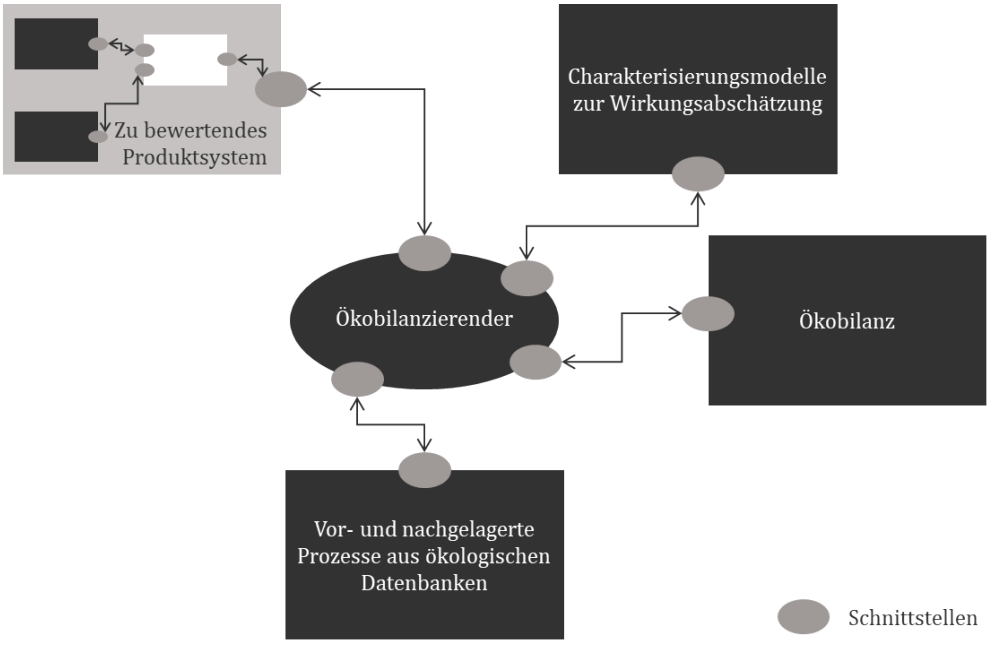


Abbildung 5.5: Schnittstellen in einer Ökobilanzierung sowie verknüpfte Grey- und Black-Box-Modelle.

Die veröffentlichten Ergebnisse von Ökobilanzen können als Black-Box interpretiert werden: so ist es mit den zur Verfügung gestellten Daten in der Regel nicht möglich, das zugrunde gelegte System zu identifizieren oder nachzubilden. Der Ökobilanzierende selbst ist in der Regel zur Verschwiegenheit verpflichtet und darf in vielen Fällen nur sehr begrenzt weitere Informationen als die möglicherweise veröffentlichten Ergebnisse preisgeben.

5.3.5 Multifunktionale Systeme

Die Festlegung einer funktionellen Einheit als Bezugseinheit spielt eine wesentliche Rolle in der Ökobilanzierung. Der quantifizierte Nutzen eines Produktsystems kann zum einen durch das Auftreten von Zusatzprodukten weitere Funktionen aufweisen. Zum anderen wird die funktionelle Einheit eines Produktsystems durch wechselnde Besitzer in unterschiedlichen Lebenswegabschnitten geprägt.

Durch das Auftreten von Zusatzprodukten kann das betrachtete Produktsystem weitere Funktionen aufweisen. Haben anfallende Zusatzprodukte intern keine Verwendung, erfüllen jedoch für andere Produktsysteme einen Zweck und können daher anderweitig verwendet oder verwertet werden, ist dies in der Regel mit weiteren Prozessen zur Aufbereitung und einhergehenden Umweltbelastungen verbunden. Während der Abfall noch dem verursachenden System zugeordnet werden kann, gibt es bei Zusatzprodukten, die an anderer Stelle Anwendung finden, ein Zuordnungsproblem. Oftmals steht zur Diskussion, ob die Aufbereitungsprozesse beiden Systemen jeweils zur Hälfte zugeordnet werden sollen, d. h. dem abgebenden als auch dem annehmenden System. Manche rechnen die Reyzklierprozesse ganz dem verursachenden System an, andere ganz dem abnehmenden System.

Es gibt zwei grundsätzliche Ansätze, wie die Umweltwirkungen von Zusatzprodukten in der ökologischen Bilanzierung berücksichtigt werden können, wenn diese nicht direkt den einzelnen Produkten zugeordnet werden können: die „Systemerweiterung“ und die „Allokation“. Das laut Norm zu priorisierende Verfahren ist die „Systemerweiterung“ (vgl. [113], S. 61 ff.), die sich in das „Warenkorbverfahren“ und die „Gutschriftenmethode“ unterteilen lässt (vgl. auch Abbildung 5.6).

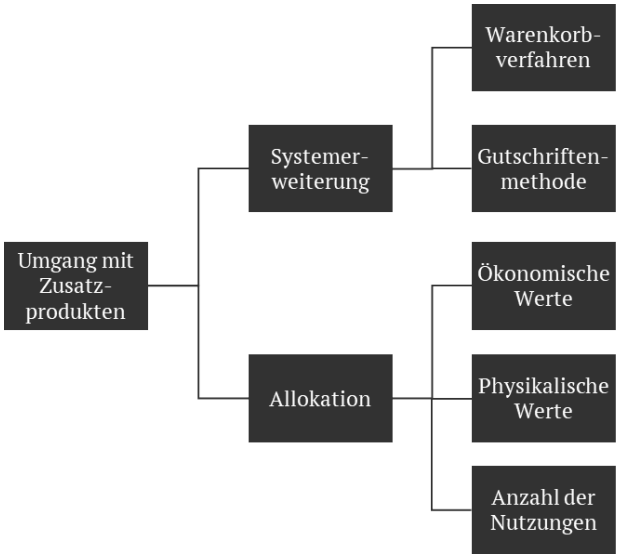


Abbildung 5.6: Methoden zum Umgang mit Zusatzprodukten.

.

Bei der „Gutschriftenmethode“ wird ein äquivalentes Produktsystem gesucht, welches funktionell gleichwertig ist zum entstandenen Zusatzprodukt (vgl. z. B. [113], S. 612 ff.). Dieses wird dann vom modellierten Produktsystem als Gutschrift abgezogen - mit der Annahme, dass durch das Zusatzprodukt die äquivalente Herstellung über das Produktsystem vermieden wird (vgl. auch Abbildung 5.7). Lasten, die durch Aufbereitungsprozesse entstehen, sind dann ebenfalls dem modellierten Produktsystem zuzuordnen.

Beim „Warenkorbverfahren“ wird hingegen das äquivalente Produktsystem, welches funktionell gleichwertig ist zur Funktion des entstandenen Zusatzprodukts, den anderen, zum Vergleich dienenden, Produktsystemen hinzugefügt, um eine gleichwertige Vergleichsbasis zu schaffen (vgl. auch Abbildung 5.8).

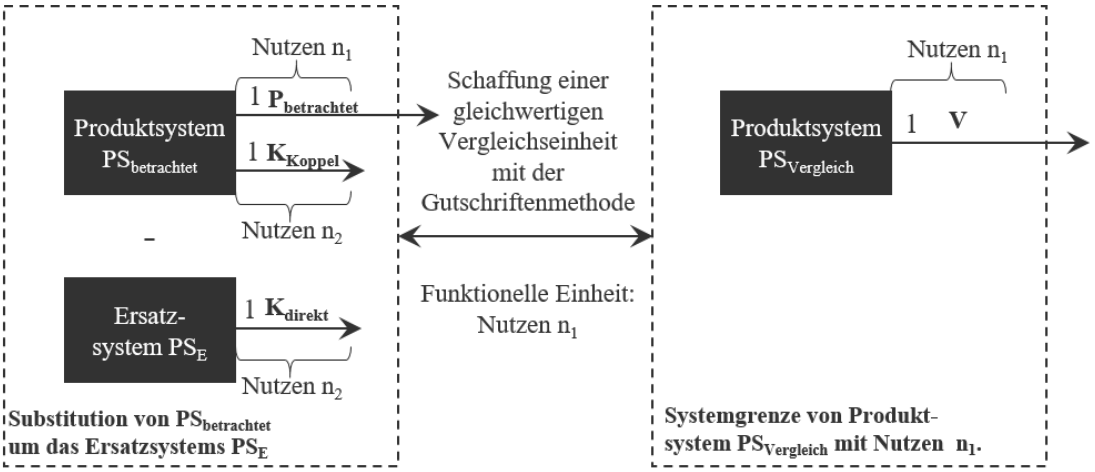


Abbildung 5.7: Grafische Darstellung der Gutschriftenmethode.

.

Da ein gesonderter Vergleich der ursprünglich festgelegten funktionellen Einheit, die sich allein aus dem Hauptprodukt ableiten lässt, nicht möglich ist, kann die Aussagekraft der Ergebnisse reduziert sein. In der für die Umweltproduktdeklarationen von Bauprodukten spezifizierten Norm DIN EN 15804 (vgl. [116], S. 38) hingegen soll für nicht eindeutig zu den einzelnen Koppelprodukten zuordenbare Flüsse bzw. Prozesse vorrangig die Allokation verwendet werden (Anmerkung des Autors: In [116] werden Koppelprodukte als „Co-Produkte“ bezeichnet).

Die Methodik der Systemerweiterung verfolgt den Ansatz, das Zusatzprodukt aus dem Betrachtungsrahmen des modellierten Produktsystems zu nehmen. Dabei wird nach festgelegten Allokationsregeln ein Teil der Inputs und Outputs dem Produktsystem des Zusatzproduktes zugeordnet, welches sich dann außerhalb der Systemgrenze des betrachteten Produktsystems befindet. Allen Methoden ist gemein, dass sie die Ergebnisse der Wirkungsabschätzung wesentlich beeinflussen können - so werden entweder Substitutionen (bei der Allokation und der Gutschriftenmethode) am *betrachteten* Produktsystem oder Additionen (beim Warenkorbverfahren) an *anderen* Produktsystemen vorgenommen.

Dies birgt das Risiko, dass tatsächlich und real auftretende Umweltbelastungen so *zwischen die Grenzen beider* Systeme geraten und durch Allokationsregeln und Gutschriften *keinem* der beiden bilanzierten Systeme zugeordnet werden. Ungeachtet der „Belohnungen“ für die Verwendung von Sekundärstoffen im Sinne der politischen Bestrebungen einer Kreislaufwirtschaft sollte die Zuordnung an den realen Gegebenheiten der Systeme orientiert sein: gibt ein Unternehmen ein Produkt an ein anderes Unternehmen ab, das es sonst hätte entsorgen müssen, wird es „belohnt“, indem es tatsächlich die Umweltbelastungen für die Entsorgung vermeidet. Die Aufbereitungsprozesse sind die Vorbereitung zur weiteren Verwendung und sollten daher dem annehmenden Produktsystem zugeordnet werden. Dem annehmenden Produktsystem kommt dies insofern zu Gute, dass es durch den Einsatz von Sekundärstoffen die Umweltbelastungen durch die Gewinnung von Primärstoffen vermeidet. Nur wenn tatsächlich Abfall von bspw. einer Deponie bezogen wird, sollte eine Gutschrift berücksichtigt werden: dem annehmendem Produktsystem wird die Umweltbelastung abgezogen, die anderenfalls durch die Deponierung des Abfalls zu erwarten gewesen wäre. Grund: entsorgt ein Unternehmen ein Produkt, wird diesem in einer Ökobilanz auch die Entsorgung angerechnet. Wird ein bereits deponiertes Produkt aus einer Deponie entnommen, entfällt auch die Umweltbelastung, weshalb sie dann vom annehmenden System substituiert werden kann.

Im Hinblick auf das Bestreben, die Stoffkreisläufe zu schließen, werben Hersteller zunehmend mit hohen Rezyklierungs- und Verwertungsraten. Doch der Nutzen eines Produktes wird im Wesentlichen von seinem Be-

sitzer bestimmt – und dieser wechselt entlang des Lebenswegs eines Produkts. Hersteller, Nutzer und Entsorger bzw. Rezyklierer eines Produktes unterscheiden sich in der Regel. Hersteller erzeugen Produkte mit dem Ziel, diese gewinnbringend zu vermarkten. Dabei soll das Produkt primär Kriterien aufweisen, welche die Wettbewerbsfähigkeit zu anderen Produkten *in der Wirtschaft* sicherstellen. Entsorger und Rezyklierer haben ebenso das Ziel, die Sortier- und Müllverbrennungsanlagen möglichst effizient zu betreiben.

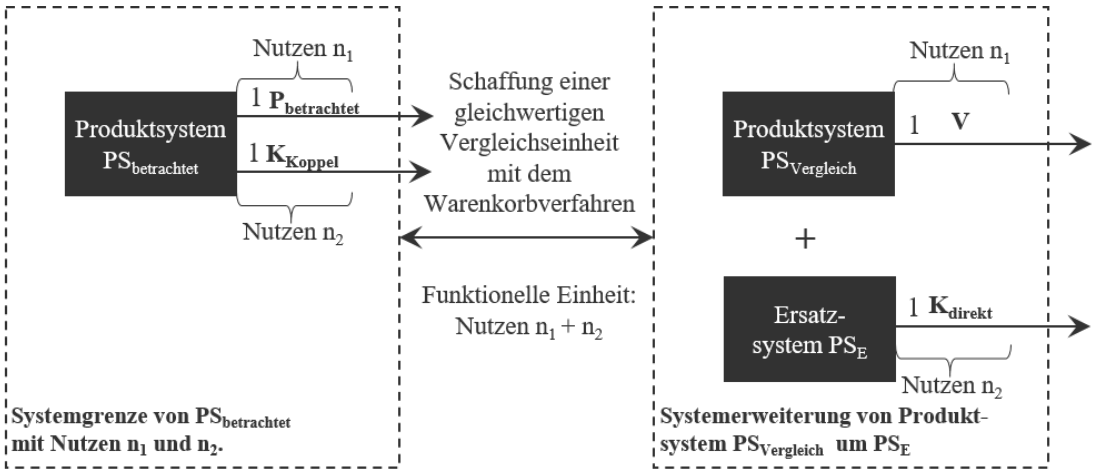


Abbildung 5.8: Grafische Darstellung des Warenkorbverfahrens.

.

Nach dem Erwerb eines Produktes liegt die die Entscheidung jedoch, ob und wie lange ein Produkt auf welche Art und Weise verwendet wird, in der Regel in der Hand des individuellen Besitzers, respektive *einem Teil der Gesellschaft*. Aktuell ist in der deutschen Bevölkerung ein zunehmend gesteigertes Interesse zu erkennen, bewusst ökologisch – oder zumindest nachhaltig – agieren zu wollen. Die *Wahrnehmung* des eigenen ökologischen Handelns und die *Einstellung* dazu können dabei z. B. für die Nutzungsdauer und -intensität eines Produktes stets eine wesentliche Rolle spielen; auch wenn sie meist wirtschaftlichen und gesellschaftlichen Aspekten untergeordnet ist. Dabei liegen nach Ansicht des Autors derzeit noch mitunter erhebliche Diskrepanzen zwischen der *objektiv berechenbaren* „Ökologie“ eines Produktes und der *subjektiv wahrgenommenen* „Ökologie“ desselben Produktes vor. Um ein *vermeintlich* ökologisches, jedoch *tatsächlich* unökologisches Handeln zu vermeiden, ist es evident, dass ein freier, umfassender, und unbedingt wissenschaftlicher Informationsfluss zur Ökologie von Produkten geschaffen wird – möglichst unbeeinflusst von wirtschaftlichen und gesellschaftlichen Interessen.

5.4 Berechnungsansätze

Zur Berechnung haben sich in der Ökobilanzierung zwei Methoden etabliert: das sog. „sequentielle Verfahren“ und das „matrixbasierte Verfahren“ (vgl. z. B. [128], S. 219 sowie [129], S. 84). Das in dieser Arbeit entwickelte Berechnungsverfahren beruht auf dem matrixbasierten Ansatz.

5.4.1 Sequentielles Verfahren

Bei der sequentiellen Methode wird die Sachbilanz des Produktsystems schrittweise ausgehend vom letzten Prozessmodul ermittelt. Dieser Ansatz wird in dieser Arbeit nicht weiter verfolgt. Der Grund hierfür ist, dass im Zuge der Berechnung dieser Verfahren bei komplexeren Produktsystemstrukturen mit Schleifen wie bspw. Rezyklierprozessen Iterationen durchgeführt werden. Dies gilt bei Anwendung der Intervallarithmetik und dem Ziel, möglichst

schmale Intervallweiten zu berechnen, als weniger geeignet (Problematik der iterativen Verfahren vgl. hierzu auch Kapitel 3.7).

5.4.2 Matrixbasiertes Verfahren

Das im Folgenden erläuterte matrixbasierte Verfahren zur Ökobilanzierung basiert hauptsächlich auf den in [97], S. 11-23 sowie S. 168 f., [130], S. 2-4 und [96], S. 50-57 beschriebenen Grundsätzen, wurde jedoch teilweise durch eigene Definitionen ergänzt. Das Verfahren beruht im Wesentlichen auf drei Matrizen sowie vier Vektoren, die hier wie folgt definiert werden: die Produktmatrix *P*, die Umweltmatrix *B* und die Charakterisierungsmatrix *Q* sowie der Bedarfsvektor f, der Skalierungsvektor s, der Umweltvektor g und der Wirkungsvektor h. Die Produktflüsse werden in der Produktmatrix *P* hinterlegt und die Elementarflüsse analog dazu in der Umweltmatrix *B*.

In der matrixbasierten Berechnung zur ökologischen Bilanzierung liegen in der Regel quadratische Koeffizientenmatrizen mit vollem Rang vor, was durch Modellierungskriterien gewährleistet werden kann (Modellierungskriterien vgl. Kapitel 8.4). Die resultierenden Gleichungssysteme sind inhomogen, da sie aufgrund des bilanziellen Ansatzes auf der rechten Seite Elemente aufweisen, die ungleich null sind.

Vorzeichenkonvention

Bei der matrixbasierten Berechnung tragen Inputs ein negatives Vorzeichen und Outputs ein positives Vorzeichen (vgl. z. B. [96], S. 48).

Produktmatrix P

Die Produktmatrix *P* enthält die Inputs und Outputs der Hauptproduktflüsse der einzelnen Prozessmodule des Produktsystems. Dabei entsprechen die Spalten der Matrix den Prozessmodulen und die Zeilen der Matrix den Hauptproduktflüssen. Die Koeffizienten der Matrix sind die Mengen der in den einzelnen Prozessmodulen jeweils benötigten oder erzeugten Produkte. In [97], S. 14 wird die Produktmatrix als Matrix *A* bezeichnet, in [96], S 53. f. als Technologiematrix.

Umweltmatrix B

Die Umweltmatrix *B* enthält die Inputs und Outputs der Elementarflüsse sowie der Zusatzproduktflüsse des Produktsystems. Dabei entsprechen die Spalten der Matrix den Prozessmodulen und die Zeilen der Matrix den Elementar - oder Zusatzproduktflüssen. Die Koeffizienten der Matrix stellen die Mengen der Flüsse dar. In [97], S. 14 wird diese Matrix als „intervention matrix B“ bezeichnet, in [96], S 53. f. als „Umwelteinwirkungsmatrix“. Es ist auch möglich, die Zusatzproduktflüsse in einer separaten Matrix, der Zusatzproduktmatrix *Z*, abzubilden; diese ist aktuell im Java-Paket *Ivari* nicht hinterlegt.

Charakterisierungsmatrix Q

Die Charakterisierungsmatrix *Q* (vgl. auch [130], S. 4 und [97], S. 168) enthält die Charakterisierungsfaktoren der einzelnen Elementarflüsse auf die jeweiligen Wirkungsindikatoren. Dabei entsprechen die Spalten der Matrix den Elementarflüssen und die Zeilen den Wirkungsindikatoren.

Bedarfsvektor f

Der Bedarfsvektor f (vgl. auch [130], S. 4 sowie „vector f“ in [97], S. 14 f.) beschreibt die aus dem Produktsystem resultierenden Endprodukte, wobei die Zeilen des Vektors den Hauptproduktflüssen entsprechen. Die Koeffizienten des Vektors sind die Mengen der durch das Produktsystem zu erzielenden Endprodukte. Im Normalfall ist der Spaltenvektor ein Einheitsvektor oder ein Vielfaches davon.

Skalierungsvektor s

Der Skalierungsvektor s (vgl. auch [130], S. 4 sowie „vector s“ in [97], S. 16 f.) beschreibt die Skalierungsfaktoren der einzelnen Prozessmodule des Produktsystems, wobei die Zeilen des Vektors den Prozessmodulen entsprechen. Der Skalierungsvektor s kann als Produkt der Inversen P^{-1} der Produktmatrix P mit dem Bedarfsvektor f definiert werden:

$$s = \mathrm{P}^{-1} \cdot f \tag{5.1}$$

Die Koeffizienten des Skalierungsvektors s geben an, wie oft die einzelnen Prozessmodule durchgeführt werden müssen, um die im Bedarfsvektor festgelegte Einheit des Endproduktes bzw. der Endprodukte herstellen zu können.

Umweltvektor g

Der Umweltvektor g (vgl. auch [130], S. 4 sowie „vector g“ in [97], S. 18 f.) beschreibt die aufsummierten Elementarflüsse sowie Zusatzproduktflüsse des Produktsystems, wobei die Zeilen des Vektors den Elementarflüssen bzw. den Zusatzproduktflüssen des Produktsystems entsprechen. Der Umweltvektor g kann als Produkt der Umweltmatrix B mit dem Skalierungsvektor s definiert werden:

$$g = \mathrm{B} \cdot s \tag{5.2}$$

Die Koeffizienten des Umweltvektors g geben die aufsummierte Menge der einzelnen Elementar- bzw. Zusatzproduktflüsse an, welche zur Deckung des festgelegten Bedarfs im Zuge des Produktsystems zu erwarten sind. Es ist auch möglich, die Zusatzproduktflüsse in einem separaten Vektor, dem Zusatzproduktvektor z, abzubilden; dieser ist aktuell im Java-Paket *Ivari* nicht hinterlegt.

Wirkungsvektor h

Der Wirkungsvektor h (vgl. auch [130], S. 4 sowie „impact vector h“ in [97], S. 168 f.) beschreibt die aufsummierten potentiellen Umweltwirkungen des Produktsystems, wobei die Zeilen des Vektors den Wirkungsindikatoren entsprechen. Der Umweltvektor h kann als Produkt der Charakterisierungsmatrix Q mit dem Umweltvektor g definiert werden:

$$h = \mathrm{Q} \cdot g \tag{5.3}$$

Die Koeffizienten des Umweltvektors h geben die aufsummierte Menge der einzelnen Wirkungsindikatorwerte der jeweiligen Wirkungsindikatoren an, welche aufgrund der Elementarflüsse im Zuge des Produktsystems zu erwarten sind.

5.5 Werkzeuge: Software und Datenbanken

Es gibt eine Reihe von Ökobilanzierungsprogrammen und Datenbanken, die frei verfügbar oder kostenpflichtig sind und in der Regel auf einem und/oder beiden Berechnungsverfahren basieren. Bei Anwendung des herkömmlichen Black-Box-Ansatzes zur Durchführung von Ökobilanzen sind vorwiegend Programme notwendig, mit denen Daten verwaltet, verarbeitet, visualisiert und analysiert werden können. Sie greifen auf große Datenbanken zu, in denen seit Beginn der Ökobilanzierung ein stetiger Zuwachs von Datensätzen zu verzeichnen ist (vgl. auch [94], S. 263).

5.5.1 Etablierte Programme und Datenbanken

Im Folgenden werden exemplarisch eine Auswahl von verfügbaren Programmen und Datenbanken aufgeführt. Weitere Informationen bietet z. B. [96], S. 184-187. *OpenLCA* (vgl. [131]) ist ein kostenfreies Java-Programm, welches sowohl die sequentielle als auch die matrixbasierte Methode verwendet (vgl. [132], S. 13. ff.) und stetig weiterentwickelt wird. *CMLCA* (vgl. [133]) ist eine kostenfreie Software und wird von der Universität Leiden zur Verfügung gestellt. *SimaPro* (vgl. [134]), *Umberto* (vgl. [135]) und *GaBi* (vgl. [136]) sind kostenpflichtige Ökobilanzierungsprogramme. Beispiele für kostenfreie Datenbanken sind *ProBas* (vgl. [137]) vom Umweltbundesamt sowie *Ökobaudat* (vgl. [138]) des Bundesministeriums des Innern, für Bau und Heimat. Das Institut Bauen und Umwelt e.V. stellt kostenfrei eine Datenbank veröffentlichter Umweltproduktdeklarationen zur Verfügung (vgl. [139]). *Ecoinvent* (vgl. [140]) ist eine kostenpflichtige Sachbilanz-Datenbank. Der Anbieter von *GaBi* stellt ebenfalls eine Reihe von kostenpflichtigen Datenbanken zur Verfügung.

5.5.2 MultiVaLCA

Das Java-Programm *MultiVaLCA* („**Multi Va**lue **L**ife **C**ycle **A**ssessment“) (vgl. [141]) stellt eine grafische Nutzeroberfläche bereit, mit der Ökobilanzen intervallbasiert durchgeführt werden können. Es wird stetig am Institut für Werkstoffe im Bauwesen der Universität Stuttgart (vgl. [142]) weiterentwickelt. Das in dieser Arbeit vorgestellte Berechnungsverfahren wurde gesondert in *Ivari* implementiert und kann in *MultiVaLCA* integriert werden.

In *MultiVaLCA* können Elementar- und Produktflüsse erstellt werden. Die Flüsse können quantifiziert Prozessmodulen zugeordnet werden (vgl. auch Abbildung 5.9). Die Produktsysteme setzen sich zusammen aus Prozessmodulen und/oder Subsystemen; dabei können auch Prozessmodulgruppen sowie Subsystemgruppen hinzugefügt werden (Erläuterung zu „Gruppen“ vgl. auch Kapitel 5.2.18). Sowohl für die Flüsse als auch für die Charakterisierungsfaktoren können Unter- und Obergrenzen definiert werden. Zusätzlich zu den Intervallgrenzen ist es in *MultiVaLCA* möglich, einen Hauptwert anzugeben. Während die Intervallgrenzen zur Erfassung möglicher Unsicherheiten als erweiterter Ansatz der ökologischen Bilanzierung aufgefasst werden können, ist der Hauptwert der „typische“ Wert, der in der herkömmlichen Ökobilanzierung verwendet wird bzw. verwendet werden würde. Derzeit ist auch die Eingabe von Ergebnissen aus Wirkungsabschätzungen möglich. Dieses Vorgehen wird auch in anderen Ökobilanzprogrammen unterstützt, birgt jedoch die Gefahr, dass in einem Produktsystem Datensätze verwendet werden, die auf veralteten Charakterisierungsmodellen basieren. Dadurch können nicht erfasste Ungenauigkeiten in das Modell eingetragen werden.

Es ist empfehlenswert, nur generische *Sachbilanzdatensätze* zu verwenden. So können alle Prozesse in der Wirkungsabschätzung mit denselben Charakterisierungsmodellen berechnet und Unsicherheiten bei den Charakterisierungsfaktoren als Intervalle berücksichtigt werden. Dabei ist jedoch zu beachten, dass auch diese Sachbilanzdatensätze auf Basis des intervallbasierten Ansatzes erfasst werden sollten.

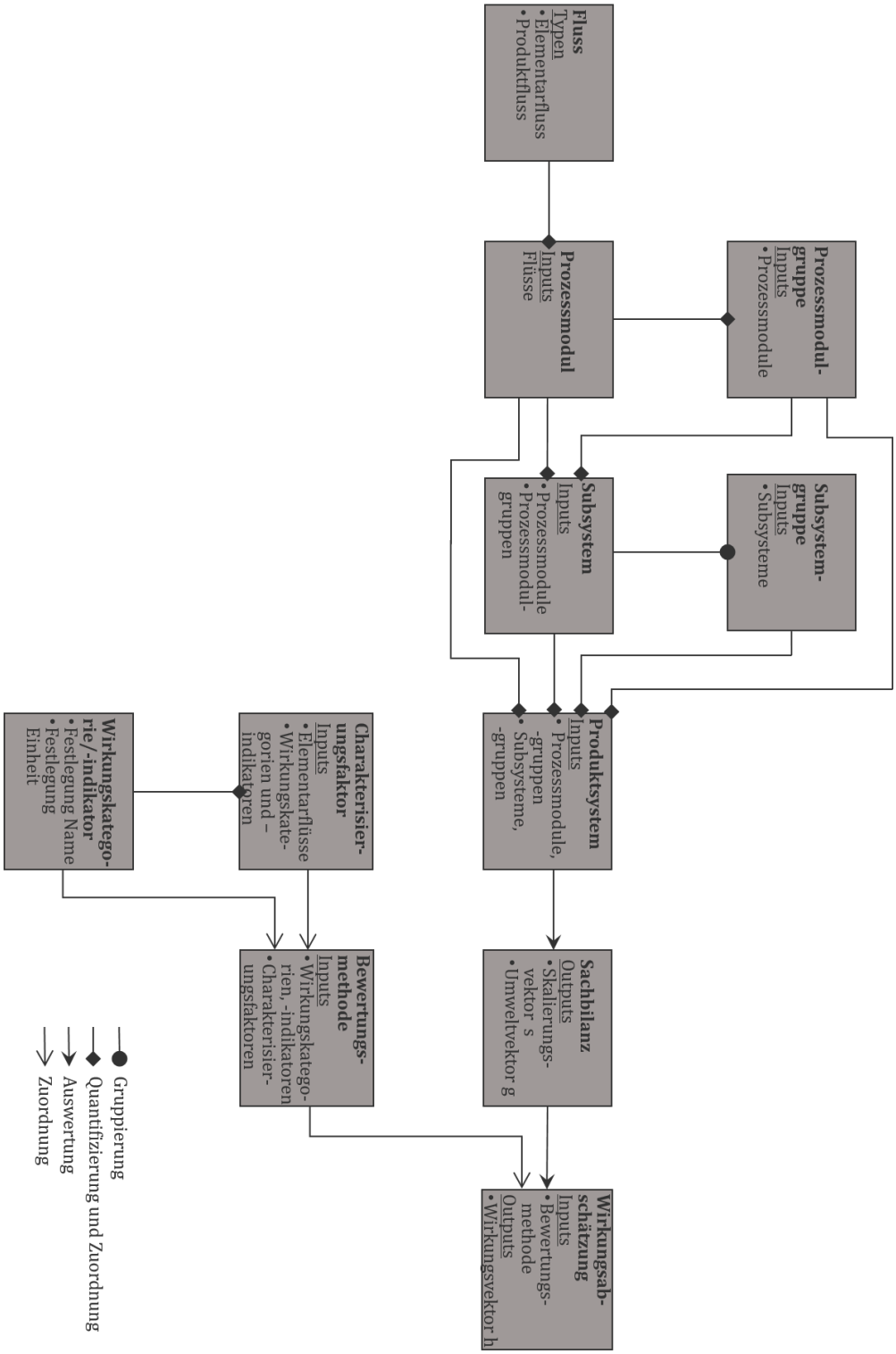


Abbildung 5.9: Aufbau des Programms *MultiVaLCA* zur ökologischen Bilanzierung.

.

Kapitel 6

Wissenschaftlicher Umgang mit Unsicherheit

Unsicherheit spielt in vielen wissenschaftlichen Bereichen eine bedeutende Rolle, wobei diese und deren Ursachen je nach Forschungsschwerpunkt und einhergehender Forschungsmethoden variieren können. Dies wird in Kapitel 6.1 erörtert. Die Vielzahl auftretender und nicht eliminierbarer Unsicherheiten ist auch in der Ökobilanzierung von großer Bedeutung und wird in Kapitel 6.2 erläutert. Dabei gibt es (noch) keine allgemeingültige Definition oder festgelegte Gliederung und Bezeichnung der Arten von Unsicherheit, weswegen für diese Arbeit eine sachgerechte Definition und Einordnung vorgenommen wird.

Der aktuelle Stand zur Analyse von Unsicherheiten, eine Auswahl von in der Ökobilanzierung praktizierten Methoden sowie in der Literatur vorgeschlagene Ansätze werden nachfolgend in Kapitel 6.3 vorgestellt und anschließend in Kapitel 6.4 ihre Anwendbarkeit diskutiert. In diesem Zusammenhang wird auch die intervallarithmetische Methodik zur Berücksichtigung von den in der Ökobilanzierung vorkommenden Unsicherheiten sowie dem aktuellen Umgang mit Unsicherheit hinsichtlich ihrer Praktikabilität und Eignung eingeordnet.

6.1 Unsicherheit in der Wissenschaft

Unsicherheit und ihre Ursachen können je nach Forschungsschwerpunkt und einhergehender Methoden unterschiedlich aufgefasst und behandelt werden (vgl. z. B. einen interdisziplinären Diskurs zur Unsicherheit in [143]). Nachfolgend wird in Kapitel 6.1.1 kurz auf die allgemeine Wertung von Unsicherheit eingegangen, einige typische Unsicherheiten aus verschiedenen Wissenschaftsbereichen in Kapitel 6.1.2 aufgezeigt und abschließend kritische Standpunkte aus der Wissenschaft mit dem allgemeinen Umgang von Unsicherheit wiedergegeben.

6.1.1 Allgemeine Wertung von Unsicherheit

Wissenschaftliche Aussagen sollten deutlich und präzise formuliert werden (vgl. z. B. [144], S. 102 f.). Oftmals werden daher auch *komplexe*, sehr detaillierte Modelle sowie *präzise* Ergebnisse mit den daraus mitunter ableitbaren *sehr genauen* Aussagen als *wissenschaftlich* und *sicher* angesehen. Bei sehr präzisen Kenntnissen über ein System mögen sehr genaue Abbildungen und Resultate möglich sein. Sobald jedoch weniger Systemkenntnisse vorliegen und vermehrt mit Unsicherheit umgegangen werden muss, führen präzisere Modellbildungen nicht notwendigerweise zu „sichereren“ Ergebnissen, obgleich die präziseren Resultate dies suggerieren.

Dies zeigt auch Abbildung 6.1 (angelehnt an [145], S. 141, Fig. 2; Anmerkung des Autors: die Bezeichnung „Fehler“ ist in [145] als „Unsicherheit“ zu verstehen und widerspricht der in dieser Arbeit vorgenommenen Definition eines Fehlers).

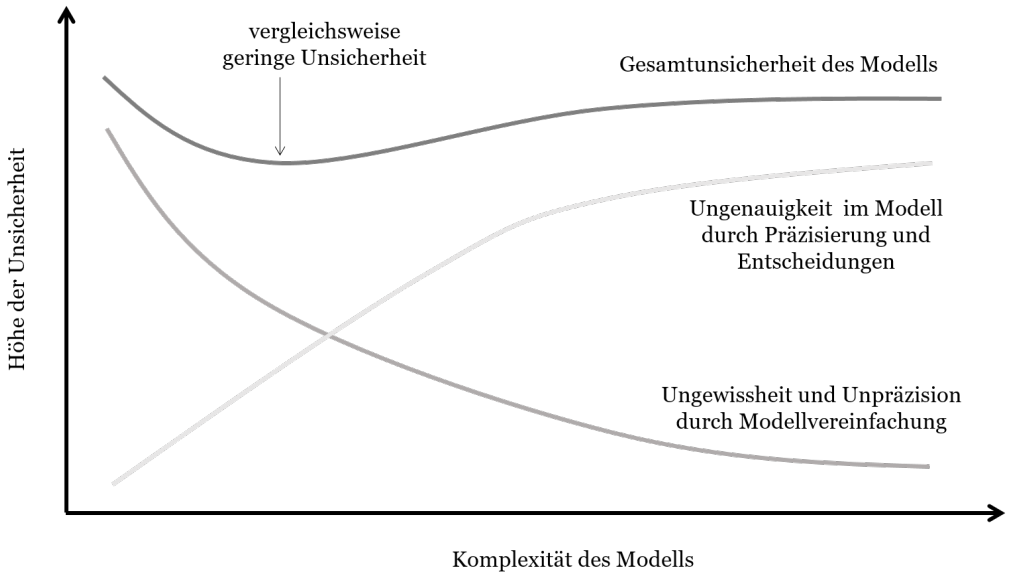


Abbildung 6.1: Komplexität des Modells und Größe von Unsicherheit angelehnt an [145], S. 141, Fig. 2.

So ist bei der Modellierung realer Systeme unabhängig des Komplexitätsgrades stets Unsicherheit zu erwarten. Zwar ist anzunehmen, dass Unsicherheit in Form von Ungewissheit und Unpräzision bei einer detaillierten Systemkenntnis und daraus resultierenden präzisen Modellbildung tendenziell abnehmen - die Ungenauigkeit im Modell kann durch Präzisierung jedoch auch zunehmen (Erläuterung zu „Unpräzision“, „Ungenauigkeit“ bzw. „Ungewissheit“ vgl. Kapitel 6.2.3, Kapitel 6.2.4 bzw. Kapitel 6.2.5). Daher sollten vage Ergebnisse aufgrund berücksichtigter Unsicherheiten und die daraus weniger exakt formulierbaren Aussagen nicht als weniger wissenschaftlich angesehen werden.

Aus verschiedenen Bereichen der Wissenschaft gibt es Anregungen, die sich gegen rein probabilistische Ansätze zur Abschätzung von Unsicherheiten aussprechen. Aus dem Bereich der Klimawissenschaften thematisieren Hillerbrand et al. (vgl. [146], S. 151 ff.) Unsicherheiten bei der Projektion zukünftiger Klimata. In der Klimaforschung werden Annahmen über die künftige Entwicklung von sozialen und wirtschaftlichen Faktoren verwendet, um auf Basis dieser das zukünftige Klima abzuschätzen. Die Ergebnisse sind als Projektionen zu verstehen und nicht als Vorhersagen. So werden bei der Abschätzung zukünftiger Klimata eine Reihe von Faktoren und Annahmen verwendet, die über rein naturwissenschaftliche Zusammenhänge hinausgehen (vgl. [146], S. 154). Projektionen stellen im Vergleich zu Vorhersagen vagere Resultate dar. Hillerbrand et al. zufolge lassen sich nicht alle Unsicherheiten über Wahrscheinlichkeitsverteilungen abbilden (vgl. [146], S. 159), wobei die nicht-quantifizierten Unsicherheiten die Ableitung von Aussagen erschweren. Dies erfordert mehr Forschung zur Erfassung und Beurteilung von Unsicherheiten mit nicht-probabilistischen Ansätzen (vgl. [146], S. 172 f.).

Von Bereich der Mathematik und Statistik ausgehend thematisieren Viertl et al. mathematische Modelle zur Abbildung von Unsicherheiten (vgl. [147], S. 271 ff.). Auch sie vertreten demnach die Meinung, dass stochastische Modelle nicht alle Arten von Unsicherheiten beschreiben können (vgl. [147], S. 271). Die Anwendbarkeit von derlei Modellen wird u. a. auf variable, zufällig abweichende Daten eingegrenzt. Für Unsicherheit aufgrund unscharfer

Daten werden unscharfe Wahrscheinlichkeitsverteilungen empfohlen (vgl. [147], S. 279 f.). Ungewissheit über zutreffende Wahrscheinlichkeits-, Nutzen oder Verlustfunktionen können mit Fuzzy-Modellen erfasst werden (vgl. [147], S. 272 f.).

Aus dem Bereich der philosophischen Logik vertreten Bradley et al. die These, dass unterschiedliche Arten von Unsicherheit zwar auf empirisch-faktische und mit Wahrscheinlichkeitsverteilungen erfassbare Zustandsunsicherheiten reduziert werden können, dies aber gleichsam die Unsicherheit vergrößert (vgl. [148], S. 1226 f.). Wenn viele verschiedene Arten von Unsicherheit vorliegen, deren Zustände aufgrund mangelnder Daten nicht bewertet werden können, kommen die herkömmlichen Methoden jedoch in der Praxis an ihre Grenzen und erschweren die Entscheidungsfindung (vgl. [149], S. 109). Aus dem Bereich der Soziologie nennen Kron et al. die zunehmende Komplexität als einen Grund für die Zunahme von Unsicherheit ([150], S. 55 ff.). Die steigende Anzahl von Zuständen und Ereignissen führen demzufolge zu vermehrten Wechselwirkungen, weswegen Trennlinien immer schwieriger herausgearbeitet werden können. Solche fließenden und unbekannten Übergänge können als „Hybride“ bezeichnet werden (vgl. [150], S. 57).

Die Unsicherheit und ihre quantitative Erfassbarkeit wird in verschiedenen Wissenschaftsbereichen diskutiert. Ein Lösungsansatz ist, dass existente, unsichere Informationen, welche nicht eliminierbar sind, direkt in der Berechnung berücksichtigt werden. Hierfür bedarf es Methoden, welche unsichere Daten verarbeiten können.

6.1.2 Unsicherheit in verschiedenen Wissenschaftsbereichen

Unsicherheit zeigt sich im Bereich der der Messtechnik z. B. in Form von Abweichungen gemessener Daten, im Management durch Unsicherheit bei Entscheidungsfragen und bei ingenieur- und naturwissenschaftlichen Simulationen in Form von Modellunsicherheit.

Für den Bereich der Messtechnik wird Unsicherheit differenziert in einen zufälligen sowie einen systematischen Anteil (vgl. [151], S. 22 f.). Zufällige Abweichungen zwischen Messergebnissen reduzieren demnach die *Präzision* eines Wertes (vgl. z. B. „measurement precision“ in [151], S. 22; Anmerkung des Autors: „measurement precision“ wurde vom Autor übersetzt als „(Mess-)Präzision“) und können auf die Natur der zu messenden Größe oder auf begrenzte Möglichkeiten der Erfassbarkeit zurückgeführt werden, während systematische Abweichungen ursächlich durch den Messvorgang verursacht werden und die *Genauigkeit* eines Wertes reduzieren (vgl. z. B. „measurement trueness“ in [151], S. 21; Anmerkung des Autors: „measurement trueness“ wurde vom Autor übersetzt als „(Mess-)Genauigkeit“).

In der klassischen Entscheidungstheorie wird unterschieden in Entscheidungen unter Sicherheit und Entscheidungen unter Unsicherheit (vgl. z. B. Amann.2019, S. 23 f.); Entscheidungen, die unter Unsicherheit getroffen werden, werden differenziert in Entscheidung unter Risiko und Entscheidung unter Ungewissheit. Bei beiden Fällen von Unsicherheit sind die möglichen Zustände, die eintreten können, sowie die Folgen bekannt; bei Entscheidungen unter Ungewissheit jedoch ist die Eintrittswahrscheinlichkeit der Zustände nicht bekannt. Dazu kann auch zwischen objektiver und subjektiver Unsicherheit differenziert werden (vgl. z. B. [152], S. 893 f.). Es können drei Dimensionen der Unsicherheit im Kontext von Entscheidungen allgemein (vgl. [148]) sowie in Bezug auf modellbasierte Entscheidungen (vgl. [153]) erfasst werden:

- die „Natur“,
- das „Objekt“ (vgl. „object“ in [148], S. 1230) bzw. der „Ort“ (vgl. „location“ in [153], S. 9 ff.)
- und der „Schweregrad“

von Unsicherheit (Anmerkung: die Bezeichnungen der oben aufgeführten Dimensionen wurden vom Autor aus dem Englischen ins Deutsche übersetzt). Die Unsicherheit liegt demnach innerhalb dieses dreidimensionalen

Konstrukts. Die Dimension „Natur“ von Unsicherheit ist gemäß Bradley et al. empirisch (Ist-Zustand), modal (Kann-Zustand) oder normativ (Soll-Zustand) (vgl. [148], S. 1229 f.). Laut Walker et al. (vgl. [153], S. 13 f.) ist sie wissensbedingt oder natürlichen Ursprungs, was Tannert et al. auch als subjektive oder objektive Unsicherheit bezeichnen (vgl. [152], S. 893 f.). Die Dimension „Objekt“ beschreibt die Fakten der realen Welt bzw. mögliche Fakten einer veränderten Welt (vgl. [148], S. 1230) bzw. die stattdessen von Walker et al. formulierte Dimension „Ort“ (vgl. [153], S. 9 ff.) den Ort des Auftretens. Die Dimension „Ort“ besteht nach Walker et al. aus den Bereichen „Kontext“, „Modell“, „Inputs“, „Parameter“ sowie „Outputs“ (vgl. [153], S. 9 ff.). Der „Kontext“ beschreibt Unsicherheit, die außerhalb des Modells vorzufinden ist, wie bspw. Unsicherheit darüber, wie die Grenzen des betrachteten Systems festgelegt werden sollen. „Modellunsicherheit“ zeigt sich in Unsicherheit über Kenntnisse der Systemstrukturen und Zusammenhänge im betrachteten System oder in Unsicherheit, die der verwendeten Technik zuzuordnen ist, wie z. B. Fehler in den Algorithmen des verwendeten Programms. Unsicherheit der „Inputs“, „Parameter“ und „Outputs“ sind letztlich unsichere Werte entlang des Unsicherheitsflusses (Erläuterung zum „Unsicherheitsfluss“ vgl. Kapitel 6.2.8). Der „Schweregrad“ repräsentiert die Größe der Unsicherheit.

Bradley et al. leiteten anhand der von ihnen bestimmten Dimensionen (vgl. [148], S. 1229 f.) qualitativ voneinander unterscheidbare Arten von Unsicherheit ab. Die Schlussfolgerung der Autoren ist, dass in vielen Wissenschaftsbereichen vorwiegend eine Art von Unsicherheit berücksichtigt wird, welche die Autoren als faktisch-empirische Unsicherheit geringer Größe (sog. „milde Zustandsunsicherheit“) bezeichnen (vgl. [148], S. 1231 f.; Anmerkung: Bezeichnung wurde vom Autor aus dem Englischen ins Deutsche übersetzt). Diese Art von Unsicherheit kann durch Wahrscheinlichkeiten beschrieben werden. Tatsächlich liegen ihnen zufolge jedoch weit mehr Arten von Unsicherheiten vor, die sich qualitativ von dieser Art von Unsicherheit unterscheiden.

6.2 Unsicherheit in der Ökobilanzierung

Die ökologische Bilanzierung vereint unterschiedliche Wissenschaften. Dies hat zur Folge, dass Informationen - und damit auch unsichere - aus dem natur- und ingenieurwissenschaftlichen Bereich benötigt werden, wobei gleichsam u. a. ökonomische und soziale Aspekte aufgrund des dreidimensionalen Verständnisses der nachhaltigen Entwicklung (Einordnung der Ökobilanzierung in die Dimensionen der Nachhaltigkeit vgl. auch Kapitel 5.1) eine Rolle spielen. Die Bedeutung von Unsicherheit in der Ökobilanzierung wird in Kapitel 6.2.1 thematisiert. In dieser Arbeit wird Unsicherheit differenziert in *Unpräzision, Ungenauigkeit* und *Ungewissheit*, wobei diese allgemein als *Unsicherheit* bezeichnet werden können. Diese Einordnung wird in Kapitel 6.2.2 erörtert. Die eingeführten Begriffe der „unpräzisen“, „ungenauen“ und „ungewissen“ Daten werden nachfolgend in Kapitel 6.2.3, Kapitel 6.2.4 und Kapitel 6.2.5 erläutert. Die Definition von Unsicherheit wird in Kapitel 6.2.6 unter Berücksichtigung dieser Klassifikation formuliert. In der Praxis vorliegende, unsichere Informationen weisen in der Regel eine Überlagerung mehrerer Arten von Unsicherheit auf.

6.2.1 Bedeutung von Unsicherheit

Unsicherheit spielt in der Ökobilanzierung eine bedeutende Rolle, da die Ergebnisse in der Regel nicht vernachlässigbare Unsicherheiten aufweisen können (vgl. z. B. [154], S. 15, [155], S. 47, [156], S. 794 f., [157]). Als besonders problematisch einzuordnen ist hierbei, dass die Sicherheit der Ergebnisse durch die exakten Ergebnisdarstellungen oft deutlich überschätzt wird (vgl. z. B. [158], S. 64). Zum einen ist die Relevanz von Unsicherheit im Zuge von Ökobilanzen in der großen Anzahl erforderlicher Informationen begründet (vgl. z. B. [96], S. 38 f.). Dies betrifft auch sehr sensible und streng vertrauliche Daten von Produzenten, die mitunter auch nicht dem Ökobilanzierenden übermittelt werden (können). Hinzu kommt, dass die Durchführung und Veröffentlichung von Ökobilanzen

bisher für Produzenten freiwillig ist. Dies hat zur Folge, dass für die Durchführung von Ökobilanzen aktuell in vielen Fällen häufig (noch) nicht genügend spezifische Informationen verfügbar sind (vgl. z. B. [75], S. 11 f.), auf die während der Modellierung zurückgegriffen werden könnte. Dies gilt insbesondere für vor- und nachgelagerte Prozesse (vgl. z. B. [96], S. 35 f. und S. 176) sowie im erhöhten Maße für innovative Produktsysteme in der frühen Entwicklungsphase (vgl. z. B. (vgl. z. B. [159], S. 369). Mangelnde oder unzureichende Informationen werden mit Annahmen, Schätzungen sowie generischen Datensätzen gedeckt, die mit Unsicherheiten einhergehen. Daher liegen bei der Erstellung von Ökobilanzen vermehrt Unsicherheiten vor, deren Größe in der Regel nicht vernachlässigbar und zugleich häufig nicht ausreichend abschätzbar ist.

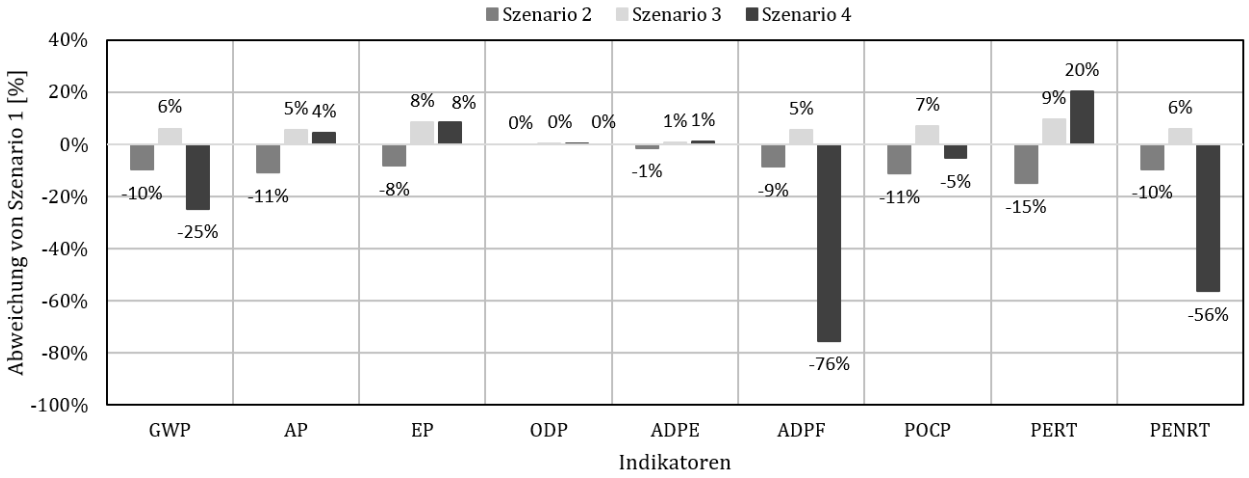
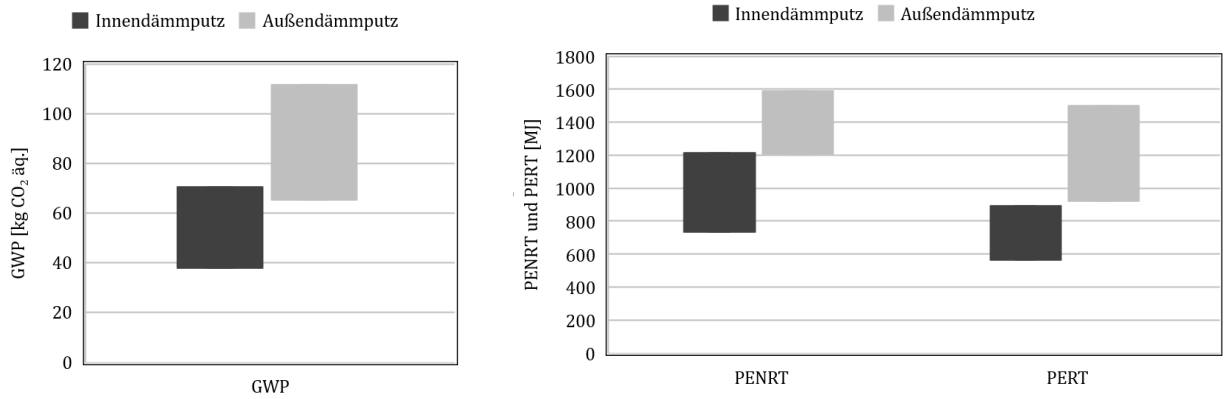


Abbildung 6.2: Abweichung von Ökobilanzergebnissen durch offene Fragestellungen der frühen Entwicklungsphase (Bild angelehnt an [159], Fig. 2.)

So wurde in Rahmen des Forschungsprojekts *HomeSkin* (vgl. [160]) die Herstellung eines aerogelhaltigen Wärmedämmstoffs ökologisch bilanziert, das sich während der ökologischen Bilanzierung noch in einer sehr frühen Entwicklungsphase befand (vgl. auch [159], S. 531 ff.). Im Zuge der Ökobilanz sollte auch der zukünftig zu erwartende, industrielle Herstellungsprozess abgeschätzt werden. Daraus ergaben sich offene Fragestellungen, die zum Zeitpunkt der ökologischen Bilanzierung noch nicht beantwortbar waren. Dies betraf u. a. den Umgang mit dem anfallenden Koppelprodukt Ethanol. Szenario 1 beschreibt den geplanten Verkauf des im Zuge des Prozesses anfallenden Koppelprodukts. Es wurde mit einer Allokation (Erläuterung zu „Allokation“ vgl. auch Kapitel 5.3.5) auf Basis des ökonomischen Wertes abgebildet. Der Marktwert war zum Zeitpunkt der Ökobilanzierung weder für das Wärmedämmstoffprodukt noch für das Koppelprodukt bekannt. Durch Angaben des Herstellers und Recherchen anderer marktgängiger Produkte wurde davon ausgegangen, dass das Verhältnis zwischen Ethanol und dem Wärmedämmstoffprodukt *1:4* beträgt. In Szenario 3 ist das Verhältnis angenommen worden zu *1:8*, in Szenario 4 hingegen wurde es variiert zu *1:2*. Daneben stellte der Hersteller in Aussicht, dass das Ethanol intern wiederverwendet werden könnte. Dies ist in Szenario 2 abgebildet. Es ergeben sich so bspw. bei den Werten des *GWP* Abweichungen zwischen +6 % (Szenario 3) und -25 % (Szenario 2), bei den Werten des nicht erneuerbaren Primärenergiebedarfs *PENRT* betrugen diese zwischen +6 % (Szenario 3) und -56 % (Szenario 2).

Im Forschungsprojekt *Wall-Ace* (vgl. [162]) wurden innovative, aerogelbasierte Wärmedämmputze entwickelt. Im Zuge der Produktentwicklung wurden die Rezepturen vielfach angepasst und variiert. Diese Änderungen führten zu relevanten Abweichungen der Ergebnisse in der Wirkungsabschätzung. Für einen Quadratmeter Außenputz und eine Schichtdicke von 10 cm wurden bspw. zwischen 45 m% und 60 m% Bindemittelgemische, zwischen

10 m% und 35 m% mineralische Leichtzuschläge und zwischen 25 m% und 40 m% Aerogel-Granulat sowie zwischen 0,5 m% und 1,5 m% Zusatzstoffe verwendet. Für den *GWP* ergibt sich im Zuge des Herstellungsprozesses unter Berücksichtigung der verwendeten Ausgangsstoffe ein Intervall von ca. [65, 111] kg CO_2 äq. (vgl. Abbildung 6.3a), das *PENRT*-Intervall beträgt etwa [1208, 1590] MJ und das erneuerbare Primärenergiebedarf *PERT*-Intervall ungefähr [923, 1497] MJ (vgl. Abbildung 6.3b). Analoge Berechnungen wurden ebenfalls für einen im Projekt entwickelten, aerogelbasierten Innendämmputz durchgeführt; die Ergebnisse sind zusätzlich in Abbildung 6.3a dargestellt und beziehen sich auf eine 5 cm Dämmstoffdicke.



(a) :Ergebnisintervalle des *GWP* in kg CO_2 äq.

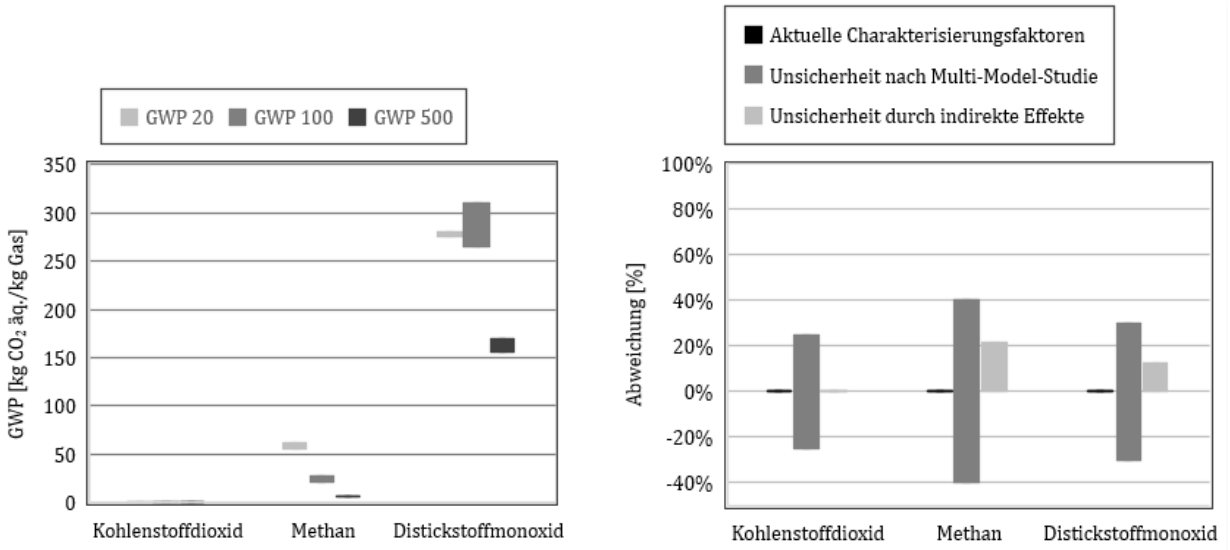
(b) : Ergebnisintervalle des *PENRT* und *PERT* in MJ.

Abbildung 6.3: Abweichung von Ökobilanzergebnissen im Zuge der Produktentwicklung (Bild angelehnt an [161], Fig. 5.)

Zum anderen stellen die realen, mitunter sehr komplexen Systeme in der Ökobilanzierung stark vereinfachte Abbildungen dar. Auswirkungen durch Änderungen der Systeme können nur begrenzt vorhergesagt werden, da im Modell keine Kenntnisse zu Wechselwirkungen und Abhängigkeiten der Kennwerte hinterlegt sind. So werden Änderungen von Charakterisierungsfaktoren zwar in zeitlichen Abständen aktualisiert, doch resultieren aus dieser Vorgehensweise Unsicherheiten, die ebenfalls berücksichtigt werden müssen. Dieser Umstand hat auch zur Folge, dass sich nicht nur Ergebnisse von Sachbilanzen im Laufe der Zeit durch modifizierte oder angepasste Prozesse ändern, sondern auch die Ergebnisse der Wirkungsabschätzung durch aktualisierte Kenntnisse zur Umweltwirkung von Stoffströmen. So betrug der *GWP*-Wert über 100 Jahre von Methan 1995 noch 21 kg CO_2 äq./kg Gas (vgl. [163], S. 22), während ihm 2013 der Wert 28 kg CO_2 äq./kg Gas zugeordnet wurde (vgl. [118], S. 731). Für Distickstoffmonoxid wurde der Wert von 310 kg CO_2 äq./kg Gas (vgl. [163], S. 22) auf 265 kg CO_2 äq./kg Gas angepasst (vgl. [118], S. 731).

In Abbildung 6.4a sind diese Änderungen grafisch für Kohlenstoffdioxid, Methan und Distickstoffmonoxid als Intervall dargestellt, wobei der *GWP* für einen Zeitraum von 20 Jahren, 100 Jahren sowie für einen Zeitraum von 500 Jahren berücksichtigt wird. Während der Charakterisierungsfaktor von Kohlenstoffdioxid als Referenzfluss des *GWP* konstant 1,0 kg CO_2 äq./kg Gas beträgt, weisen Methan und Distickstoffmonoxid Abweichungen auf. Die Verweildauer von Methan ist ca. 12 Jahre, die von Distickstoffmonoxid etwa 120 Jahre (vgl. [118], S. 714 und S. 731). Daher nehmen die Werte des *GWP* bei Methan und Distickstoffmonoxid mit zunehmender Betrachtungsdauer ab. Bei Distickstoffmonoxid ist dies aufgrund der hohen Verweilzeit von ca. 120 Jahren erst beim *GWP 500* zu erkennen. Gleichzeitig ist in Abbildung 6.4a zu erkennen, dass der *GWP 100* von Distickstoffmonoxid und Methan größere Abweichungen aufweist als der *GWP 20*. Dies ist auf zunehmende Unsicherheiten zurückzuführen, die mit längeren Betrachtungsdauern einhergehen. Die aktuellen Charakterisierungsfaktoren beinhalten allesamt

Unsicherheiten, die aus diesen selbst nicht transparent ablesbar sind. So sind bei den Werten des *GWP 100* für Kohlenstoffdioxid Abweichungen von ±25 % möglich (vgl. [164], S. 2802). Für Methan werden Abweichungen von ±40 % als möglich erachtet und für Distickstoffmonoxid werden Abweichungen um ±30 % angenommen (vgl. [118], S. 713).



(a) :Veränderungen der Charakterisierungsfaktoren zwischen 1995 und 2013.

(b) : Mögliche Abweichungen der aktuellen Charakterisierungsfaktoren.

Abbildung 6.4: Mögliche Abweichungen der *GWP*-Werte ausgewählter Gase.

Eine weitere Unsicherheit stellen indirekte Effekte bzw. Wechselwirkungen zwischen den Treibhausgasen dar. So beeinflusst bspw. der Ausstoß von Methan die Wirkung von Kohlenstoffdioxid. Die Berücksichtigung von Rückkopplungen in Zusammenhang mit Kohlenstoff resultiert in erhöhten Charakterisierungsfaktoren. Bei Methan wurde unter Berücksichtigung indirekter Einflüsse eine Erhöhung um ca. 21 % von 28 kg CO_2 äq./kg Gas auf 34 kg CO_2 äq./kg Gas ermittelt, bei Distickstoffmonoxid eine Erhöhung um ca. 12 % von 265 kg CO_2 äq./kg Gas auf 298 kg CO_2 äq./kg Gas (vgl. [118], S. 714, Table 8.7). In Abbildung 6.4b sind diese Abweichungen als Intervalle dargestellt. Die Abbildung soll verdeutlichen, in welchen (unsicheren) Bereichen die Ergebnisse von Ökobilanzen einzuordnen sind, die auf Basis dieser Charakterisierungsfaktoren ermittelt werden. Dabei ist anzumerken, dass auch zu rein technischen und etablierten Herstellungsprozessen oftmals nur wenige spezifische Sachbilanzen (vollumfänglich) verfügbar sind. Der Autor ist überzeugt davon, dass auch mit unbegrenzten zeitlichen und finanziellen Ressourcen Unsicherheiten zu vor- und nachgelagerten Prozessen nicht völlig beseitigt werden können; mitunter auch aufgrund der eingeschränkten Transparenz der Industrie wegen ihres berechtigten und nachvollziehbaren Bedürfnisses zum Schutz des geistigen Eigentums.

Ungeachtet dieses Aspekts erschwert die hohe Anzahl an erforderlichen Messwerten die Datenerhebung im Zuge der Sachbilanz. So sind in der Datenbank der Universität Leiden (vgl. [165]) aktuell 97 Elementarflüsse allein für den Wirkungskategorie-Indikator *GWP* gelistet (nach IPCC 2013, vgl. [118], S. 58 ff.). Lasvaux et al. (vgl. [166], S. 142 ff.) untersuchten, wie sich eine Reduktion der Elementarflüsse auf wenige relevante Emissionen auf die Ergebnisse von Ökobilanzen am Beispiel von Bauprodukten auswirken können. Dabei wurde eine vereinfachte Version des Indikators *GWP* entwickelt, der nur drei relevante Emissionen berücksichtigt: Kohlenstoffdioxid, Methan und Distickstoffmonoxid (vgl. [166], S. 144). Eine Analyse von 110 Datensätzen von Baustoffen der Herstellungsphase zeigte, dass der vereinfachte Indikator bei 105 von 110 Datensätzen nicht mehr als fünf Prozent vom ursprüng-

lichen *GWP*-Wert abweicht (vgl. [166], S. 146). Dies ist im Vergleich zu den vorherrschenden Unsicherheiten der Charakterisierungsfaktoren als gering einzustufen. Ausnahmen sind die Produktion von Aluminium sowie von Polystyrol, bei der diverse Verbindungen der Fluorkohlenwasserstoffe emittiert werden (vgl. [166], S. 146 f.). Diese sollen in Zukunft aber gänzlich verboten werden.

Die Studie lässt den Schluss zu, dass in der Ökobilanzierung weitere Vereinfachungen in der Datenerfassung und Modellierung denkbar sind. Sie sollten auf ihre Anwendbarkeit hin näher untersucht und zukünftig in Betracht gezogen werden.

6.2.2 Sachgerechte Klassifizierung von Unsicherheit

In dieser Arbeit wird Unsicherheit unterschieden in *Unpräzision, Ungenauigkeit* und *Ungewissheit*. Die Klassifizierung basiert auf der Analyse diverser Quellen (vgl. [167], S. 16-18, [168], S. 95-122, [169], S. 91 ff., [154], S. 15-21, [170], S. 3 und S. xvii f., [148], S. 1225 ff., [153], S. 5 ff. [152]), S. 892 ff.). Hierbei ist anzumerken, dass auch noch spezifischere Untergliederungen in der Literatur zu finden sind (vgl. Übersichten z. B. in [171], S. 161 ff., [172]). Diese werden für das in dieser Arbeit entwickelte Verfahren jedoch nicht als weiter relevant erachtet.

Die vorgenommene Klassifikation wird in Abbildung 6.5 visualisiert. Dargestellt ist ein zweidimensionales Pyramidenmodell, dessen Dimensionen „Ort“ und „Natur“ den Dimensionen von Walker et al. entsprechen (vgl. [153], S. 4 f.). Sie wurden für modellbasierte Entscheidungen formuliert. Die dritte Dimension zur Unsicherheit, die gemäß Bradley et al. (vgl. [148], S. 1231) und Walker et al. (vgl. [153], S. 4) die Größe von Unsicherheit beschreibt, ist nicht abgebildet. Sie kann Bradley et al. zufolge bei jeder Art von Unsicherheit groß sein (vgl. [148], S. 1232 f.). Der dreidimensionale Ansatz kann ein Erklärungsansatz für die unterschiedlichen Klassifikationen und Definitionen von Unsicherheit sein: So liegt bspw. der Schwerpunkt in der Messtechnik in den Bereichen „Modell“ und „Werte“ im Bereich der naturbedingten Unsicherheit, wobei die Größe der Unsicherheit (in Abbildung 6.5 nicht abgebildet) gering ist. Zusätzlich werden in Abbildung 6.5 die zur Modellierung notwendigen - und möglicherweise unsicheren - Informationen unterschieden in qualitative und quantitative Daten. Definitionen, die Unsicherheit als Differenz zum wahren Wert beschreiben (Definition von Unsicherheit vgl. Kapitel 6.2.6), beschränken sich vorwiegend auf quantitative Daten, die Entscheidungstheorie hingegen fokussiert Unsicherheit vorwiegend im Bereich qualitativer Daten.

Die unterschiedlichen Arten von Daten können sich überlagern bzw. voneinander abhängen - und damit auch deren enthaltene Unsicherheit. So fußt die Spitze der Pyramide, welche qualitative Daten repräsentiert (die z. B. zur Beantwortung folgender Fragestellung dienen: „Welche Produkte?“), auf dem Pyramidenboden quantitativer Daten (die z. B. zur Beantwortung folgender Fragestellung dienen: „erforderliche Menge der Produkte?“). Gleichsam ragt die Pyramidenspitze quantitativer Daten (die z. B. die Lebensdauer eines Produktes repräsentieren) wiederum in den Pyramidenboden (festgelegter) qualitativer Daten (die z. B. die Systemgrenze der Ökobilanz definieren). Daraus resultiert auch eine Überlagerung bzw. Abhängigkeit zwischen den Unsicherheiten: So ist bspw. die qualitative Wahl zwischen alternativen Produkten ungewiss (vgl. Kapitel 6.2.5); die Produkte werden jedoch durch quantifizierte Parameter modelliert, welche durch die Erfassung und Modellierung ungenau (vgl. Kapitel 6.2.4) und/ oder unpräzise (vgl. Kapitel 6.2.3) sein können.

Diese Abhängigkeit wird auch durch die Aussage von Tannert et al. (vgl. [152], S. 893 f.) untermauert. Demnach werden *objektive* Entscheidungen unter Unsicherheit - teilweise und sofern es möglich ist - mithilfe von logischen Regeln, Erfahrungen oder von probabilistischen Ansätzen abgeleitet bzw. die Unsicherheit kann (oftmals) Bradley et al. zufolge (vgl. [148], erster Absatz S. 10) darauf reduziert werden - wobei die Größe der Unsicherheit dabei steigt. Letzten Endes basieren diese Informationen auf quantifizierten oder quantifizierbaren Daten - jedoch gibt es nicht immer (ausreichend) Informationen, um darauf basierend eine Entscheidung treffen zu können und nicht

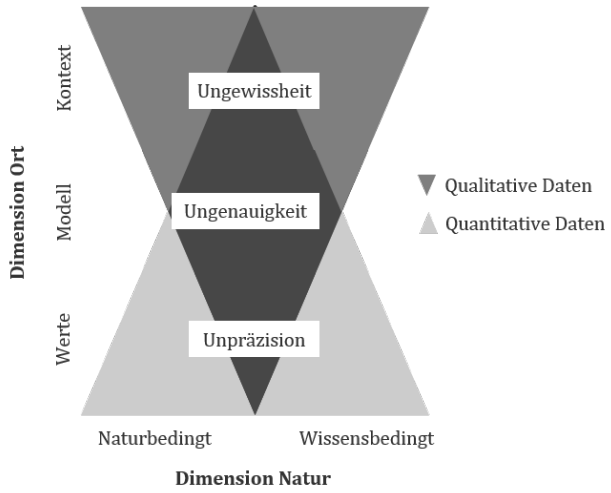


Abbildung 6.5: Pyramidenmodell: Typen von Unsicherheit entlang der Dimensionen „Natur“ und „Ort“.

.

immer basieren Entscheidungen rein auf dieser Art von Daten.

Die in dieser Arbeit als „Unpräzision“ bezeichnete Unsicherheit erstreckt sich über weite Teile über die Dimension „Natur“ und ist vorwiegend bei den „Werten“ der Dimension „Ort“ entlang des Unsicherheitsflusses vorzufinden (Erläuterung zu „Unsicherheitsfluss“ vgl. Kapitel 6.2.8). Ungewissheit ist ebenfalls in weiten Teilen der Dimension „Natur“ verortet, jedoch primär im Bereich des „Kontextes“ der Dimension „Ort“. Ungenauigkeit kommt hingegen hauptsächlich im Übergangsbereich wissens- und naturbedingter Unsicherheit der Dimension „Natur“ vor und ist vermehrt bei Unsicherheiten der Abbildung selbst und damit dem Bereich „Modell“ der Dimension „Ort“ angesiedelt. (Erläuterung zu „Unpräzision“, „Ungenauigkeit“ bzw. „Ungewissheit“ vgl. Kapitel 6.2.3, 6.2.4 bzw. 6.2.5)

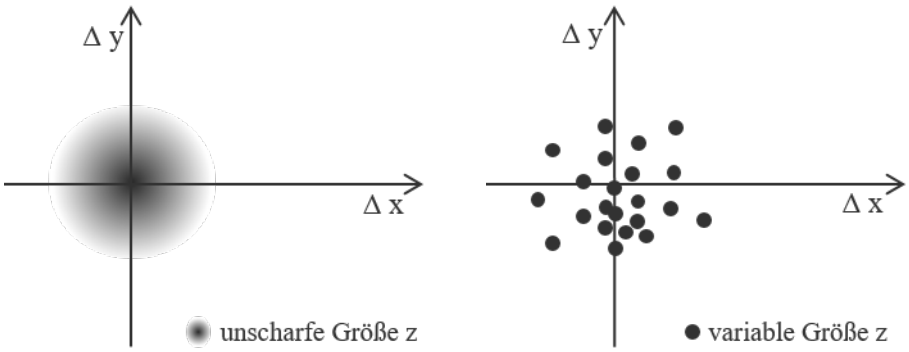
Die Ungewissheit nimmt allgemein bei qualitativen Fragestellungen mehr Raum ein als bei quantitativen Informationen (vgl. Schnittmenge des Flächenanteils vom Bereich „Ungewissheit“ sowie vom Bereich der „qualitativen Daten“ in Abbildung 6.5). Gleichzeitig ist zu erwarten, dass stattdessen bei quantitativen Daten die „Unpräzision“ (vgl. Schnittmenge des Flächenanteils von „Unpräzision“ sowie vom Bereich „quantitativer Daten“ in Abbildung 6.5) dominiert. Dies besagt jedoch nichts über den Schweregrad der Unsicherheit aus - die Größe der Unsicherheit als dritte Dimension ist in Abbildung 6.5 nicht dargestellt. Es ist davon auszugehen, dass eine unsichere Information in der Ökobilanzierung letztlich alle Arten von Unsicherheit implizieren kann und dies in der Praxis eher die Regel als den Ausnahmefall darstellt.

6.2.3 Unpräzise Daten durch Erfassung natürlicher Größen

In der Literatur (vgl. [167], S. 17, [154], S. 15, [170], S. 3, S. 20 ff. sowie S. xvii) wird eine Art natürlich variierende oder zufällige Unsicherheit beschrieben. Derlei Daten werden in dieser Arbeit als *unpräzise* bezeichnet angelehnt an Streiner et al. (vgl. [173], S. 327 ff., Anmerkung des Autors: in [173] wird zwischen „precision“ und „accuracy“ unterschieden, deren Antonyme hier als Unpräzision und Ungenauigkeit übersetzt werden).

Huijbregts (vgl. [154], S. 16) definiert eine „parameter uncertainty“, die aus Größen resultiert, welche aufgrund ihrer Natur nicht exakt erfasst werden (können) oder durch mangelnde sowie fehlende Informationen entstehen. Dies definiert Roš (vgl. [167], S. 17) als unscharfe und vage Informationen, begründet in der natürlichen Variabilität realer Daten, mangelnden Informationen oder modellbedingt unscharfen Parametern wie bspw. Charakteri-

sierungsfaktoren. Dinkel (vgl. [169], S. 92) ergänzt die Definition von Vagheit und Unschärfe dahingehend, dass ein „wahrer Wert“ existiert, dessen Ermittlung aber nicht möglich ist. Natürliche Variabilität, versehentliche Eingabefehler oder vergessene Eingaben bezeichnet Ciroth als „zufällige Fehler“ (vgl. [170], S. 21, Anmerkung des Autors: Fehler ist hier als Unsicherheit zu verstehen und widerspricht der in dieser Arbeit vorzufindenden Definition eines Fehlers - vgl. Kapitel 6.2.7), die zufällig vom „wahren Wert“ und nicht richtungsbezogen abweichen.



(a) : Unscharfes Verhalten in Richtung x, y. (b) : Natürliche Variabilität in Richtung x, y.

Abbildung 6.6: Unpräzise Werte aufgrund von Unschärfe (a) sowie Variabilität (b).

Unpräzise Daten aufgrund von Unschärfe (vgl. Abbildung 6.6a) bzw. Variabilität (vgl. Abbildung 6.6b) liegen vor, wenn die zu beschreibenden Größen z. B. eine der folgenden Eigenschaften aufweisen:

- sie sind kontinuierlich, d. h. die Übergänge zwischen zwei Zuständen sind fließend;
- sie sind nicht (direkt) messbare oder in der Zukunft liegende Größen und die Kalkulationen bzw. Prognosen lassen nur vage Abschätzungen zu;
- die zugrunde gelegten Informationen sind unvollständig oder fehlend, weshalb nur vage Vermutungen möglich sind;
- sie sind stochastischer Natur und unterliegen natürlichen Schwankungen, d. h. der Zustand, den sie annehmen bzw. annehmen werden, kann wissenschaftlich nicht exakt bestimmt werden.

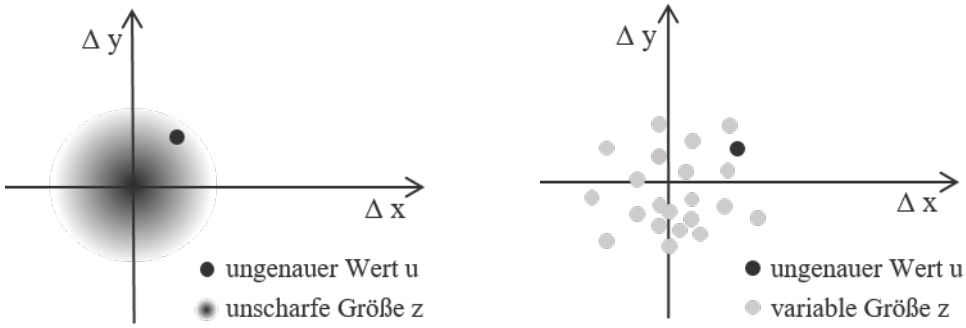
In dieser Arbeit wird *rein unpräzisen* Daten eine ausschließlich zufällige Abweichung zugeordnet. Auch für die geschätzten und prognostizierten Werte wird angenommen, dass sie nicht systematisch bspw. aufgrund eines nicht berücksichtigten Aspekts unter- oder überschätzt worden sind, obgleich es natürlich sein kann, dass alle Erfahrungswerte bzw. alle prognostizierten Werte unter oder über dem wahren Wert liegen, welche dann aber gemäß dieser Definition auch einen ungenauen Anteil aufweisen (Erläuterung zu „ungenauen Daten“ vgl. Kapitel 6.2.4). Die Beschreibungen definieren eine Unsicherheit, die natürlich und objektiv bedingt sein kann, womit diese sich über die gesamte Dimension „Natur“ erstreckt (Dimension „Natur“ vgl. Abbildung 6.5). Dabei liegt der Schwerpunkt auf Werten entlang des Unsicherheitsflusses (Definition von „Unsicherheitsfluss“ vgl. Kapitel 6.2.8), was einem Teilbereich der Dimension „Ort“ entspricht (Dimension „Ort“ vgl. Abbildung 6.5).

6.2.4 Ungenaue Daten durch Vereinfachung

Daneben wird in der Literatur (vgl. [169], S. 92, [154], S. 17, [170], S. 21) eine weitere Art von Unsicherheit beschrieben, die vorwiegend durch die Notwendigkeit einer Vereinfachung hervorgerufen wird. Im Grundsatz ent-

spricht dies der Modellbildung als vereinfachte Abbildungen realer Systeme, weshalb sich diese Art von Unsicherheit im Modell manifestiert. Durch die hierfür notwendigen Bestimmungen ist eine systematische Abweichung vom „wahren Wert“ zu erwarten, die bestenfalls null sein kann. Daher werden derlei Daten angelehnt an Streiner et al. (vgl. [173], S. 327) als *ungenau* bezeichnet (Anmerkung des Autors: In [173] wird zwischen „precision“ und „accuracy“ unterschieden, deren Antonyme hier als „Unpräzision“ und „Ungenauigkeit“ übersetzt werden).

Huijbregts (vgl. [154], S. 17) beschreibt „model uncertainty“ als eine Unsicherheit, die aus Vereinfachungen resultiert, welche bei der Abbildung realer Systeme getroffen werden (müssen). Dies hat zur Folge, dass bestimmte Eigenschaften und Merkmale realer Systeme oder Beziehungen von Systemen nicht (vollständig) berücksichtigt werden (können). Beispiele hierfür sind die Annahme der Linearität in der Ökobilanzierung sowie die Nichtberücksichtigung zeitlicher und lokaler Aspekte. Hinzu kommt, dass z. B. die Charakterisierungsfaktoren für die Abschätzung des Treibhauspotentials in die ökologische Bilanzierung einfließen, die durch andere Modelle (z. B. komplexe Klimamodelle) kalkuliert wurden. Dies trägt Unsicherheiten anderer Modelle in die Ökobilanzierung ein.



(a) : Ungenauer Wert u zur Beschreibung der unscharfen Größe z.

(b) : Ungenauer Wert u zur Beschreibung der variablen Größe z.

Abbildung 6.7: Ungenauer Wert u zur Beschreibung einer unscharfen (a) sowie einer variablen (b) Größe z.

Dinkel (vgl. [169], S. 92) definiert „Ungenauigkeit“ als unsichere Informationen, für die ein „wahrer Wert“ existiert, deren Erfassung jedoch (zu) aufwändig ist, sodass sie bspw. geschätzt werden. Als weiteres Beispiel für diese Art von Unsicherheit führt Dinkel Messfehler auf (vgl. [169], S. 92). Ciroth ordnet Messfehler den systematischen Fehlern zu (vgl. [170], S. 21, Anmerkung des Autors: Fehler ist hier als Unsicherheit zu verstehen und widerspricht der in dieser Arbeit vorzufindenden Definition eines Fehlers - vgl. Kapitel 6.2.7). Roš beschreibt ebenfalls eine weitere Art von Unsicherheit, die er als Abweichung vom „wahren Wert“ definiert und er unter anderem neben „Unpräzision“ und „Unexaktheit“ als „Ungenauigkeit“ bezeichnet (vgl. [167], S. 17). Als Ursachen werden Variabilität oder Unschärfe sowie die Notwendigkeit von Annahmen aufgeführt. Auch wenn sich Ungenauigkeit selbst vorwiegend im Modell äußert, können die Ursachen der Ungenauigkeit vielfältig sein. In der Ökobilanzierung resultiert *reine* Ungenauigkeit z. B. aus:

- gemessenen Werten mit Abweichungen, die der Technik bzw. dem Messvorgang, nicht aber der zu messenden Größe zuzuordnen sind;
- Abweichungen in den Bilanzierungsergebnissen, die den verwendeten Berechnungsalgorithmen zuzuordnen sind;
- der Verwendung generischer Datensätze, da nicht genügend spezifische Daten zu vor- oder nachgelagerten Prozessen verfügbar sind;

- Subjektivität, Fokus oder Erwartungshaltung des Ökobilanzierenden, was dessen Entscheidungen - unbewusst - systematisch prägen kann.

Ungenaue Daten können auch entstehen, wenn unpräzise oder ungewisse Größen nicht vollständig erfasst werden (können). Ungenauigkeiten unpräziser Größen sind exemplarisch in Abbildung 6.7a und Abbildung 6.7b dargestellt. Wird bspw. die Unschärfe bzw. Variabilität einer unpräzisen Größe nicht berücksichtigt oder erkannt und diese in Form bspw. eines diskreten Werts beschrieben, kann dies zu Ungenauigkeit führen. Dies ist bspw. der Fall, wenn:

- einer kontinuierlichen Größe bspw. das arithmetische Mittel als diskreter Wert zugewiesen wird;
- einer stochastischen Größe bspw. ein diskreter Wert zugewiesen wird, den sie laut angenommener Wahrscheinlichkeitsverteilung in mindestens 95 Prozent aller Fälle annehmen wird;
- ein fehlender Wert geschätzt bzw. angenommen wird oder ein generischer Datensatz aus mehreren verfügbaren Alternativen zur Anwendung kommt (der Entscheidungsprozess entspricht einer Ungewissheit, die Folge davon, d. h. die Entscheidung zur Verwendung eines der generischen Datensätze entspricht einer Ungenauigkeit);
- einer zukünftigen Größe ein diskreter Prognose- oder Schätzwert zugewiesen wird.

Ungenaue Daten können aus natürlicher und/oder subjektiver Unsicherheit resultieren, da diese jedoch vereinfacht nur einen Teilbereich erfassen (können), beschreiben sie auch nur einen Teilbereich der Dimension „Natur“ (Dimension „Natur“ vgl. Abbildung 6.5). Analog verhält sich Ungenauigkeit bei der Dimension „Ort“ (Dimension „Ort“ vgl. Abbildung 6.5). Während die Ursachen sich über die gesamte Dimension „Ort“ erstrecken können, manifestiert sich die Ungenauigkeit hauptsächlich im Modell, d. h. dem lokalen Übergangsbereich der Dimension „Ort“, der die Parameter und den Kontext verknüpft. Ungenaue Größen werden oftmals auch als „fehlerhaft“, „falsch“ und „inkorrekt“ bezeichnet, doch ist hier konsequent zu differenzieren zwischen einem groben, fahrlässigen Fehler und einer „Unsicherheit“ (Definition von „Fehler“ vgl. auch Kapitel 6.2.7).

6.2.5 Ungewisse Daten durch vorliegende Alternativen

In der Literatur wird eine weitere Art von Unsicherheit definiert, welche auf Entscheidungsmöglichkeiten zurückzuführen ist (vgl. [154], S. 18, [169], S. 91 f., [168], S. 97 f.). Diese Art von Unsicherheit wird in dieser Arbeit angelehnt an Pohl als *ungewiss* bezeichnet (vgl. [168], S. 98, Anmerkung des Autors: Pohl unterscheidet zwischen objektbezogener Ungewissheit und subjektbezogener Unsicherheit; diese Unterscheidung erfolgt in dieser Arbeit nicht). Huijbregts erläutert eine „Unsicherheit durch Wahlmöglichkeiten“ (vgl. [154], S. 18, Anmerkung des Autors: diese Bezeichnung wurde vom Autor aus dem Englischen ins Deutsche übersetzt). Ungewissheit kann z. B. bei der Wahl der Allokationsmethode, der funktionellen Einheit oder der Methode zur Wirkungsabschätzung auftreten. Dinkel definiert diese Art von Unsicherheit dadurch, dass kein „wahrer Wert“ existiert (vgl. [169], S. 91 f.). Bradley et al. bezeichnen mögliche Zustände, denen keine Wahrscheinlichkeit zugeordnet werden kann, als schwere empirisch-faktische Zustandsunsicherheit bzw. Ambiguität (vgl. [148], S. 8 f.). Pohl definiert diese Alternativen „als Bereich des gleichberechtigt Möglichen“ (vgl. [168], S. 98), womit sie alle gleichsam „wahr“ sind.

Ungewissheit im Rahmen einer Ökobilanz kann auftreten in Form von „datenbasierter Ungewissheit“, „modellbasierter Ungewissheit“ und „Ungewissheit konzeptueller Natur“ (vgl. Abbildung 6.8):

- „Datenbasierte Ungewissheit“ resultiert aus Entscheidungen, die innerhalb des Bilanzierungsrahmens in Bezug auf Daten getroffen werden und daher die Ergebnisse beeinflussen können. Gibt es für einen Input

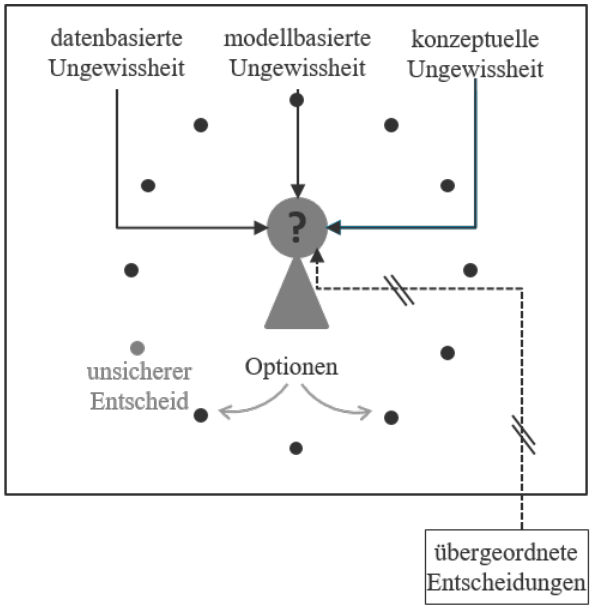


Abbildung 6.8: Datenbasierte, modellbasierte und konzeptuelle Ungewissheit.

.

einen Wertebereich und keinen exakten Wert, muss bspw. in der herkömmlichen Bilanzierung entschieden werden, wie der Input definiert werden soll.

- „Modellbasierte Ungewissheit“ kann bei Entscheidungen auftreten, die innerhalb des Bilanzierungsrahmens in Bezug auf das Modell getroffen werden (müssen) und daher die Ergebnisse beeinflussen können. Liegen bspw. für ein Produkt, dessen spezifische Modellierung nicht möglich ist, mehrere gleichsam geeignete, generische Datensätze vor, ist in der herkömmlichen Ökobilanzierung in der Regel ein Datensatz zu wählen.

- „Ungewissheit konzeptueller Natur“ sind auf Entscheidungen zurückzuführen, die den Bilanzierungsrahmen der durchzuführenden Ökobilanz betreffen und daher die Ergebnisse beeinflussen können. Werden zusätzliche Lebenszyklusphasen berücksichtigt, oder die Abschneidekriterien verschärft, werden „Entscheidungen konzeptueller Natur“ getroffen. Dies ist ein Vorgang, der durch den iterativen Ansatz einer Ökobilanzierung notwendig sein kann.

„Entscheidungen übergeordneter Struktur“ (vgl. auch Abbildung 6.8) sind hingegen Entscheidungen, die außerhalb einer Ökobilanz im Zusammenhang oder auf Basis dieser getroffen werden und welche nicht die Durchführung der Ökobilanz selbst oder dessen Ergebnisse beeinflussen (sollten). Sie beeinträchtigen demzufolge auch nicht die Robustheit der in einer Ökobilanz erzeugten Ergebnisse; sie können jedoch dazu führen, dass die Ergebnisse vermehrt angezweifelt, unter Verschluss gehalten oder - im Gegenteil - vielfach zitiert und öffentlich präsentiert werden.

Ungewissheit impliziert sowohl die objekt- als auch die subjektbezogene Unsicherheit, womit sie den gesamten Bereich der Dimension „Natur“ umfasst (Dimension „Natur“ vgl. Abbildung 6.5). Sie tritt vorwiegend im Kontext auf, dem Teilbereich der Dimension „Ort“ (Dimension „Ort“ vgl. Abbildung 6.5), bei dem qualitative Fragestellungen vermehrt auftreten.

6.2.6 Definition von Unsicherheit

Es existieren verschiedene Definitionen zur Unsicherheit in der Literatur im Kontext der Ökobilanzierung. So definieren Finnveden et al. Unsicherheit bspw. als „Abweichung zwischen der gemessenen oder berechneten Größe und dem tatsächlichen Wert dieser Größe“ (vgl. [174], S. 14, Anmerkung des Autors: die Definition wurde vom Autor aus dem Englischen ins Deutsche übersetzt). Diese Definition ist jedoch nicht auf alle in dieser Arbeit beschriebenen Arten von Unsicherheit anwendbar. So verfügen ungewisse Daten zum einen nicht über einen wahren Wert. Zum anderen sind die „Abweichungen“ unscharfer Größen nicht exakt bestimmbar. So kann einer unscharfen Größe kein diskreter Wert zugewiesen werden, sondern allenfalls ein Wertebereich - z. B. erfasst als Intervall. Ein Intervall kann aber mit der real zu beschreibenden Größe gemeinsame Schnittmengen aufweisen oder diese gänzlich umhüllen (vgl. hierzu auch Vergleichsoperatoren von Intervallen in Kapitel 3.4), wodurch die Bestimmung einer Diskrepanz - wie in obiger Formulierung beschrieben - erschwert ist.

Eine weitere Definition finden bspw. Walker et al. als „irgendeine Abweichung vom unerreichbaren Ideal einer vollständig deterministischen Kenntnis des betreffenden Systems“ (vgl. [153], S. 7, Anmerkung des Autors: die Definition wurde vom Autor aus dem Englischen ins Deutsche übersetzt). Diese Definition wiederum ist nonkonform mit der in dieser Arbeit zugrunde gelegten Ansicht, dass vollständiger Determinismus nicht unweigerlich als das wissenschaftliche Ideal angesehen werden sollte bzw. kann, sondern mitunter vage Modellierungen mit Berücksichtigung von Unsicherheiten robustere Ergebnisse liefern können (Informationen zur allgemeinen Wertung von Unsicherheit in der Wissenschaft vgl. auch Kapitel 6.1.1).

Im Rahmen dieser Dissertation und im Hinblick des entwickelten Verfahrens zur umfassenden Berücksichtigung von Unsicherheit wird Unsicherheit wie folgt definiert: „Unsicherheit ist die Gesamtheit an Unbestimmtheit, die aus der Verwendung von all denjenigen Informationen resultiert, deren zugrunde gelegten Parameter nicht exakt bestimmt werden können. Je größer der Bereich ist, der die möglichen und relevanten Werte umfasst, die ein Parameter annehmen kann, desto unsicherer ist er.“

6.2.7 Abgrenzung der Unsicherheit von Fehlern

Der Begriff „Unsicherheit“ ist klar abzugrenzen von einem „Fehler“. Fehler werden in dieser Arbeit als Abweichungen bezeichnet, die irrtümlich durch fehlerhaftes Verhalten verursacht werden, wie z. B. durch das Eintippen eines falschen Inputwertes in das Bilanzierungsprogramm. Derlei Fehler können durch Sorgfältigkeit und Achtsamkeit vermieden werden. Unsicherheiten existieren trotz Einhaltung aller Regeln und korrekter Durchführung. Besteht eine systematisch bedingte Ungenauigkeit trotz sorgfältigen Verhaltens, wiederholter Prüfung und guter wissenschaftlicher Praxis und wird nicht erkannt, stellt sie dann - und nur dann - keinen „Fehler“, sondern eine „Unsicherheit“ dar.

6.2.8 Unsicherheiten nach Ort des Auftretens

Neben der Art kann Unsicherheit auch nach dessen Entstehungsort im Zuge der ökologischen Bilanzierung unterschieden werden. Heijungs et al. (vgl. [172]) differenzieren zwischen Unsicherheit in den Eingabedaten, Unsicherheit im Zuge der Berechnung und Unsicherheit in den Ausgabedaten. Gelangen unsichere Werte in die Bilanzierung, „fließen“ auch die enthaltenen „Unsicherheiten“ bis zu den Ergebnissen, wobei sie sich – in Wechselwirkung mit weiteren Unsicherheiten - vergrößern oder aber verringern können. In dieser Arbeit wird dies als „Unsicherheitsfluss“ bezeichnet (Anmerkung des Verfassers: In der Vergangenheit wurden die Auswirkungen von Unsicherheiten der Eingabedaten auf die Endergebnisse allgemein als „Fehlerfortpflanzung“ bezeichnet (vgl. z. B. [170], S. 3). Da mittlerweile in der Messtechnik derlei Abweichungen (vgl. z. B. Grundbegriffe in [124], S. 14)

klar von groben Fehlern abgegrenzt werden, wird der Begriff der „Fehlerfortpflanzung“ als obsolet angesehen und daher in dieser Arbeit nicht verwendet). In der Intervallarithmetik ist der Unsicherheitsfluss aufgrund abhängiger Intervalle im Zuge der Berechnung (Informationen zu „abhängigen Intervallen“ vgl. Kapitel 3.6.2) von Bedeutung.

6.3 Verfahren zur Analyse von Unsicherheit

Bis heute fehlt ein zuverlässiges, standardisiertes Verfahren zur Behandlung aller Arten von Unsicherheit in der Ökobilanzierung (vgl. z. B. [175], S. 2). Es gibt jedoch eine Reihe von Verfahren und Ansätzen, wie je nach Art der zugrunde gelegten Unsicherheit - auch über den Kontext der Ökobilanzierung hinaus - mit dieser umgegangen werden kann. Bei Unpräzision aufgrund unscharfer Größen wird allgemein die Fuzzylogik empfohlen (vgl. z. B. [176], S. 244-249, [177], S. 251-267, [178], S. 1-5), für unpräzise Daten aufgrund variabler Größen hingegen stochastische Modelle (vgl. z. B. [178], S. 1-5, [179], S. 57 ff.). Ungewisse Daten aufgrund (gleichsam möglicher) Alternativen werden mittels entscheidungstheoretischer Konzepte angegangen (vgl. z. B. [180], S. 31-35, [152], S. 892 ff.). Ungenaue Daten können ebenfalls zu relevanten Abweichungen und damit eingetragenen Unsicherheiten in den Ergebnissen führen. In der Regel sind diese jedoch nicht bekannt und müssen daher abgeschätzt werden (vgl. z. B. [181], S. 51 f.). Um den Einfluss der Ungenauigkeit auf die Endergebnisse zu ermitteln, können bei vermuteter geringer Ungenauigkeit analytische Formeln zur Berechnung verwendet werden (vgl. [170], S. 5 f.). Andere eingetragene Ungenauigkeiten komplexerer Art, wie sie bspw. durch die Verwendung repräsentativer Datensätze oder aufgrund getroffener Annahmen verursacht werden (können), sind durch eine erneute Analyse und vertiefte Recherche überprüfbar (vgl. [182], S. 92).

In der Praxis der Ökobilanzierung angewandte Methoden und in den geläufigen Programmen verfügbare Verfahren sind bspw. statistische Methoden wie die Monte-Carlo-Simulation (Monte-Carlo-Simulation vgl. auch Kapitel 6.3.1) für unpräzise Daten sowie Szenarioanalysen (Szenarioanalysen vgl. Kapitel 6.3.2) für ungewisse Daten (vgl. z. B. [96], S. 37 f.). Je nach Art und Schwerpunkt der Unsicherheit wurden weitere Ansätze zum Umgang mit Unsicherheit in der Ökobilanzierung erforscht. Dies sind u. a. die Anwendung von Näherungsformeln bei systematischen und zufälligen Abweichungen (Verfahren auf Basis analytischer Formeln vgl. Kapitel 6.3.3) sowie die Wahrscheinlichkeitstheorie für zukünftige Ereignisse (Verfahren auf Basis der Wahrscheinlichkeitstheorie vgl. Kapitel 6.3.4). Des Weiteren wird die Formulierung *vager* Werte durch Anwendung der Fuzzy-Set-Theorie vorgeschlagen (Verfahren auf Basis der Fuzzy-Set-Theorie vgl. Kapitel 6.3.5).

6.3.1 Monte-Carlo-Simulation

Die Monte-Carlo-Simulation ist ein stochastisches Verfahren, mit dem zufällige Abweichungen der Inputs bestimmt werden und mit diesen die Ökobilanzierung vielfach berechnet wird (vgl. auch [183], S. 40 ff.). Hierfür sind für die Inputs Annahmen über die Verteilung festzulegen. Die Festlegung einer geeigneten Verteilungsfunktion obliegt bei herkömmlichen Ökobilanzierungsprogrammen in der Regel dem Ökobilanzierenden selbst (vgl. [183], S. 3). Die Monte-Carlo-Simulation gilt nach [170] ungeachtet der Größe der Unsicherheit und deren Art - in [170] differenziert nach „systematischen Fehler“ und „zufälligem Fehler“ - als geeignet (vgl. [170], S. 127). Die Methode stößt als stochastisches Verfahren jedoch an ihre Grenzen, wenn bspw. die Unsicherheit aus Ungewissheit (vgl. Kapitel 6.2.5) oder Unschärfe (vgl. Kapitel 6.2.3) resultiert, d. h. wenn Wahrscheinlichkeitsverteilungen nicht bestimmt werden können oder unbekannt sind.

6.3.2 (Vollständige) Szenarioanalyse

Gibt es in der ökologischen Bilanzierung Ungewissheit über die Entscheidung zwischen mehreren Alternativmöglichkeiten, können Szenarien erfasst und berechnet werden. Szenarien können zwischen deskriptiven und normativen Szenarien unterschieden werden (vgl. [184], S. 144). Während deskriptive Szenarien z. B. die möglichen zukünftigen Zustände ausgehend vom jetzigen Zustand zu prognostizieren versuchen, werden normative Szenarien modelliert, um ein festgesetztes Ziel zu entwickeln. Beide Arten von Szenarien finden sich in der Ökobilanzierung wieder. So können zukünftige Änderungen eines Produksystems ausgehend vom aktuellen, als Referenz dienenden Produktsystem modelliert und bilanziert werden (Modellierungsansätze vgl. auch Kapitel 4.3.2). Daneben werden mithilfe von Ökobilanzierungen auch Analysen für definierte, ökologisch orientierte Zielvorgaben durchgeführt. So kann beispielsweise durch die Modellierung verschiedener Produktsysteme die Frage beantwortet werden, mit welchen Produkten das Treibhauspotential eines Neubaus um bspw. 30 Prozent im Vergleich zu einer herkömmlichen Bauweise gesenkt werden kann. Szenarioanalysen werden häufig bei der Entscheidungsfindung angewandt (vgl. [184], S. 255).

Es ist auch möglich, vollständige Szenarioanalysen durchzuführen, dies ist mitunter jedoch zu aufwendig. Dabei werden durch Variation alle möglichen Kombinationen als Szenarien bilanziert werden. Für m Alternativmöglichkeiten an n verschiedenen lokalen Positionen ergeben sich n^m Variationen, weswegen dies in vielen Fällen nur für einzelne Optionen durchgeführt wird.

6.3.3 Analytische Formeln zur Fehlerrechnung

Ciroth entwickelte ein Verfahren zur quantitativen Berechnung von „Fehlern“ mittels analytischer Formeln (vgl. [170], Anmerkung des Verfassers: der Begriff „Fehler„ist hier als eine zufällige bzw. systematische Abweichung zu verstehen und nicht mit der Definition eines Fehlers aus Kapitel 6.2.7 gleichzusetzen). Hierbei wurde zwischen zufälligen und systematischen „Fehlern“ differenziert. Zusätzlich wurde berücksichtigt, an welchen einzelnen Berechnungsschritten die „Fehler“ auftreten, wobei die Berechnungsschritte als „Systemelemente“ bezeichnet werden (vgl. [170], S. 17 ff.). Die Analyse erfolgte mit einer sog. „virtuellen LCA“, die auf ca. 500 „Massenströmen“ und ca. 60 „Prozessen“ basiert (vgl. [170], S. 23 ff.). Die Massen und die „Fehler“ der Massenströme wurden zufällig erzeugt. Insgesamt wurden folgende Methoden aus der Taylorreihenentwicklung erster und zweiter Ordnung für zufällige als auch systematische Fehler untersucht:

- die einfache systematische Fehlerrechnung für systematische Fehler
- das Gaußsche Fehlerfortpflanzungsgesetz für zufällige Fehler
- die allgemeine Taylorformel zweiter Ordnung für systematische Fehler
- die Fehlerrechnung nach Bader/Baccini für zufällige Fehler

Als Referenz wurde zum einen die Formel zur Berechnung des jeweiligen tatsächlichen systematischen Fehlers verwendet, zum anderen die Monte-Carlo-Simulation (Monte-Carlo-Simulation vgl. Kapitel 6.3.1). Die Standardabweichung des Ergebniswerts diente zur Abschätzung der Standardabweichung des zufälligen Fehlers. Die Anwendung der Taylorformeln erster Ordnung setzen Annahmen voraus. Eine Bedingung ist, dass die Fehler gegenüber dem wahren Wert gering sind und eine weitere, dass die Parameter voneinander unabhängig sind (vgl. [170], S. 5 f.). Für große Abweichungen können die Taylorformeln zweiter Ordnung angewandt werden.

Das Verfahren sieht vor, zunächst die angenommenen oder identifizierten systematischen Fehler in der Berechnung zu eliminieren und im Anschluss die Größe der zufälligen Fehler zu berechnen (vgl. [170], S. 126). Je

nachdem, in welcher Phase der Ökobilanz die Fehler ermittelt werden sollen und wie groß die relativen Fehler sind, werden unterschiedliche Methoden empfohlen. Demnach kann bspw. das Gaußsche Fehlerfortpflanzungsgesetz bei der Ermittlung der Mengenmultiplikatoren (Anmerkung des Autors: Skalierungsfaktoren werden hier als „Mengenmultiplikatoren“ bezeichnet (vgl. in [170], S. 14)) zuverlässig verwendet werden, wenn die Abweichungen geringer als 20 Prozent sind; die relativen Fehler aus Charakterisierungsfaktoren und Sachbilanz sollen in der Wirkungsabschätzung geringer als 40 Prozent sein. Die Fehlerrechnung nach Bader/Baccini gilt für Sachbilanz sowie Wirkungsabschätzung noch bis zu einer Abweichung unter 100 Prozent als geeignet. Die Ermittlung über Monte-Carlo-Simulation gilt nach Ciroth (vgl. [170], S. 127) ungeachtet der einzelnen Berechnungsschritte immer als geeignet, kann jedoch sehr aufwendig werden.

Das Verfahren wurde an einer linearen Produktsystemstruktur analysiert. Baum- und schleifenartige Produktsysteme können nach Ciroth (vgl. auch [170], S. 119 ff.) durch redundante Prozessmodule in Form einzelner linearer Prozessketten dargestellt werden. Dies führt jedoch zu Korrelationen, welche Auswirkungen auf die Anwendbarkeit des Verfahrens haben können (vgl. [170], S. 115 und S. 119-125).

6.3.4 Wahrscheinlichkeitstheorie

Gantner stellte ein auf Wahrscheinlichkeiten basierendes Verfahren zur ökologischen Bewertung vor (vgl. [185]). Mit dem Verfahren können relevante, zukünftige Entscheidungen und Ereignisse wahrscheinlichkeitsbasiert berücksichtigt werden. Das Verfahren ist so konzipiert, dass es direkt ohne Anpassungen bestehender Ökobilanzierungsprogrammen angewandt werden kann. Es basiert im Wesentlichen darauf, potentielle, zukünftige Entscheidungen und Ereignisse möglichst umfassend abzuleiten und auf so wenige wie mögliche, sog. „Entscheidungsstufen“ zu reduzieren (vgl. [185], S. 56 ff.). Diesen werden neben Eintrittswahrscheinlichkeiten zu bestimmten Zeitpunkten auch sog. „Fortschreibungsfaktoren“ (vgl. [185], S. 50 ff.) zur Berücksichtigung von weniger bedeutsamen Einflussfaktoren zugeschrieben. Die Wahrscheinlichkeiten werden über Erfahrungswerte von Experten sowie mithilfe von Statistiken, Studien oder Annahmen definiert. Die Einflussfaktoren können auf Basis verfügbarer Studien, Szenarioanalysen und Ökobilanzen sowie Erfahrungswerten von Experten oder Brainstormings ermittelt werden.

Die Faktoren sollen anschließend bezüglich ihrer Wirkung auf andere Faktoren mit den Werten von 0 (keine Wirkung) bis 3 (große Wirkung) abgeschätzt werden (vgl. [185], S. 61). Weist ein Faktor eine hohe Summe dieser so quantifizierten Wirkungen auf andere Faktoren auf, die sog. „Aktivsumme“ eines Faktors, wird er als wichtig eingestuft. Wichtige Einflussfaktoren werden dynamisch erfasst (vgl. [185], S. 63 ff.). Weniger wichtige Einflussfaktoren werden in Form eines Fortschreibungsfaktors berücksichtigt, welche sich aus dem Produkt der jeweiligen Korrekturfaktoren der einzelnen Einflussfaktoren ergeben.

6.3.5 Fuzzy-Set-Theorie

Roš und Pohl setzten sich mit einem Verfahren zur Unsicherheitsanalyse in Ökobilanzen auf Basis der Fuzzylogik auseinander (vgl. [167] und [168]). Während Pohl (vgl. [168], S. 95 ff.) den Schwerpunkt auf ungewisse Informationen legt, untersucht Roš die ungenauen und unpräzisen Informationen (vgl. [167], S. 53 ff.). Letzterer thematisiert dabei auch, dass die Unsicherheiten von Äquivalenzfaktoren bedeutsam sein können (Anmerkung des Autors: „Charakterisierungsfaktoren“ werden in [167] als „Äquivalenzfaktoren“ bezeichnet).

Bei Anwendung der Fuzzylogik werden Elemente nicht mehr eindeutig zugeordnet, sondern mithilfe von sogenannten Zugehörigkeitsfunktionen der Grad der Zugehörigkeit zu unterschiedlichen Mengen bestimmt (vgl. auch [167], S. 60 f.). Ist bspw. nicht eindeutig nachweisbar, welchen Einfluss eine Emission e auf den Treibhauseffekt

hat, kann sie über eine Zugehörigkeitsfunktion diversen Charakterisierungsfaktor-Mengen zugewiesen werden. Roš verwendete Fuzzy-Trapeze (vgl. auch [167], S. 65 ff.), für welche zwei Intervalle definiert werden. Zur Definition der sog. „Stützmenge“ werden die „pessimistischen“ Grenzwerte bestimmt, welche die Grenze zum Unmöglichen darstellen. Dies bedeutet, dass im obigen Beispiel der minimale sowie maximale Einfluss der Emission *e* auf den Treibhauseffekt definiert wird. Die sog. Eckwerte bilden dagegen als „optimistischer“ Wertebereich den sog. „Kern“ des fuzzy-Trapezes mit „wirklichen“, „erwarteten“ bzw. „plausiblen“ Werten (vgl. [167], S. 70). Der Grad der Zugehörigkeit zum Kern bzw. zur Stützmenge bleibt eine Schätzung, die „aus Gründen der Praktikabilität [...] linear interpoliert wird“ ([167], S. 66). Die Berechnung der beiden Intervalle erfolgt gesondert voneinander (vgl. [167], S. 75).

6.4 Problematik und Anwendungsgrenzen

In diesem Kapitel werden die grundlegenden Schwierigkeiten erläutert, die sich in Ökobilanzen bei der Erfassung und Analyse von Unsicherheiten ergeben. Dies betrifft zum einen die Vielzahl qualitativ differenzierter Unsicherheiten und der daraus abgeleiteten unterschiedlichen Methoden (vgl. Kapitel 6.4.1), zum anderen ist es das Fehlen einer sicheren Seite durch den relativen Ansatz der Methodik der Ökobilanzierung selbst, was die Erfassung von Unsicherheit erschwert (vgl. Kapitel 6.4.2).

6.4.1 Diskontinuität und Heterogenität unsicherer Daten

Die bestehenden mathematischen Methoden zur Fehlerberechnung sind nicht für jede Art von Unsicherheit gleichermaßen geeignet. Liegen sowohl unpräzise, ungenaue als auch ungewisse Daten vor, ist eine spezifische Behandlung mit den jeweils geeigneten Methoden zwar möglich, sie können jedoch nicht ohne weiteres miteinander „verrechnet“ werden (vgl. Abbildung 6.9). Oftmals ist eine Kombination verschiedener Methoden zur Fehlerabschätzung nicht möglich (vgl. z. B. [186], S. 381 f. sowie [161], S. 369 f.). Es gibt jedoch Ansätze, die mehrere Theorien vereinen. So lassen sich bspw. unscharfe Daten als Fuzzy-Zahlen erfassen und eine dazu vorliegende Variabilität stochastisch bestimmen - beide Theorien sind kombinierbar (vgl. [176], S. 244 ff.). Hierbei ist jedoch anzumerken, dass aus derartigen Methoden nur schwer begreifbare Ergebnisse resultieren wie z. B. Fuzzy-Erwartungswerte. Solche Methoden sind daher für die Ökobilanzierung als weniger praktikabel einzustufen. Begründet ist dies auch dadurch, dass Ökobilanzen zunehmend veröffentlicht werden und die Bilanzierungsergebnisse so einem breiten Publikum - auch außerhalb des Expertenkreises - zur Verfügung gestellt werden. Diese schwer erfassbaren Ergebnisse derart aufzubereiten, dass diese nicht missinterpretiert werden, wird vom Autor als äußert schwierig erachtet.

Zusätzlich ist zu beachten, dass die Umsetzung der Verfahren mitunter sehr zeitaufwendig sein kann. Obgleich in den veröffentlichten Ökobilanzen zunehmend Unsicherheitsanalysen dokumentiert sind (vgl. z. B. [174], S. 15), bleibt die Durchführung von Unsicherheitsanalysen zeitintensiv, sodass diese in vielen Ökobilanzstudien nicht mit ausreichender Zuverlässigkeit durchgeführt werden können (vgl. z. B. [174], S. 14).

6.4.2 Das Fehlen einer sicheren Seite

Die Ökobilanzierung basiert grundsätzlich auf einem *relativen, vergleichenden* Ansatz. Die in der Wirkungsabschätzung bilanzierten Ergebnisse stellen keine absoluten Umweltwirkungen dar, sondern repräsentieren die potentiellen, relativen Umweltwirkungen. Um Aussagen zu Umweltwirkungen machen zu können, sind daher Vergleiche erforderlich.

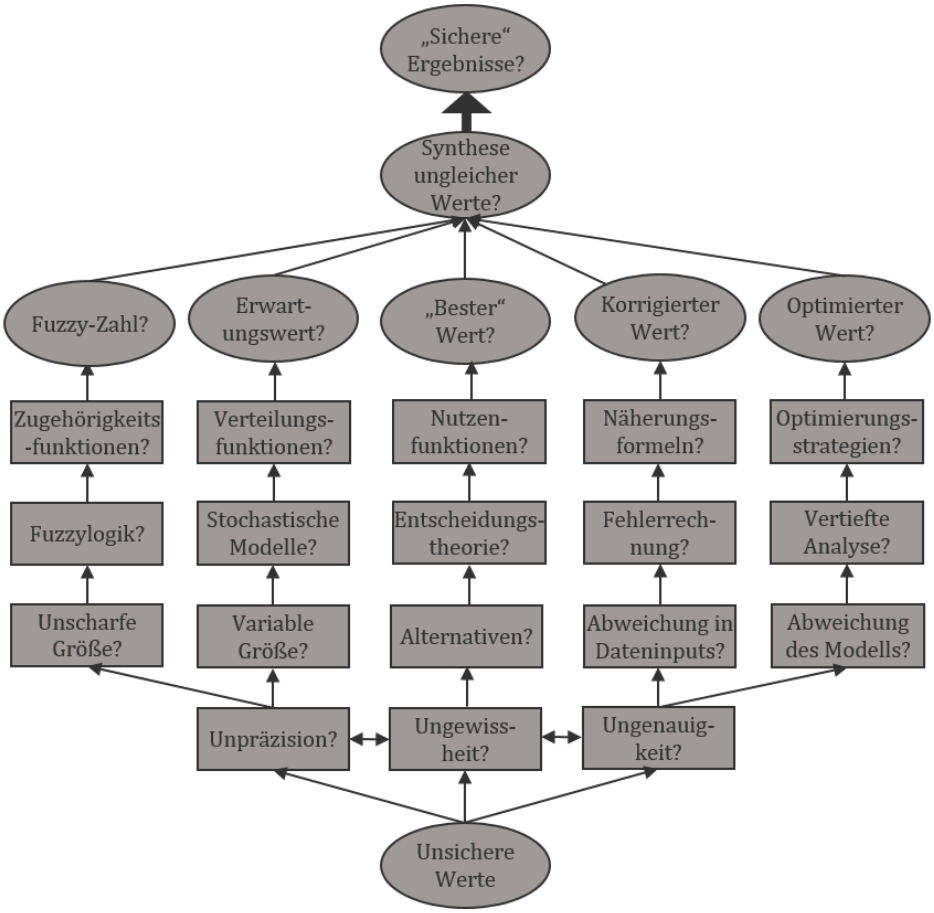


Abbildung 6.9: Diverse Arten von Unsicherheit sowie geeignete Methoden zum Umgang mit diesen.

Dieser vergleichende Ansatz hat zur Folge, dass es keine sichere Seite gibt, wie dies bspw. bei statischen Berechnungen im Bauingenieurwesen der Fall ist. Bei statischen Berechnungen werden gemäß des semiprobabilistischen Sicherheitskonzeptes höhere Lastannahmen auf der Einwirkungsseite und geringere Widerstände auf der Gegenseite durch Teilsicherheitsfaktoren berücksichtigt - mit der Folge, dass die Statik *auf der sicheren Seite liegend* bewertet werden kann. Dies kann auf die Ökobilanzierung nicht übertragen werden, was durch die folgenden Beispiele kurz erläutert wird. In Abbildung 6.10 sind von fünf Stoffen exemplarisch die Umweltwirkungen des fiktiven Umweltpotentials U abgebildet, wobei der Stoff E der Referenzstoff der Umweltpotentials U ist.

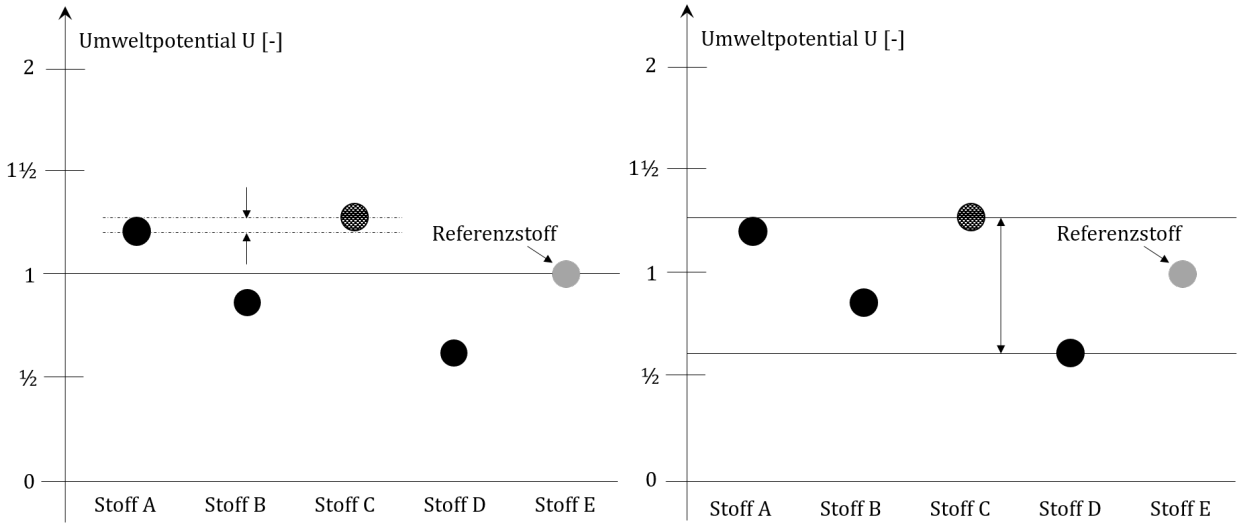


Abbildung 6.10: Fiktive Umweltwirkungen der Stoffe A bis E, sowie mögliche Aussagen bei einem Vergleich.

.

Ein Vergleich der dargestellten Stoffe kann zu folgenden Aussagen führen:

- „Die Umweltwirkungen von Stoff A und Stoff C sind vergleichsweise etwas höher als von Referenzstoff E, die Umweltwirkungen von Stoff B und Stoff D sind etwas geringer."
- „Die Umweltwirkungen von Stoff D sind vergleichsweise am geringsten, die von Stoff C sind vergleichsweise am höchsten; die von Stoff C sind im Vergleich zu Stoff D ca. doppelt so hoch."

So werden auch die Charakterisierungsfaktoren der umweltrelevanten Stoffe für die Wirkungsindikatoren durch Bezug auf einen Referenzfluss bestimmt, womit die Stoffe stets in Relation zu allen anderen Stoffen stehen. Dieser Umstand hat zur Folge, dass Charakterisierungsfaktoren nicht mit Sicherheitsfaktoren belegt werden können, um den unsicheren Einfluss einzelner Stoffe abzusichern. In Abbildung 6.11 ist das Beispiel aus der vorigen Abbildung 6.10 aufgegriffen, wobei nun angenommen wird, dass der Einfluss des Stoffes A auf das Umweltpotential U unsicher ist. So kann der Charakterisierungsfaktor des Stoffes A höher oder niedriger sein als ursprünglich angenommen. Eine mögliche Aussage kann hierzu sein:

- „Die Umweltwirkungen des Stoffes A sind vergleichsweise unsicher, die Umweltwirkungen von Stoff B, Stoff C, Stoff D und Stoff E sind vergleichsweise sicher."

Den Charakterisierungsfaktor des Stoffes A nun *auf der sicheren Seite liegend* mit einem Sicherheitsfaktor γ_R wie auf der Widerstandsseite R zu verringern bzw. wie bei der Einwirkungsseite E mit einem Sicherheitsfaktor γ_E

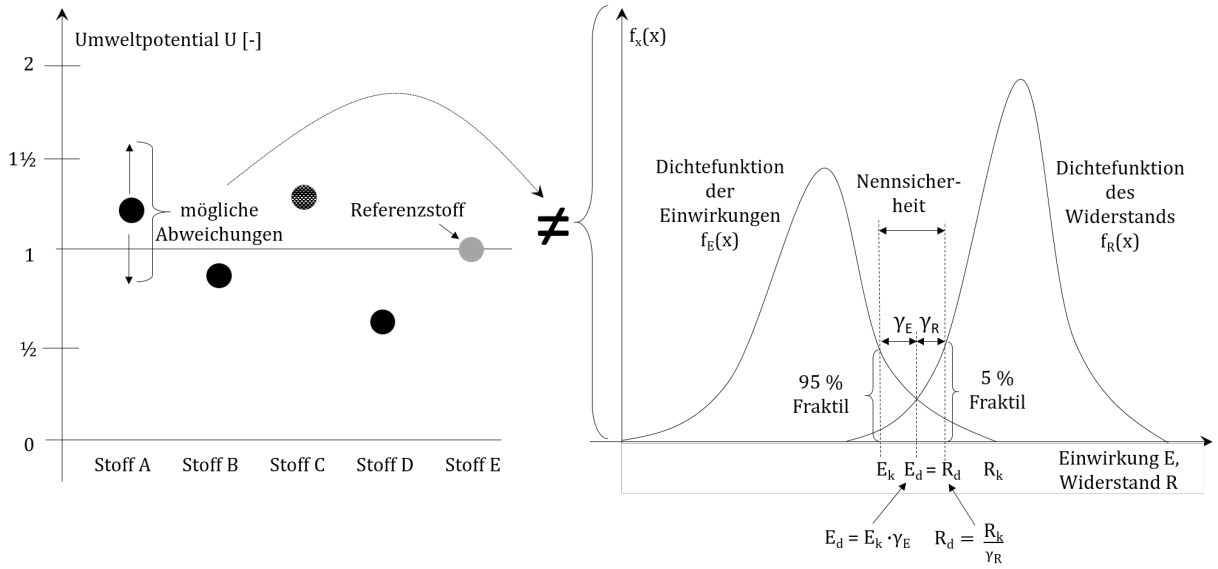


Abbildung 6.11: Umweltpotential *U* des Stoffes *A* ist unsicher, jedoch Sicherheitsfaktoren nicht anwendbar.

.

auf der sicheren Seite liegend zu erhöhen, ist in der ökologischen Bilanzierung nicht zielführend. Dies soll nun nachfolgend erläutert werden.

In Abbildung 6.12 wird hierfür das Beispiel aus Abbildung 6.10 sowie Abbildung 6.11 aufgegriffen. Der Fokus soll im Beispiel nachfolgend auf Stoff *C* gelegt werden, der weder Referenzfluss ist, noch mit einem Sicherheitsfaktor versehen wurde. Wird der Charakterisierungsfaktor von Stoff *A* reduziert, ist folgende Aussage möglich:

- „Die Umweltwirkungen von Stoff *C* sind vergleichsweise höher als die vom Referenzstoff *E*, die Umweltwirkungen von Stoff *A*, Stoff *B* und Stoff *D* sind vergleichsweise geringer."

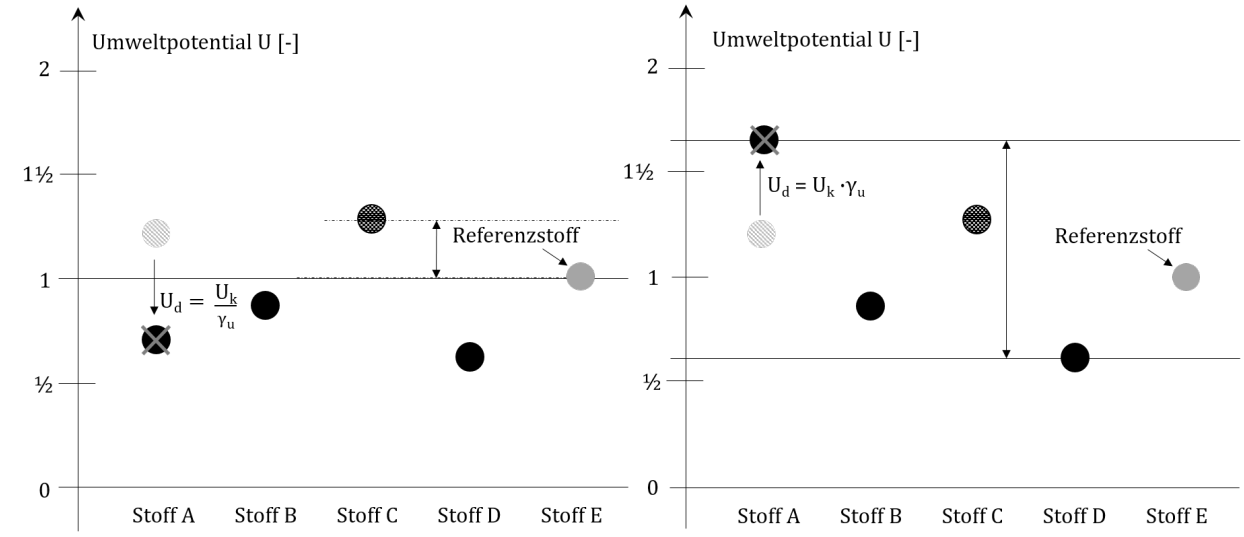


Abbildung 6.12: Sicherheitsfaktoren bei Stoff *A* beeinflussen die relative Einordnung der übrigen Stoffe.

Im Beispiel aus Abbildung 6.10 liegen die Umweltwirkungen des Stoffes *C* nur geringfügig über den Umwelt-

wirkungen des Stoffes *A*. Würde der Charakterisierungsfaktor des Stoffes *A* durch Teilsicherheitsfaktoren reduziert werden, wiese der Stoff *C* als einziger Stoff höhere Umweltwirkungen auf als der Referenzstoff *E*. Dem Stoff *C* wären dann die zweitgrößten Umweltwirkungen zugeordnet. Damit vervielfältigte sich auch die betragsmäßige Differenz zwischen den größten Umweltwirkungen (von Stoff *C*) und den vergleichsweise zweitgrößten Umweltwirkungen (jetzt von Stoff *E*), vgl. hierzu auch das linke Diagramm in Abbildung 6.12. Wird dagegen der Charakterisierungsfaktor des Stoffes *A* durch einen Sicherheitsfaktor erhöht, ist folgende Aussage möglich:

- „Die Umweltwirkungen von Stoff *D* sind vergleichsweise am geringsten, die von Stoff *A* sind vergleichsweise am höchsten; die von Stoff *A* sind im Vergleich zu Stoff *D* ca. dreimal so groß."

Im Beispiel aus Abbildung 6.10 sind die größten Differenzen zwischen den Unweltwirkungen des Stoffes *C* und denen des Stoffes *D* festzustellen, Die Umweltwirkungen von Stoff *C* sind ca. doppelt so groß wie die von Stoff *D*. In Abbildung 6.12 liegt die größte Differenz dagegen zwischen den Umweltwirkungen von Stoff *A* und Stoff *D*; die Umweltwirkungen von Stoff *A* sind ca. dreimal so groß wie die von Stoff *D* (vgl. rechtes Diagramm in Abbildung 6.12). Die Differenz zwischen Referenzstoff *E* und Stoff *C* ist in Relation zu den anderen Stoffen im linken Diagramm der Abbildung 6.12 geringer als im rechten Diagramm derselben Abbildung 6.12, obgleich die absolute Differenz in beiden Diagrammen dieselbe ist. Der beschriebene Effekt zeigt sich gleichermaßen auch bei der Mittelung von Daten.

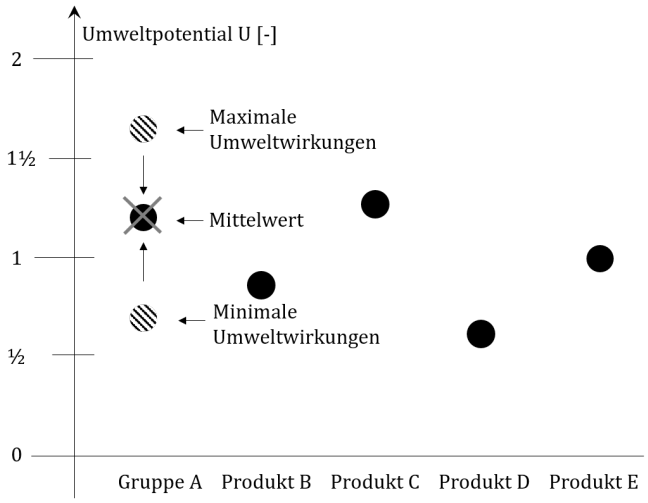


Abbildung 6.13: Mittelwerte verfälschen Bewertung von Produktgruppen und in Relation gesetzte Stoffe.

In Abbildung 6.13 wird das vorangegangene Beispiel auf den Vergleich verschiedener Produkte *A, B, C, D* und *E* übertragen; allerdings wird angenommen, dass keine Informationen vorliegen, um die Umweltwirkungen des Produktes *A* abzubilden. Stattdessen wird ein repräsentativer generischer Datensatz einer vergleichbaren Produktgruppe (Gruppe *A*) verwendet (weiterführende Informationen zur Gruppenbildung vgl. auch Kapitel 5.2.18). Dies kann bei einem Vergleich zu folgender Aussage führen:

- „Die Umweltwirkungen der Gruppe *A* liegen höher als die Umweltwirkungen der Stoffe *B, D* und *E*."

Der Datensatz der Gruppe *A* basiert aus gemittelten Werten mehrerer Produkte. Die Produktart der gruppierten Produkte von Gruppe *A* ist zwar hinsichtlich der Materialität oder Funktionalität gleich oder vergleichbar, doch die Umweltwirkungen der einzelnen Produkte können sich deutlich voneinander unterscheiden. So variiert

beispielsweise der *GWP*-Wert zur Produktion von einer Tonne Zement unter Berücksichtigung von 19 deutschen Produktionsanlagen zwischen ca. 400 kg CO_2 äq./t Zement bis fast 1000 kg CO_2 äq./t Zement (vgl. auch [187], S. 196, Fig. 1). Die jeweiligen Herstellungsprozesse, die verwendeten Öfen, die Qualität der Abgasanlagen und die sonstigen technischen Standards sowie die lokale Verfügbarkeit von Rohstoffen und Entsorgungsanlagen haben Auswirkung auf die resultierenden Umweltwirkungen. Dies hat zur Folge, dass auch die Umweltwirkungen von Produkten *desselben* Herstellers von zwei unterschiedlichen Produktionsanlagen signifikant voneinander abweichen können.

Verwendet man im Beispiel der Abbildung 6.13 den generischen Datensatz „Gruppe *A*“ zur Modellierung des Produktes *A*, kann es sein, dass die tatsächlichen, *unbekannten* Umweltwirkungen des spezifischen Produktes *A* deutlich unter- oder überschätzt werden - was sich unmittelbar auch auf die Bewertung der übrigen Produkte *B, C, D* und *E* auswirkt. Werden die Umweltwirkungen des Produktes *A unterschätzt*, hat dies zur Folge, dass die Umweltwirkungen der Stoffe *B, C, D* und *E* hingegen vergleichsweise *überschätzt* werden.

6.4.3 Intervallansatz als Lösungsvorschlag

In Abbildung 6.14 ist das Beispiel aus dem vorangegangenen Kapitel 6.4.2 der Abbildung 6.13 aufgegriffen. Dabei werden die Daten der „Gruppe *A*“ nicht gemittelt, sondern als Intervall erfasst.

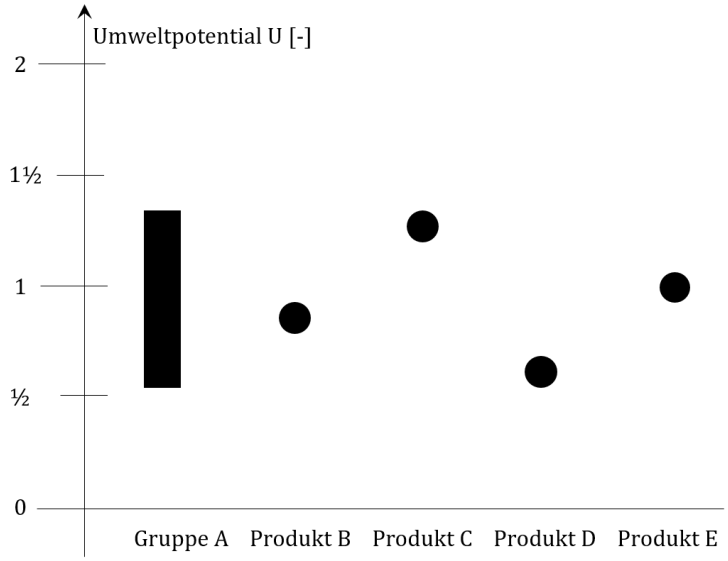


Abbildung 6.14: Intervalle ermöglichen bei relativen Ansätzen die Berücksichtigung von Unsicherheit.

Eine Zusammenfassung der Umweltwirkungen der Gruppe *A* als Intervall mit den Maximalwerten als Obergrenze und den Minimalwerten als Untergrenze hingegen verhindert, dass falsche Rückschlüsse gezogen werden (vgl. Abbildung 6.14). Eine solche Handhabe kann beispielsweise zu folgender Aussage führen:

- „Die Umweltwirkungen der Gruppe *A* liegen im Bereich der Umweltwirkungen der Stoffe *B, C, D* und *E*.“.

Auch Entscheidungsfindungen, bei denen ökologische Aspekte berücksichtigt werden sollen, werden in der Regel durch Vergleiche mit Alternativen hinsichtlich ihrer „Ökologie“ bewertet. Sie erfolgen dann nicht auf Basis von festgelegten Obergrenzen wie beim semiprobabilistischen Ansatz statischer Berechnungen. Dies bedeutet, dass mögliche Abweichungen und Unsicherheiten in einer Ökobilanz möglichst umfassend berücksichtigt wer-

den müssen, ohne dabei den relativen Ansatz der Methodik außer Acht zu lassen. Ein Lösungsvorschlag ist das intervallarithmetische Verfahren, das in dieser Arbeit im Mittelpunkt steht.

In einigen Veröffentlichungen (vgl. neben [167]und [168] z. B. auch [188] sowie [189]) wird die Fuzzylogik als Ansatz zur Abschätzung von Unsicherheiten vorgeschlagen. Damit wird eine *Erfassung und Integration* der Unsicherheit empfohlen, anstatt die Unsicherheit so weit wie möglich zu entfernen (vgl. auch die allgemeine Wertung von Unsicherheit in Kapitel 6.1.1). Die Theorie der Fuzzylogik basiert auf Intervallen; die Intervallarithmetik kann als eine Vereinfachung der Fuzzylogik aufgefasst werden, da sie weniger Intervalle zur Beschreibung einer unsicheren Information erfordert. Die Idee, die Ökobilanzierung intervallbasiert durchzuführen, um Unsicherheiten umfassend berücksichtigen zu können, wurde bereits in den 1990er Jahren vorgeschlagen (vgl. z. B. [190] sowie [191]). Der Vorschlag wird in dieser Arbeit aufgegriffen.

Es wird die These vertreten, dass die intervallbasierte Ökobilanzierung ein alternatives, praktikables Verfahren zur Abschätzung potentieller Umweltwirkungen darstellt, mit dem es möglich ist, eine große Anzahl von Unsicherheiten direkt in der Ökobilanz transparent zu berücksichtigen.

Kapitel 7

Graphentheoretische Betrachtungen

Das in dieser Arbeit entwickelte Verfahren basiert auf Bereichen der Intervallarithmetik, der objektorientierten Programmierung sowie der Graphentheorie. Graphen dienen auch zur Visualisierung algebraischer Strukturen wie Mengen in der Mathematik sowie zur Illustration von Datenstrukturen wie Objekten der objektorientierten Programmierung (vgl. auch Abbildung 7.1). Die Abbildungen aus dem Gebiet der Mathematik sowie die Methoden der objektorientierten Programmentwicklung können durch Kanten dargestellt werden. Die Vektorräume der Mathematik und Klassen der Programmierung legen die wesentliche Struktur fest; in der Graphentheorie erfolgt dies durch die Klassifikation von Graphentypen.

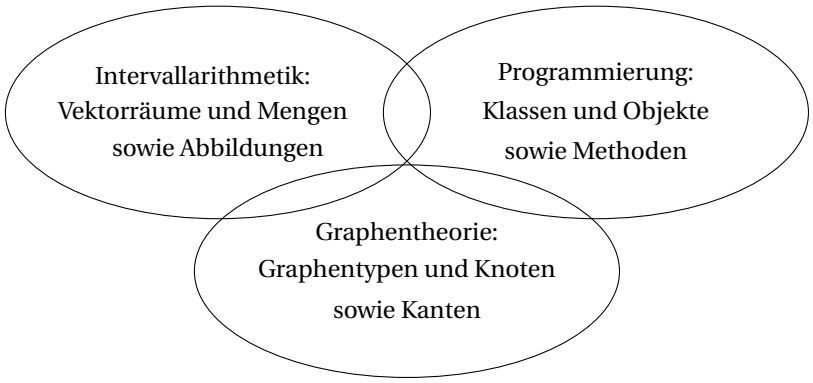


Abbildung 7.1: Strukturen der Intervallarithmetik, objektorientierten Programmierung und der Graphentheorie.

Im Folgenden werden mögliche Strukturen identifiziert, die in Produktsystemen auftreten können. Diese werden in Kapitel 7.1 aufgeführt. Einfache Produktsysteme werden in Kapitel 7.2 erläutert, komplizierte Produktsysteme werden in Kapitel 7.3 vorgestellt. In Kapitel 7.4 wird das Produktsystem graphentheoretisch in die Methodik der Ökobilanzierung eingeordnet. Die tabellarische Darstellung der Produktsysteme erfolgt in Kapitel 7.5, wobei die einhergehenden Produktmatrizen in Kapitel 7.6 diversen Matrixtypen zugeordnet werden. Zuletzt werden in Kapitel 7.7 Besonderheiten behandelt, die im Zuge der Modellierung von Produktsystemen zu beachten sind.

7.1 Produktsysteme als Graphen

Produktsysteme können als Graphen dargestellt werden, wobei die Prozessmodule die Knoten und die Flüsse die Kanten der Graphen bilden. Die Prozessmodule von Produktsystemen werden üblicherweise als Rechtecke dargestellt (vgl. Abbildung 7.3). Es sollen nur Prozessmodule abgebildet werden, welche unmittelbar zur Erzeugung oder Verarbeitung von Zwischen- oder Endprodukten beitragen (vgl. hierzu auch Kapitel 5.2.12). Daher sind *alle* Prozessmodule über *mindestens* einen Produktfluss mit *mindestens* einem anderen Prozessmodul verbunden. Die Graphen von Produktsystemen können zwischen einfachen und komplizierten Graphen unterschieden werden. Demnach können die Prozessmodule von Produktsystemen linear angeordnet sein, einer baumartigen Struktur entsprechen oder auch komplizierte Strukturen aufweisen. Als komplizierte Strukturen werden im Folgenden diejenigen Strukturen bezeichnet, deren Graphen nicht frei sind von Zyklen oder Maschen. Eine Übersicht zu verschiedenen Strukturen, zwischen denen in dieser Arbeit unterschieden wird, ist in Abbildung 7.2 dargestellt.

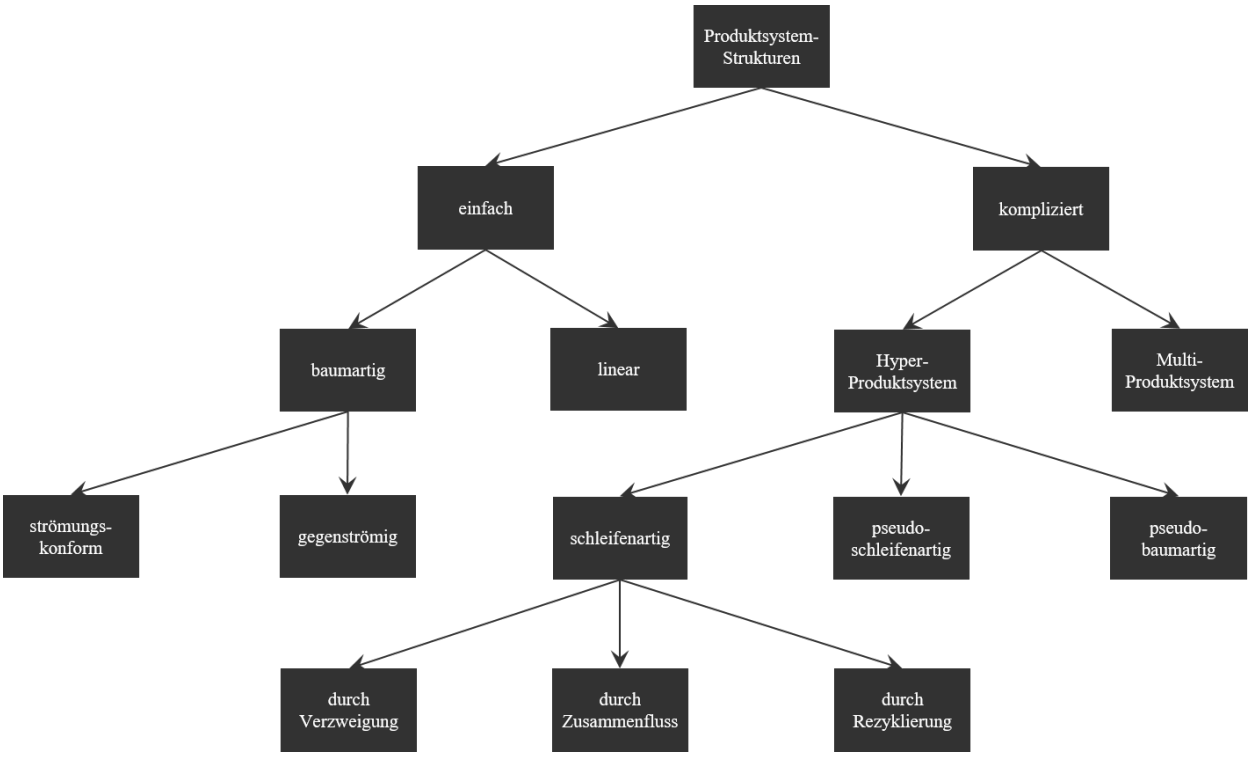


Abbildung 7.2: Übersicht über die Unterscheidung diverser Produktsystemstrukturen.

In der Praxis setzen sich die Produktsysteme aus verschiedenen Strukturen zusammen. Die Prozessmodule weisen i. d. R. auch Elementarflüsse auf (ansonsten ist die Durchführung einer Ökobilanzierung nicht notwendig, da keine Flüsse in die Umwelt abgegeben werden, die umweltrelevant sein könnten); diese führen über die Systemgrenze des Produktsystems hinweg zu weiteren, nicht im Untergraph enthaltenen „Knoten“, welche die Umweltwirkungen repräsentieren. Zur Abbildung der inneren Struktur der Produktsysteme sind die Elementarflüsse zunächst nicht von Relevanz. Im Folgenden werden die Strukturen von möglichen Produktsystemen am Beispiel *herstellender* Produktsysteme erläutert; dieselben Produktsystemstrukturen sind analog bei *verarbeitenden* Produktsystemen zu erwarten (Definition von „herstellenden Produktsystemen“ sowie „verarbeitenden Produktsystemen“ vgl. Kapitel 5.2.12).

7.2 Strukturen einfacher Produktsysteme

Einfache Produktsysteme können in lineare und baumartige Produktsysteme unterschieden werden. Diese Systemstrukturen können im gesamten Produktsystem vorliegen oder auch nur in Teilgraphen des Produktsystems, d. h. in Produkt*sub*systemen. Im Folgenden wird allgemein von *Produktsystemen* gesprochen, wobei dies gleichsam auch für *Produktsubsysteme* gilt.

7.2.1 Lineare Produktsysteme

Im einfachsten Fall handelt es sich bei einem Produktsystem um einen linearen Graph, dessen Prozessmodule jeweils nur genau einen Vorgänger und einen Nachfolger aufweisen.

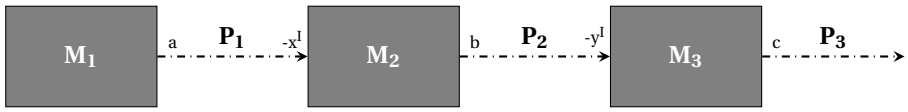


Abbildung 7.3: Lineares, *erzeugendes* Produktsystem, bestehend aus drei Prozessmodulen.

In Abbildung 7.3 ist exemplarisch ein solches lineares Produktsystem abgebildet, das aus den drei Prozessmodulen M_1, M_2 und M_3 besteht. Es werden die Zwischenprodukte P_1 und P_2 sowie das Endprodukt P_3 mit der Menge *c* erzeugt.

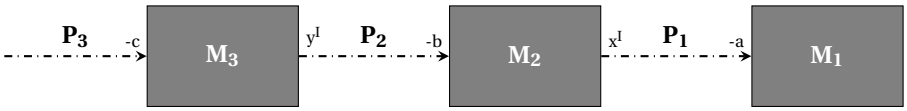


Abbildung 7.4: Lineares, *verarbeitendes* Produktsystem, bestehend aus drei Prozessmodulen.

In Abbildung 7.4 ist zur Vollständigkeit und späteren Erläuterungszwecken analog ein lineares, *verarbeitendes* Produktsystem abgebildet, durch welches das Endprodukt P_3 mit der Menge *c* verarbeitet wird.

7.2.2 Baumartige Produktsysteme

Ein baumartiges Produktsystem liegt vor, wenn der gerichtete Graph einem Baum entspricht (Definition von „Baum“ vgl. Kapitel 4.2.1). Diese Systemstrukturen können zusätzlich hinsichtlich ihrer Kantenrichtungen differenziert werden. Unter Berücksichtigung dieser wird unterschieden zwischen „gegenströmigen Bäumen“ und „strömungskonformen Bäumen“, deren Strukturen nachfolgend erläutert werden.

Gegenströmige Bäume

Ein gegenströmiger Baum ist gegeben, wenn die Wurzel die Nachfolger der Blätter darstellt und daher entgegen der Flussrichtung aufgespannt wird. Diese Systemstrukturen sind in der Praxis häufig der Fall, da für die Herstellung eines Produkts in der Regel mehr als ein Vorprodukt benötigt wird. Bei diesen baumartigen Produktsystemen weist ein Prozessmodul, das als „Wurzel“ bezeichnet werden kann, mehrere Inputs auf, die gesondert in vorangehenden Prozessmodulen erzeugt werden (vgl. Abbildung 7.5).

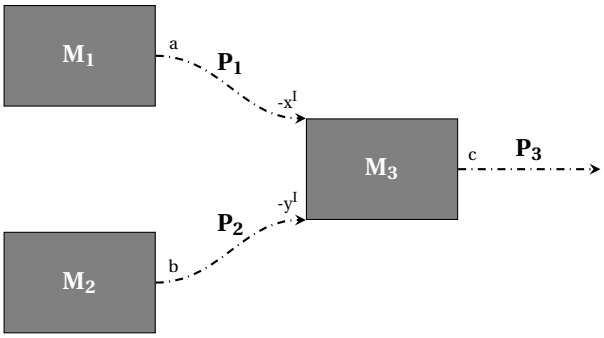


Abbildung 7.5: Gegenströmiger Baum: Baumartiges Produktsystem, entgegen der Flussrichtung aufgespannt.

Strömungskonforme Bäume

Ein strömungskonformer Baum ist gegeben, wenn die Wurzel das vorangehende Prozessmodul der nachfolgenden Blätter darstellt und daher entlang der Flussrichtung aufgespannt wird. Strömungskonforme Bäume können entstehen, wenn neben dem zu produzierenden Hauptprodukt Zusatzprodukte erzeugt und behandelt werden (Definition „Zusatzprodukte“ vgl. Kapitel 5.2.9). Sie stellen damit streng genommen ein Mischsystem aus verarbeitendem und erzeugendem Produktsystem dar. In Abbildung 7.6 ist exemplarisch ein solches Produktsystem mit dem Produktfluss P_b sowie dem Prozessmodul M_b dargestellt. Repräsentiert der Produktfluss P_b Abfall, wird dieser durch den in M_b dargestellten Beseitigungsprozess endgültig entsorgt. Wird durch den Produktfluss P_b ein Koppelprodukt beschrieben, wird dieses in M_b durch Anwendung der Gutschriftenmethode substituiert.

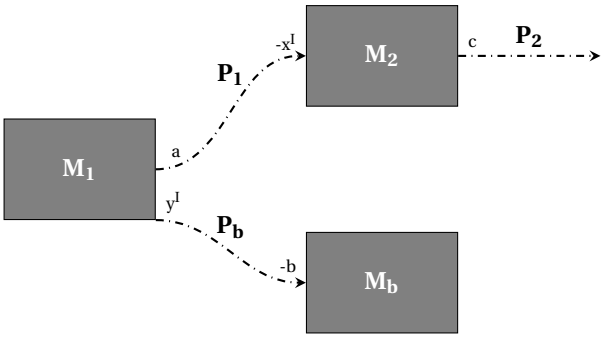


Abbildung 7.6: Strömungskonformer Baum: Baumartiges Produktsystem, entlang der Flussrichtung aufgespannt.

7.3 Strukturen komplizierter Produktsysteme

Komplizierte Systemstrukturen treten auf, wenn interne Koppelprodukte erzeugt werden oder wenn ein erzeugtes Hauptprodukt in mehreren Prozessmodulen als Input benötigt wird. Treten derlei Systemstrukturen an einem Subsystem des Produktsystems auf, wird das gesamte Produktsystem als kompliziert bezeichnet. Die Subsysteme des Produktsystems können jedoch weiterhin einfachen Systemstrukturen entsprechen. Die Strukturen solcher komplizierter Systeme können hinsichtlich der Kantentypen differenziert werden: Produktsysteme mit Multikanten werden im Folgenden auch als Multi-Produktsysteme bezeichnet, Produktsysteme mit Hyperkanten als Hyper-Produktsysteme.

7.3.1 Multi-Produktsysteme

Multi-Produktsysteme, die Produktsysteme mit parallel verlaufenden Produktflüssen darstellen, entstehen dann, wenn durch ein Prozessmodul neben dem Hauptprodukt mindestens ein internes Koppelprodukt erzeugt wird, welches in keinem anderen Prozessmodul als Hauptprodukt generiert wird und gleichzeitig mindestens zwei der erzeugten Produkte in *einem* weiteren Prozessmodul des Produktsystems als Inputs Anwendung finden.

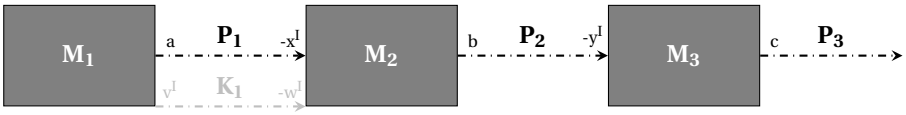


Abbildung 7.7: Multi-Produktsystem mit linearer Grundstruktur, bestehend aus drei Prozessmodulen.

In Abbildung 7.7 ist ein Multi-Produktsystem mit linearer Grundstruktur und einem Koppelproduktfluss dargestellt, der parallel zum zugehörigen Fluss des Hauptprodukts verläuft. Der Fluss des Koppelprodukts K_1 entspricht nur dann den Modellierungskriterien (Erläuterung der „Modellierungskriterien“ vgl. Kapitel 8.4.3) des in dieser Arbeit vorgestellten Verfahrens, wenn die Parameter v^I, w^I, x^I und y^I des gesamten Produktsystems Intervallweiten von null aufweisen und $a/x^I = v^I/w^I$ gilt. In diesem Fall wird der Koppelproduktfluss bei der Berechnung des Skalierungsvektors nicht in der Produktmatrix aufgeführt. Die dargestellte Struktur stellt daher einen Sonderfall dar; in der Regel sind bei Auftreten eines internen Koppelprodukts weitere Prozessmodule zur Deckung des Bedarfs oder zur Behandlung eines Überschusses zu erwarten (Modellierung von Koppelprodukten vgl. auch Kapitel 7.7.3).

7.3.2 Hyper-Produktsysteme

Produktsysteme mit Hyperkanten entstehen dann, wenn ein Produkt mehrfach im selben System benötigt wird oder im Produktsystem ein internes Koppelprodukt auftritt. Diese Systemstrukturen können differenziert werden in „pseudo-baumartige Produktsysteme“, „pseudo-schleifenartige Produktsysteme“ und „schleifenartige Produktsysteme“, deren Strukturen nachfolgend erläutert werden.

7.3.3 Pseudo-baumartige Produktsysteme

Pseudo-baumartige Produktsysteme mit sich verzweigenden Produktflüssen entstehen dann, wenn ein Produkt in mehreren Prozessmodulen benötigt wird. Ihre grafische Darstellung ähnelt einem strömungskonformen Baum mit einem Prozessmodul, welches eine „Wurzel“ repräsentiert und zwei Prozessmodulen, welche „Blätter“ darstellen. Der Unterschied ist, dass das einer „Wurzel“ ähnelnde Prozessmodul M_1 über *einen*, sich verzweigenden Produktfluss, einer Hyperkante, mit den nachfolgenden Prozessmodulen verbunden ist. In Abbildung 7.8 ist ein pseudo-baumartiges Hyper-Produktsystem mit dem sich verzweigenden Produktfluss P_1 dargestellt. Das Produkt P_1 wird sowohl im Prozessmodul M_2 mit x^I-facher Menge als auch im ProzessmodulM_3 mit y^I-facher Menge benötigt. Dadurch hängt die notwendige Bedarfsmenge a sowohl von M_2 als auch M_2 ab.

7.3.4 Pseudo-schleifenartige Produktsysteme

Pseudo-schleifenartige Produktsysteme können in Produktsystemen mit internen Koppelprodukten entstehen. In Abbildung 7.9 ist ein Produktsystem mit einfachem Kantenzusammenfluss abgebildet, bei dem das Hyperproduktsystem mit linearer Grundstruktur gestört wird durch den Zusammenfluss des Produktflusses P_2.

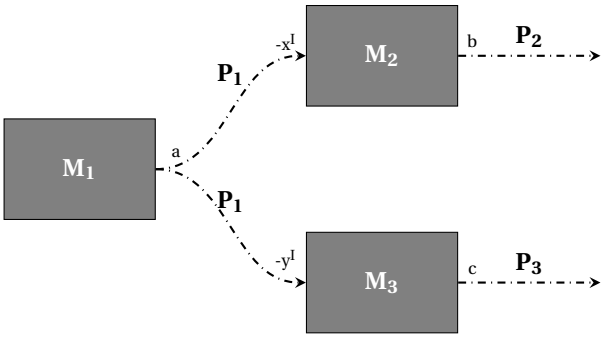


Abbildung 7.8: Pseudo-baumartiges Produktsystem.

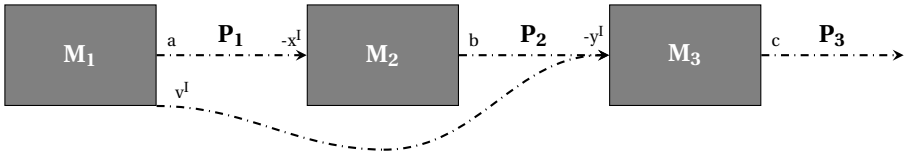


Abbildung 7.9: Produktsystem mit einfachem Kantenzusammenfluss und linearer Grundstruktur.

Das Produkt P_2 wird sowohl im Prozessmodul M_1 als auch M_2 erzeugt. Das erzeugte (Koppel-)Produkt P_2 wird mit y^I-facher Menge im Prozessmodul M_3 benötigt. Da M_1 und M_2 den Bedarf von P_2 des Prozessmoduls M_3 gemeinsam decken, entsteht eine mehrfache Abhängigkeit zwischen den Prozessmodulen M_1, M_2 und M_3.

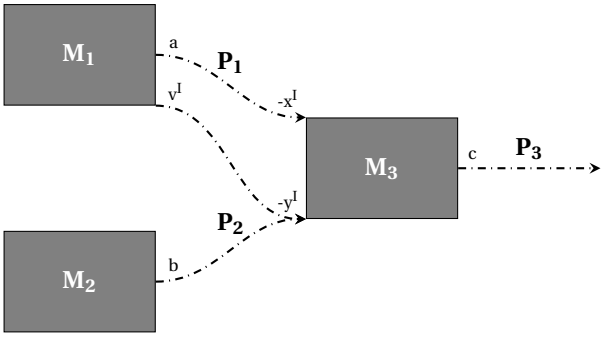


Abbildung 7.10: Produktsystem mit einfachem Kantenzusammenfluss und baumartiger Grundstruktur.

Analog dazu ist in Abbildung 7.10 ein Produktsystem mit einfachem Kantenzusammenfluss abgebildet, dessen *baumartige* Grundstruktur gestört wird durch den Zusammenfluss des Produktflusses P_2. Das Produkt P_2 wird sowohl im Prozessmodul M_1 als auch im Prozessmodul M_2 erzeugt, wodurch eine mehrfache Abhängigkeit zwischen den Prozessmodulen M_1, M_2 und M_3 entsteht. Des Weiteren ist es möglich, dass in einem Produktsystem mehrere interne Koppelprodukte auftreten.

In Abbildung 7.11 ist exemplarisch ein Hyper-Produktsystem mit *zwei* Kantenzusammenflüssen und *linearer* Grundstruktur dargestellt, bei dem die (Koppel-)Produktflüsse P_2 und P_3 in den Prozessmodulen M_1 und M_2 bzw. M_2 und M_3 erzeugt werden. Dies führt zu multiplen Abhängigkeiten zwischen den Prozessmodulen M_1, M_2, M_3 und M_4. In Abbildung 7.12 ist analog exemplarisch ein Hyper-Produktsystem mit *zwei* Kantenzusammenflüssen und *baumartiger* Grundstruktur dargestellt, bei dem die (Koppel-)Produktflüsse P_1 und P_2 in den Prozessmodulen

M_1 und M_2 erzeugt werden. Dies führt zu multiplen Abhängigkeiten zwischen den Prozessmodulen M_1, M_2 und M_3.

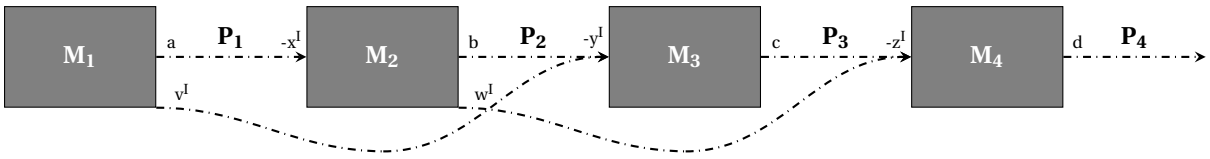


Abbildung 7.11: Produktsystem mit zweifachem Kantenzusammenfluss und linearer Grundstruktur.

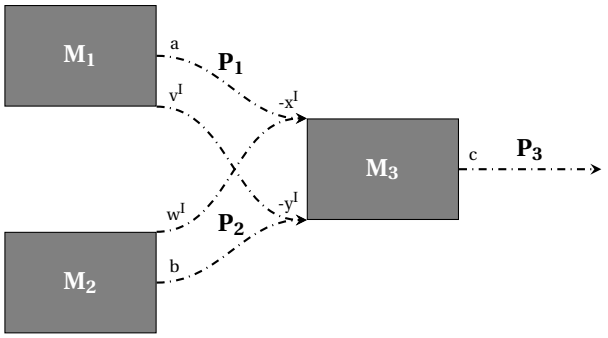


Abbildung 7.12: Produktsystem mit zweifachem Kantenzusammenfluss und baumartiger Grundstruktur.

7.3.5 Schleifenartige Produktsysteme

Schleifenartige Produktsysteme liegen vor, wenn mindestens ein internes Koppelprodukt in einem *vorangehenden* Prozessmodul, mitunter durch vorherige Aufbereitung, verwendet wird oder aber wenn ein erzeugtes Produkt in einem *vorangehenden* Prozessmodul als Input benötigt wird. Diese Systemstrukturen können differenziert werden in „schleifenartige Produktsysteme durch Zusammenfluss“, „schleifenartige Produktsysteme infolge von Rezyklierprozessen“ und „schleifenartige Produktsysteme durch Verzweigung“, deren Strukturen nachfolgend erläutert werden.

Schleifenartige Produktsysteme durch Zusammenfluss

Schleifenartige Produktsysteme durch Zusammenfluss entstehen, wenn ein internes Koppelprodukt in einem *vorangehenden* Prozessmodul verwendet wird. Exemplarisch ist ein solches Produktsystem in Abbildung 7.13 dargestellt. Streng genommen kann das dort dargestellte Koppelprodukt P_1 nicht im selben Produktionsprozess verwendet werden, da das Prozessmodul M_3 zeitlich nach M_2 durchgeführt wird. Das erzeugte Koppelprodukt P_1 kann aber bei einem zeitlich nachfolgenden Produktionsprozess verwendet werden.

Die einmalig zur Inbetriebnahme der Produktionslinie zusätzlich bereitzustellende Menge w^{I} des Produkts P_1 wie auch die einmalig übrig verbleibende Menge w^{I} bei Stilllegung wird aufgrund der anzunehmenden sehr großen Anzahl von Prozessabläufen in der Praxis im Regelfall als vernachlässigbar gering eingestuft und daher in der Bilanzierung nicht berücksichtigt. Stattdessen wird vereinfacht davon ausgegangen, dass die im Prozessmodul M_2 notwendige Menge x^{I} des Produkts P_1 gemeinsam von M_1 und M_3 bereitgestellt wird, weshalb eine mehrfache Abhängigkeit zwischen den Prozessmodulen M_1, M_2 und M_3 besteht.

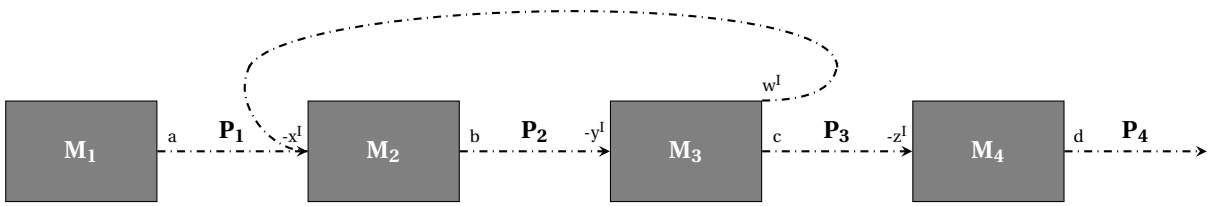


Abbildung 7.13: Schleifenartiges Produktsystem durch Zusammenfluss.

Schleifenartige Produktsysteme durch Verzweigung

Schleifenartige Produktsysteme durch Verzweigung können entstehen, wenn ein erzeugtes Produkt in einem *vorangehenden* Prozessmodul als Input benötigt wird. Ein solches Produktsystem ist exemplarisch in Abbildung 7.14 dargestellt. Analog zum in Abbildung 7.13 dargestellten schleifenartigen Produktsystem durch Zusammenfluss kann das in Abbildung 7.14 dargestellte schleifenartige Produktsystem durch Verzweigung das in M_3 hergestellte Produkt P_3 nicht in demselben Produktionsprozess verwendet werden, sondern nur im nachfolgenden Produktionsprozess, da das Prozessmodul M_3 zeitlich nach M_2 durchgeführt wird.

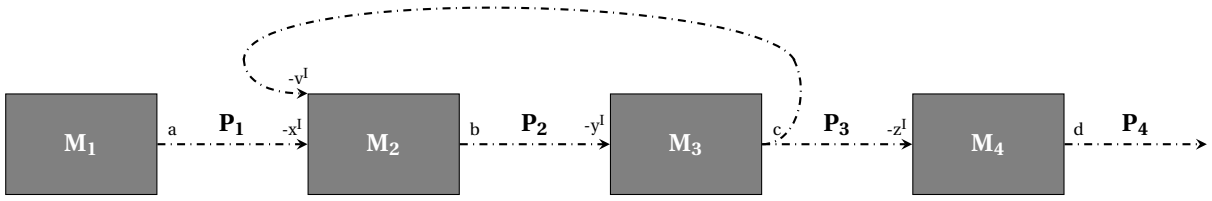


Abbildung 7.14: Schleifenartiges Produktsystem durch Verzweigung.

Dieser Aspekt wird auch bei schleifenartigen Produktsystemen durch Verzweigung in der vereinfachten Berechnung vernachlässigt. In der Modellierung weist somit der Produktfluss P_3 eine Verzweigung auf, weshalb eine mehrfache Abhängigkeit zwischen den Prozessmodulen M_2 und M_3 besteht.

Schleifenartige Produktsysteme infolge von Rezyklierprozessen

Schleifenartige Produktsysteme infolge von Rezyklierprozesse können entstehen, wenn in den Produktsystemen Zusatzprodukte erzeugt werden, welche vor ihrer Weiterverwendung in einem gesonderten Prozessmodul zu einem internen Zwischenprodukt aufbereitet werden. Sie stellen eine besondere Form von schleifenartigen Zusammenflüssen dar, da zwei Aspekte vereint auftreten: So wird durch den Rezyklierprozess nicht nur ein Zusatzprodukt aufbereitet, sondern es wird dadurch auch ein internes Zwischenprodukt hergestellt, das bereits in einem anderen Prozessmodul als Hauptprodukt produziert wird. Gemäß den in dieser Arbeit für das intervallbasierte Berechnungsverfahren festgelegten Modellierungskriterien darf der Output des Rezyklierprozesses eine Intervallweite ungleich null aufweisen, der Input hingegen muss als Punktintervall bestimmt werden (Erläuterung der Modellierungskritierien vgl. Kapitel 8.4.3).

Ein solches Produktsystem ist exemplarisch in Abbildung 7.15 dargestellt. Dabei wird im Prozessmodul M_b das Zusatzprodukt P_b generiert, das zur weiteren Verwendung in Prozessmodul M_b zum Produkt P_1 aufbereitet wird. Das Produkt P_1 wird mit einem zusätzlichen Outputwert erfasst, welches gemäß der Modellierungskriterien in

Form eines unsicheren Intervalls v^{I} modelliert werden kann. Beim Rezyklierprozess M_b hingegen muss der *Input* des Hauptproduktflusses P_b als Punktintervall definiert werden (Definition von „Punktintervall“ vgl. Kapitel 3.1).

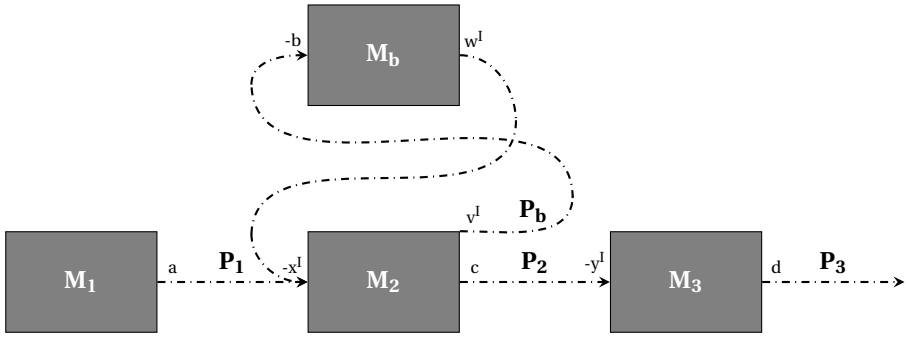


Abbildung 7.15: Schleifenartiges Produktsystem infolge eines Rezyklierprozesses.

7.4 Indikatoren und Produktsysteme als Teilgraphen

In der Ökobilanzierung bilden die Produktsysteme Untergraphen eines übergeordneten Graphen, welcher das Gesamtsystem repräsentiert und sich aus den Teilgraphen der Produktsysteme *und* den Knoten und Kanten der Wirkungsabschätzung zusammensetzt.

Die Elementarflüsse können somit als Kanten von Graphen interpretiert werden, welche das modellierte Produktsystem mit weiteren Knoten der Wirkungsabschätzung ergänzen. Diese Knoten repräsentieren jedoch weniger Ökoysteme bzw. Systemkomponenten von Ökosystemen - sie sind vielmehr die Wirkungsindikatoren, mit denen die potentiellen Einflüsse auf die Umwelt abgeschätzt werden sollen. Der in der Wirkungsabschätzung dargestellte Graph besteht folglich aus Knoten, welche Systeme aus dem Bereich der Technosphäre, d. h. Produktsysteme, repräsentieren, die mit Knoten, welche Wirkungskategorien zur Beschreibung der Umweltwirkungen auf die Ökosphäre, d. h. Wirkungsindikatoren, repräsentieren, verbunden sind.

Ein Beispiel hierzu ist in Abbildung 7.16 illustriert. In diesem ist vereinfacht ein Ausschnitt des Ökosystems Erde mit den Subsystemen „Atmosphäre“, „Böden“ und „Meere“ dargestellt sowie die Indikatoren „Treibhauspotential“, „Ozonabbaupotential“ als auch „Versauerungspotential“. Die Indikatoren stellen keine Ökosystemmodelle dar, da sie nur aus daraus abgeleiteten Charakterisierungsfaktoren bestehen. Sie werden daher in Abbildung 7.16 bewusst außerhalb der grafisch veranschaulichten Subsysteme des Ökosystems dargestellt und nur durch durchgehende Linien mit diesen gekoppelt. Da die Emissionen als Elementarflüsse gemäß dem Ansatz der Ökobilanzierung die Technosphäre verlassen, sind sie außerhalb der Produktsysteme eingezeichnet.

Graphen, die Elementarflüsse und Indikatoren abbilden, können analog zu Produktsystemen Hyper- und Multigraphen sein. Analog zu den Produktflüssen können Elementarflüsse sich verzweigen, zusammenfließen oder parallel zueinander verlaufen. So ist es bspw. möglich, dass ein Elementarfluss mehreren Wirkungskategorien zugeordnet wird, was bedeutet, dass ein Elementarfluss sich analog zu Produktflüssen verzweigen kann. Dies ist exemplarisch in 7.16 mit dem Elementarfluss E_4 dargestellt, der mit der Menge e sowohl zum Ozonabbaupotential als auch mit der Menge a zum Treibhauspotential beiträgt. Die Wirkungsabschätzung erfolgt erst nach der Sachbilanz; die Elementarflüsse verlassen daher als zusammengefasste Flüsse das Produktsystem. So emittieren bspw. sowohl das Prozessmodul M_1 als auch das Prozessmodul M_2 die Emission E_3 mit den Mengen f_1 bzw. f_2. Im Zuge der Wirkungsabschätzung werden kumulierte Elementarflüsse mit dem Charakterisierungsfaktor multi-

pliziert und anschließend die Indikatorwerte aufaddiert (weiterführende Informationen zur Berechnung vgl. auch Kapitel 5.4.2).

Bei dem in dieser Arbeit entwickelten intervallbasierten Berechnungsverfahren kommt möglichst schmalen Ergebnisintervallen eine bedeutende Rolle zu. Hierbei kann die Reihenfolge und Anzahl der Rechenschritte einen Einfluss auf die Intervallweiten haben. In der Wirkungsabschätzung sind in der Regel infolge unterschiedlicher Rechenwege keine Differenzen in den Intervallweiten zu erwarten; eine Vertauschung der Faktoren oder Summanden hat keinen Einfluss auf die Intervallweiten des Ergebnisses. Bei der Ermittlung des Skalierungsvektors hingegen können unterschiedliche Rechenwege zu unterschiedlichen Ergebnissen führen. Daher wird in dieser Arbeit der Schwerpunkt auf die Berechnung des Skalierungsvektors gelegt.

7.5 Tabellarische Darstellung von Produktsystemen

Im Zuge von matrixbasierten Verfahren zur Durchführung einer Ökobilanzierung können Produktsysteme tabellarisch in Form von Produktmatrizen dargestellt werden (Definition von Produktmatrix vgl. auch Kapitel 5.4.2). Die unterschiedlichen Matrixtypen, welche sich bei den jeweiligen Produktsystemstrukturen ergeben können, werden im Folgenden erläutert.

7.5.1 Allgemeine Besonderheiten von Produktmatrizen

Die Produktmatrizen weisen typische Muster auf. So stellt eine Produktmatrix eine transponierte Inzidenzmatrix des zugehörigen Graphen dar. Die Inzidenzmatrix eines Produktsystems P mit n Prozessmodulen weist eine Dimension von $(\mathbb{I})\mathbb{R}^{nxn}$ auf, da einem Prozessmodul genau ein Hauptproduktfluss zugeordnet wird. Die Transponierung der Inzidenzmatrix ist zur Berechnung notwendig. So ist der Bedarf an Produkten bekannt, die Skalierungsfaktoren der Prozessmodule hingegen sind unbekannt. Im Zuge der Sachbilanz wird daher der Skalierungsvektor ermittelt.

Die Hauptdiagonalen von Produktmatrizen enthalten bei entsprechender Ordnung der Matrix die Hauptproduktelemente. In der n-ten Zeile ist das zu erzielende Endprodukt des Produktsystems hinterlegt, welches (bei den hier exemplarisch repräsentierten *erzeugenden* Produktsystemen) als Output das Produktsystem verlässt. Die Besonderheiten der Produktmatrizen lassen sich auf verarbeitende Produktsysteme übertragen. Weitere Besonderheiten, die sich bei einer (soweit möglichen) Umformung von Gleichungssystemen mit einer solchen Produktmatrix zu einer oberen Dreiecksmatrix ableiten lassen, werden im Folgenden für die jeweiligen Produktsystemstrukturen erläutert. Kann eine Produktmatrix zu einer Dreiecksmatrix umgeformt werden, ist die Determinante das Produkt der Hauptprodukte auf der Hauptdiagonalen.

7.5.2 Produktmatrizen linearer Produktsysteme

Produktmatrizen linearer Produktsysteme lassen sich stets als obere Dreiecksmatrix umformen. Die Determinante ergibt sich dann durch Multiplikation der Diagonalelemente a_{11} bis a_{nn}.

Produktmatrizen linearer, erzeugender Produktsysteme

Das in Abbildung 7.17 dargestellte Gleichungssystem $GS_{lin,erz.}$ repräsentiert ein lineares Produktsystem mit n Prozessmodulen, wobei die Hauptprodukte nach entsprechender Matrixumformung in Form von positiven Punktintervallen auf der Hauptdiagonalen hinterlegt sind. Lineare Produktsysteme sind koppelproduktfrei, d. h. das

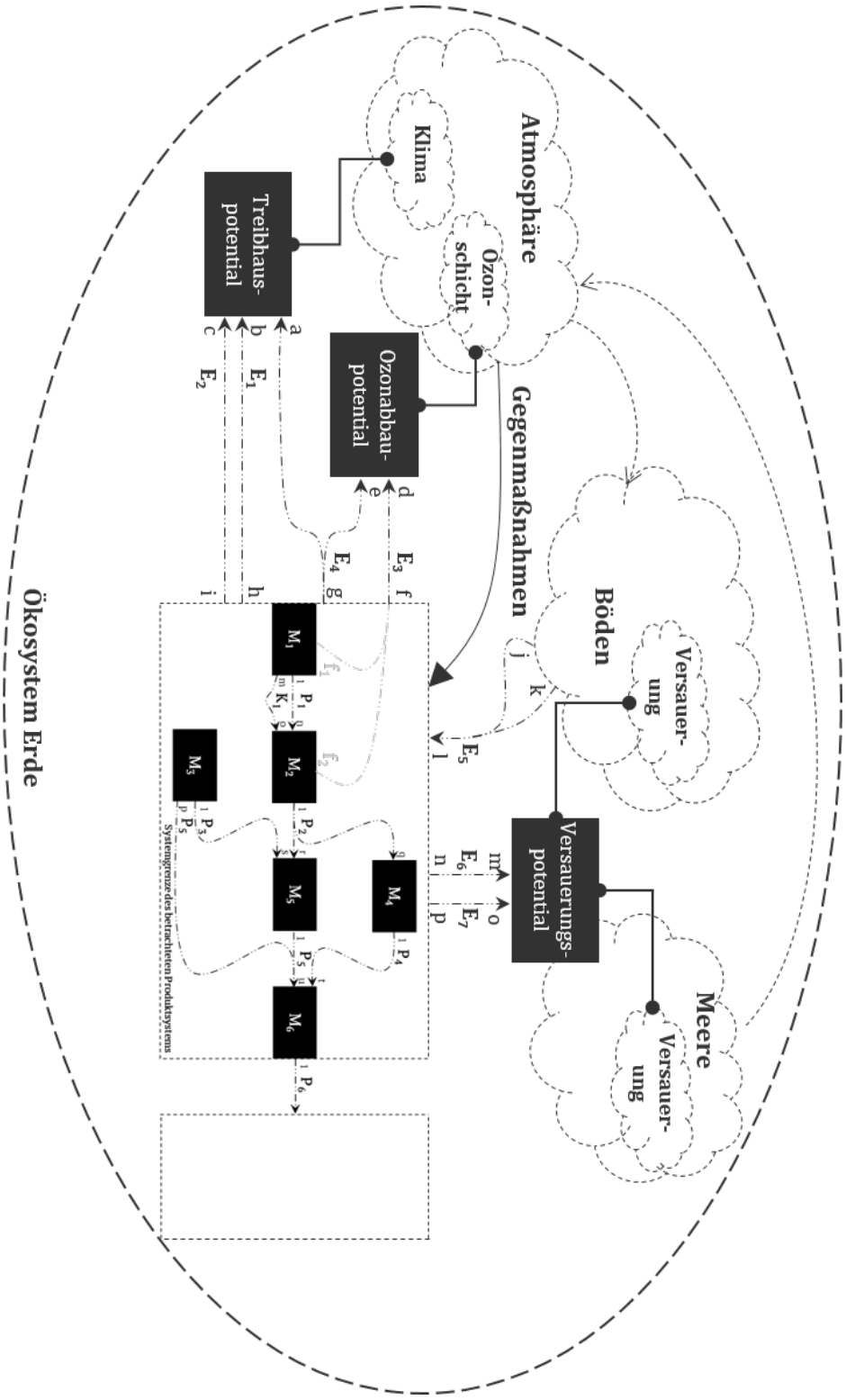


Abbildung 7.16: Produktsysteme als Untergraphen übergeordneter Graphen.

Zwischenprodukt eines Prozessmoduls wird durch genau ein Prozessmodul hergestellt, weshalb je Zeile der Produktmatrix genau ein positives Punktintervall definiert ist.

$$\mathrm{GS}_{\text{lin, erz.}} = \begin{bmatrix} a_{11} & -a_{12}^{\mathrm{I}} & 0 & \cdots & 0 & 0 \\ 0 & a_{22} & -a_{23}^{\mathrm{I}} & \cdots & 0 & 0 \\ 0 & 0 & a_{33} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{(n-1)(n-1)} & -a_{(n-1)n}^{\mathrm{I}} \\ 0 & 0 & 0 & \cdots & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.17: Gleichungssystem eines *erzeugenden*, linearen Produktsystems mit n Prozessmodulen.

In Produktmatrizen linearer Produktsysteme ist oberhalb der Hauptdiagonalen, nach entsprechender Umformung, zeilenweise genau ein negativer Input definiert, der direkt rechts neben dem Hauptdiagonalelement zu finden ist. Die Determinante von Produktmatrizen rein linearer Produktsysteme ergibt sich dann durch Multiplikation der Diagonalelemente a_{11} bis a_{nn} (bei entsprechender Darstellung als Dreiecksmatrix). In der n-ten Zeile ist das zu erzielende Endprodukt des Produktsystems hinterlegt, welches als Output das Produktsystem verlässt.

Produktmatrizen linearer, verarbeitender Produktsysteme

Zur Erläuterung ist zusätzlich und stellvertretend für alle übrigen Produktsystemstrukturen in Abbildung 7.18 das Gleichungssystem $\mathrm{GS}_{lin,verarb.}$ eines *verarbeitenden* Produktsystems dargestellt. Die Produktmatrix ist angelehnt an das in Kapitel 7.2.1, Abbildung 7.4 vorgestellte, verarbeitende Produktsystem.

$$\mathrm{GS}_{\text{lin, verarb.}} = \begin{bmatrix} -a_{11} & a_{12}^{\mathrm{I}} & 0 & \cdots & 0 & 0 \\ 0 & -a_{22} & a_{23}^{\mathrm{I}} & \cdots & 0 & 0 \\ 0 & a_{12}^{\mathrm{I}} & -a_{33} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -a_{(n-1)(n-1)} & a_{(n-1)n}^{\mathrm{I}} \\ 0 & 0 & 0 & \cdots & 0 & -a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} -f_1 \\ -f_2 \\ -f_3 \\ \vdots \\ -f_{n-1} \\ -f_n \end{bmatrix}$$

Abbildung 7.18: Gleichungssystem eines *verarbeitenden*, linearen Produktsystems mit n Prozessmodulen.

Dabei bezieht sich die Produktmatrix auf ein Produktsystem mit n Prozessmodulen. Die Produktmatrix ist derart geordnet, dass in der ersten Zeile der Produktfluss des Produkts P_1 dargestellt ist und in der ersten Spalte das zugehörige, P_1 verarbeitende Prozessmodul M_1. Eine Multiplikation des Gleichungssystems mit *-1* zeigt, dass $\mathrm{PS}_{lin,verarb.} = \mathrm{PS}_{lin,erz.}$.

7.5.3 Produktmatrizen baumartiger Produktsysteme

Die Produktmatrizen baumartiger Produktsysteme lassen sich stets, mitunter durch Matrixumformungen, als obere Dreiecksmatrix darstellen. Die Determinante ergibt sich dann durch Multiplikation der Diagonalelemente a_{11} bis a_{nn}. Baumartige Produktsysteme sind koppelproduktfrei, d. h. das Zwischenprodukt eines Prozessmoduls wird durch genau ein Prozessmodul hergestellt, weshalb je Zeile der Produktmatrix ein positives Punktintervall definiert ist.

Produktmatrizen gegenströmiger Bäume

$$\mathrm{GS}_{\text{baum, gegenstr.}} = \begin{bmatrix} a_{11} & & 0 & \cdots & 0 & -a^{\mathrm{I}}_{1n} \\ 0 & a_{22} & & \cdots & 0 & -a^{\mathrm{I}}_{2n} \\ 0 & 0 & a_{33} & \cdots & 0 & -a^{\mathrm{I}}_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{(n-1)(n-1)} & -a^{\mathrm{I}}_{(n-1)n} \\ 0 & 0 & 0 & \cdots & 0 & a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.19: Gleichungssystem eines gegenströmigen Baums mit n Prozessmodulen.

Das in Abbildung 7.19 dargestellte Gleichungssystem $\mathrm{GS}_{baum,gegenstr.}$ repräsentiert ein Produktsystem mit der Struktur eines gegenströmigen Baums mit n Prozessmodulen, wobei nach entsprechender Umformung die Hauptprodukte als Hauptdiagonalelemente a_{ii} mit i von *1* bis n in Form von positiven Punktintervallen hinterlegt sind. In solchen Produktmatrizen ist oberhalb der Hauptdiagonalen zeilenweise genau ein negativer Input definiert, der in der rechten Dreiecksmatrix zu finden ist. Die Inputs befinden sich in derselben Spalte wie der Output der Wurzel.

Produktmatrizen strömungskonformer Bäume

Das in Abbildung 7.20 dargestellte Gleichungssystem $\mathrm{GS}_{baum,strömungskonf.}$ repräsentiert einen strömungskonformen Baum mit n Prozessmodulen, wobei die Hauptprodukte der „Blätter“ in Form von *negativen* Punktintervallen auf der Hauptdiagonalen hinterlegt sind.

$$\mathrm{GS}_{\text{baum, strömungskonf.}} = \begin{bmatrix} -a_{11} & & 0 & \cdots & a^{\mathrm{I}}_{1(n-1)} & 0 \\ 0 & -a_{22} & 0 & \cdots & a^{\mathrm{I}}_{2(n-1)} & 0 \\ 0 & 0 & -a_{33} & \cdots & a^{\mathrm{I}}_{3(n-1)} & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{(n-1)(n-1)} & -a^{\mathrm{I}}_{(n-1)n} \\ 0 & 0 & 0 & \cdots & 0 & a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.20: Gleichungssystem eines *strömungskonformen* Baums mit n Prozessmodulen.

In solchen Produktmatrizen ist oberhalb der Hauptdiagonalen genau ein positiver Output in der rechten Dreiecksmatrix definiert. Die Wurzel ist dasjenige Prozessmodul, in dessen zugehöriger Spalte nur positive Werte zu finden sind. Die Determinante von Produktmatrizen rein baumartiger Produktsysteme ergibt sich dann durch Multiplikation der Diagonalelemente a_{11} bis a_{nn}.

7.5.4 Produktmatrizen pseudo-baumartiger Produktsysteme

Die Produktmatrizen pseudo-baumartiger Produktsysteme lassen sich stets, mitunter durch Matrixumformungen, als obere Dreiecksmatrix darstellen. Die Determinante ergibt sich dann durch Multiplikation der Diagonalelemente a_{11} bis a_{nn}. Das in Abbildung 7.21 dargestellte Gleichungssystem $\mathrm{GS}_{pseudo-baum}$ repräsentiert ein pseudo-baumartiges Produktsystem mit n Prozessmodulen, wobei die Hauptprodukte in Form von positiven Punktintervallen auf der Hauptdiagonalen hinterlegt sind.

$$\mathrm{GS}_{\text{pseudo-baum}} = \begin{bmatrix} a_{11} & -a^{\mathrm{I}}_{12} & -a^{\mathrm{I}}_{13} & \cdots & -a^{\mathrm{I}}_{1(n-1)} & -a^{\mathrm{I}}_{1n} \\ 0 & a_{22} & 0 & \cdots & 0 & 0 \\ 0 & 0 & a_{33} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{(n-1)(n-1)} & 0 \\ 0 & 0 & 0 & \cdots & 0 & a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.21: Gleichungssystem eines pseudo-baumartigen Produktsystems mit n Prozessmodulen.

Pseudo-baumartige Produktsysteme sind koppelproduktfrei, d. h. das Zwischenprodukt eines Prozessmoduls wird durch genau ein Prozessmodul hergestellt, weshalb je Zeile der Produktmatrix genau ein positives Punktintervall definiert ist. In solchen Produktmatrizen können oberhalb der Hauptdiagonalen zeilenweise mehrere negative Inputs definiert sein. Die Inputs von pseudo-baumartigen Systemen ohne interne Pseudo-Wurzelknoten finden sich allesamt in derselben Zeile wie der Output der Pseudo-Wurzel.

7.5.5 Produktmatrizen pseudo-schleifenartiger Produktsysteme

Die Produktmatrizen pseudo-schleifenartiger Hyper-Produktsysteme lassen sich nicht, auch nicht durch Matrixumformungen, als obere Dreiecksmatrix darstellen. Bei Produktmatrizen dieser-Produktsysteme ist daher zu erwarten, dass mehrere Permutationen der Leibniz-Formel ungleich null sind. Pseudo-schleifenartige Produktsysteme sind nicht koppelproduktfrei, d. h. es werden Zwischenprodukte einzelner Prozessmodule durch mehr als ein Prozessmodul hergestellt. Die Nichtnullelemente unterhalb der Hauptdiagonalen stellen, nach entsprechender Umformung, die Outputs der entstandenen Koppelprodukte dar. In solchen Produktmatrizen ist oberhalb der Hauptdiagonalen in der rechten Dreiecksmatrix zeilenweise genau ein negativer Input definiert. Die Matrix oberhalb der Diagonalen ist weiterhin geprägt von der Grundstruktur des Produktsystems, die sich ohne Koppelprodukt ergeben würde.

Pseudo-Schleifen mit linearer Grundstruktur

In Abbildung 7.22 ist das Gleichungssystem $\mathrm{GS}_{pseudo-schleife,lin}$ abgebildet, das ein pseudo-schleifenartiges Produktsystem mit linearer Grundstruktur sowie n Prozessmodulen repräsentiert.

$$\mathrm{GS}_{\text{pseudo-schleife, lin}} = \begin{bmatrix} a_{11} & -a^{\mathrm{I}}_{12} & 0 & \cdots & 0 & 0 \\ 0 & a_{22} & -a^{\mathrm{I}}_{23} & \cdots & 0 & 0 \\ 0 & 0 & a_{33} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a^{\mathrm{I}}_{(n-1)1} & 0 & 0 & \cdots & a_{(n-1)(n-1)} & -a^{\mathrm{I}}_{(n-1)n} \\ 0 & 0 & 0 & \cdots & 0 & a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.22: Gleichungssystem eines pseudo-schleifenartigen Produktsystems mit linearer Grundstruktur.

Die Hauptprodukte sind, nach entsprechender Umformung, in Form von positiven Punktintervallen auf der Hauptdiagonalen hinterlegt. Das Prozessmodul M_1 wird durch die erste Spalte von $\mathrm{GS}_{\text{pseudo-schleife, lin}}$ repräsentiert. M_1 erzeugt ein Koppelprodukt in Form des Produkts P_{n-1} und der Menge $a^{\mathrm{I}}_{(n-1)1}$; (vgl. *n-1*-te Zeile).

Pseudo-Schleifen mit baumartiger Grundstruktur

In Abbildung 7.23 ist das Gleichungssystem $\mathrm{GS}_{pseudo-schleife,baum}$ abgebildet, das ein pseudo-schleifenartiges Produktsystem mit baumartiger Grundstruktur sowie n Prozessmodulen repräsentiert.

$$\mathrm{GS}_{\text{pseudo-schleife, baum}} = \begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 & -a^{\mathrm{I}}_{1n} \\ 0 & a_{22} & 0 & \cdots & 0 & -a^{\mathrm{I}}_{2n} \\ 0 & 0 & a_{33} & \cdots & 0 & -a^{\mathrm{I}}_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a^{\mathrm{I}}_{(n-1)1} & 0 & 0 & \cdots & a_{(n-1)(n-1)} & -a^{\mathrm{I}}_{(n-1)n} \\ 0 & 0 & 0 & \cdots & 0 & a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.23: Gleichungssystem eines pseudo-schleifenartigen Produktsystems, baumartige Grundstruktur.

Die Hauptprodukte sind, nach entsprechender Umformung, in Form von positiven Punktintervallen auf der Hauptdiagonalen hinterlegt. Das Prozessmodul M_1 erzeugt dabei ein Koppelprodukt in Form der Produkts P_{n-1}. Das Prozessmodul M_1 wird durch die erste Spalte von $\mathrm{GS}_{\text{pseudo-schleife, baum}}$ repräsentiert. M_1 erzeugt ein Koppelprodukt in Form des Produkts P_{n-1} und der Menge $a^{\mathrm{I}}_{(n-1)1}$; es ist in der *n-1*-ten Zeile hinterlegt.

7.5.6 Produktmatrizen schleifenartiger Produktsysteme durch Zusammenfluss

Die Produktmatrizen schleifenartiger Produktsysteme durch Zusammenfluss lassen sich stets, mitunter durch Matrixumformungen, als obere Dreiecksmatrix darstellen. Die Determinante ergibt sich dann durch Multiplikation der Diagonalelemente a_{11} bis a_{nn}. Schleifenartige Produktsysteme durch Zusammenfluss sind nicht koppelproduktfrei, d. h. es werden Zwischenprodukte einzelner Prozessmodule durch mehr als ein Prozessmodul hergestellt. Die positiven Nichtnullelemente oberhalb der Hauptdiagonalen entsprechen, nach entsprechender Umformung, den Outputs der entstandenen Koppelprodukte. In solchen Produktmatrizen ist oberhalb der Hauptdiagonalen in der rechten Dreiecksmatrix zeilenweise genau ein negativer Input definiert. Die Anordnung der Matrixelemente oberhalb der Diagonalen ist weiterhin geprägt von der Grundstruktur des Produktsystems, welche sich ohne Koppelprodukt ergeben würde.

$$\mathrm{GS}_{\text{schleife-zusammenfluss}} = \begin{bmatrix} a_{11} & -a^{\mathrm{I}}_{12} & 0 & \cdots & a^{\mathrm{I}}_{1(n-1)} & 0 \\ 0 & a_{22} & -a^{\mathrm{I}}_{23} & \cdots & 0 & 0 \\ 0 & 0 & a_{33} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{(n-1)(n-1)} & -a^{\mathrm{I}}_{(n-1)n} \\ 0 & 0 & 0 & \cdots & 0 & a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.24: Gleichungssystem eines schleifenartigen Produktsystems durch Zusammenfluss.

In Abbildung 7.24 ist das Gleichungssystem $\mathrm{GS}_{schleife-zusammenfluss}$ abgebildet, das ein schleifenartiges Produktsystem durch Zusammenfluss mit n Prozessmodulen repräsentiert. Die Hauptprodukte sind in Form von Punktintervallen auf der Hauptdiagonalen hinterlegt. Hauptprodukte, die in Rezyklierprozessen aufbereitet werden, weisen ein *negatives* Punktintervall auf der Hauptdiagonalen auf; Hauptprodukte aus erzeugenden Prozessmodulen hingegen ein *positives* Punktintervall. Das Prozessmodul M_{n-1} wird durch die *n-1*-te Spalte in Abbildung

7.24 repräsentiert. M_{n-1} erzeugt ein Koppelprodukt in Form des Produkts P_1 und der Menge $a^{I}_{1(n-1)}$; es ist in der ersten Zeile hinterlegt. M_{n-1} entspricht dem Prozessmodul M_3 desjenigen Produktsystems, welches in Abbildung 7.13 dargestellt ist.

7.5.7 Produktmatrizen schleifenartiger Produktsysteme durch Verzweigung

Die Produktmatrizen schleifenartiger Produktsysteme durch Verzweigung lassen sich nicht, auch nicht durch Matrixumformungen, als obere Dreiecksmatrix darstellen. Bei Produktmatrizen dieser Produktsysteme ist daher zu erwarten, dass mehrere Permutationen der Leibniz-Formel ungleich null sind. Schleifenartige Produktsysteme durch Verzweigung sind koppelproduktfrei, d. h. das Zwischenprodukt eines Prozessmoduls wird durch genau ein Prozessmodul hergestellt, weshalb je Zeile der Produktmatrix genau ein positives Punktintervall definiert ist. In schleifenartigen Produktsystemen durch Verzweigung werden Zwischenprodukte einzelner Prozessmodule in nachfolgenden sowie *vorangehenden* Prozessmodulen benötigt. Die Nichtnullelemente unterhalb der Hauptdiagonalen stellen, nach entsprechender Umformung, als Inputs die Ursache der Kantenverzweigung dar.

$$\mathrm{GS}_{\text{schleife-verzw.}} = \begin{bmatrix} a_{11} & -a^{I}_{12} & 0 & \cdots & 0 & a^{I}_{1n} \\ 0 & a_{22} & -a^{I}_{23} & \cdots & 0 & a^{I}_{2n} \\ 0 & 0 & a_{33} & \cdots & 0 & a^{I}_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & -a^{I}_{(n-1)2} & 0 & \cdots & a_{(n-1)(n-1)} & -a^{I}_{(n-1)n} \\ 0 & 0 & 0 & \cdots & 0 & a_{(nn)} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

Abbildung 7.25: Gleichungssystem eines schleifenartigen Produktsystems durch Verzweigung.

In solchen Produktmatrizen ist oberhalb der Hauptdiagonalen in der rechten Dreiecksmatrix zeilenweise genau ein negativer Input definiert. Die Matrix oberhalb der Diagonalen ist weiterhin geprägt von der Grundstruktur des Produktsystems, die sich ohne Hyperkante ergeben würde. In Abbildung 7.25 ist das Gleichungssystem $\mathrm{GS}_{schleife-verzw.}$ abgebildet, das ein schleifenartiges Produktsystem mit linearer Grundstruktur sowie n Prozessmodulen repräsentiert. Die Hauptprodukte sind in Form von positiven Punktintervallen auf der Hauptdiagonalen hinterlegt. Das Prozessmodul M_2 wird durch die zweite Spalte von $\mathrm{GS}_{\text{schleife-verzw.}}$ repräsentiert. M_2 benötigt ein Zwischenprodukt in Form des Produkts P_{n-1} und der Menge $a^{I}_{(n-1)2}$; es ist in der *n-1*-ten Zeile hinterlegt. M_{n-1} entspricht dem Prozessmodul M_3 desjenigen Produktsystems, welches in Abbildung 7.14 dargestellt ist.

7.6 Klassifikation der Produktmatrizen

Auf Basis der Produktmatrizen kann mitunter, ggfs. nach entsprechender Umformung) die zugrunde gelegte Produktsystemstruktur abgeleitet werden.

So können anhand der Produktmatrix folgende Rückschlüsse gezogen werden:

- Produktmatrizen, die sich zu einer Dreiecksmatrix umformen lassen und deren Determinante einem Punktintervall entspricht, sind in der Regel einfache Produktsysteme mit linearer oder baumartiger Produktsystemstruktur sowie schleifenartige Produktsysteme durch Zusammenfluss.
- Positive Elemente unterhalb der Hauptdiagonalen in der unteren Dreiecksmatrix deuten auf Pseudo-Schleifen hin.

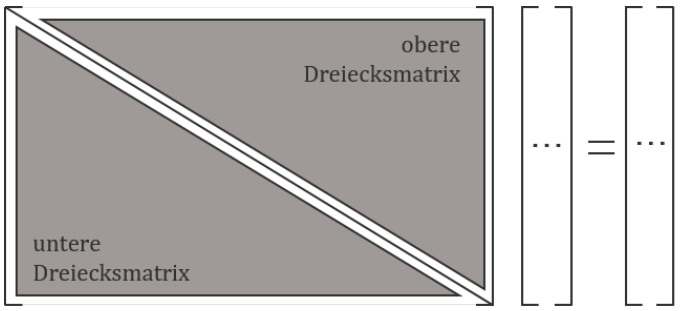


Abbildung 7.26: Produktmatrix eines Gleichungssystems, unterteilt in untere und obere Dreiecksmatrix.

- Negative Elemente unterhalb der Hauptdiagonalen in der unteren Dreiecksmatrix lassen Schleifen durch Verzweigung vermuten.

- Mehrfach negative Werte in einer Zeile oberhalb der Hauptdiagonalen in der oberen Dreiecksmatrix deuten auf pseudo-baumartige Produktsysteme hin.

- Positive Elemente oberhalb der Hauptdiagonalen in der oberen Dreiecksmatrix liegen in der Regel bei Schleifen durch Zusammenfluss vor. Davon ausgenommen sind positive Elemente, die links vom negativen Input angeordnet sind, in diesem Fall handelt es sich um ein baumartiges Produktsystem mit Pseudoschleife.

Die Produktmatrizen können mitunter *M*-, *H*- oder *Z*-Matrizen zugeordnet werden (Definition der Matrixtypen vgl. auch Kapitel 2.1).

7.6.1 Einordnung linearer und (pseudo-)baumartiger Produktmatrizen

Produktmatrizen linearer und (pseudo-)baumartiger Produktsysteme können derart umgeformt werden, dass sie nur auf der Hauptdiagonalen positive Punktintervalle aufweisen. Die Produktmatrizen linearer, baumartiger und pseudo-baumartiger Produktsysteme entsprechen daher den Anforderungen einer Z-Matrix. Die Vergleichsmatrix $\langle \mathrm{A}^{\mathrm{I}} \rangle \in \mathbb{R}^{nxn}$ einer Produktmatrix $\mathrm{A}^{\mathrm{I}} \in \mathbb{IR}^{nxn}$ eines linearen, baumartigen oder pseudo-baumartigen Produktsystems ist identisch den Mignituden $\langle \mathrm{A}_{ij} \rangle$ (Definition von Mignitude vgl. auch Kapitel 3.3, Abschnitt „Mignitude eines Intervalls“) der Matrixelemente ij von der Intervallmatrix A^{I}, womit gilt:

$$\langle \mathrm{A}^{\mathrm{I}} \rangle = \langle \mathrm{A}_{ij} \rangle \in \mathrm{A}^{\mathrm{I}}.$$

Damit stellt $\langle \mathrm{A}_{ij} \rangle \in \mathrm{A}^{\mathrm{I}}$ auch die Vergleichsmatrix von $\langle \mathrm{A}^{\mathrm{I}} \rangle$ dar:

$$\langle \langle \mathrm{A}^{\mathrm{I}} \rangle \rangle = \langle \mathrm{A}^{\mathrm{I}} \rangle = \langle \mathrm{A}_{ij} \rangle \in \mathrm{A}^{\mathrm{I}}.$$

Die Produktmatrix A^{I} bzw. deren Vergleichsmatrix $\langle \mathrm{A}^{\mathrm{I}} \rangle$ kann zu einer oberen Dreiecksmatrix umgeformt werden, deren Hauptdiagonale ausschließlich positive Nichtnullelemente enthält. Dies hat zur Folge, dass die Produktmatrix A^{I} eines linearen oder (pseudo-)baumartigen Produktsystems bzw. dessen Vergleichsmatrix $\langle \mathrm{A}^{\mathrm{I}} \rangle$ die Anforderungen einer M-Matrix erfüllt und somit invers-positiv ist - gleichsam können beide Matrizen A^{I} und $\langle \mathrm{A}^{\mathrm{I}} \rangle$ auch als H-Matrix deklariert werden.

7.6.2 Einordnung (pseudo-)schleifenartiger Produktmatrizen

Produktmatrizen pseudo-schleifenartiger Produktsysteme sowie schleifenartiger Produktsysteme durch Zusammenfluss treten, nach entsprechender Umformung, aufgrund der anfallenden Koppelprodukte auch neben der Hauptdiagonalen positive Nichtnullelemente auf. Die Produktmatrizen pseudo-schleifenartiger Produktsysteme und die von schleifenartigen Produktsystemen durch Zusammenfluss entsprechen daher nicht den Anforderungen einer Z-Matrix. Sie können zwar invers-positiv sein, stellen jedoch keine M-Matrizen dar. Produktmatrizen schleifenartiger Produktsysteme durch Verzweigung treten, nach entsprechender Umformung, neben der Hauptdiagonalen keine positiven Nichtnullelemente auf. Produktmatrizen schleifenartiger Produktsysteme durch Verzweigung entsprechen daher den Anforderungen einer Z-Matrix.

Die Vergleichsmatrix $\langle A^I \rangle \in \mathbb{R}^{nxn}$ einer Produktmatrix $A^I \in \mathbb{IR}^{nxn}$ von schleifenartigen Produktsystemen durch Zusammenfluss entspricht einer M-Matrix, womit die Produktmatrix A^I dann den H-Matrizen zugeordnet werden kann (vgl. auch Kapitel 3.5.7). Die Vergleichsmatrix $\langle A^I \rangle$ einer Produktmatrix $A^I \in \mathbb{IR}^{nxn}$ von Pseudo-Schleifen oder von schleifenartigen Produktsystemen durch Verzweigung kann invers-positiv sein - sie ist dies jedoch nicht in jedem Fall. Insbesondere wenn die Beträge der Nichtnullelemente unterhalb der Hauptdiagonalen groß sind, kann die Inverse negative Elemente aufweisen. Dies bedeutet, dass bei pseudo-schleifenartigen Produktsystemen wie auch bei schleifenartigen Produktsystemen durch Verzweigung nicht gewährleistet ist, dass deren Produktmatrix A^I den H-Matrizen zugeordnet werden kann.

7.7 Bei der Modellierung zu beachtende Besonderheiten

Bei der grafischen Darstellung von Produktsystemen sowie bei der Berechnung dieser gibt es Besonderheiten, die es zu beachten gilt. Dies betrifft zum einen die Definition der funktionellen Einheit, welche in Form des Referenzflusses quantifiziert wird. Der Referenzfluss verlässt herstellende Produktsysteme als Output und fließt als Input in verarbeitende Produktsysteme (vgl. Kapitel 7.7.1). Dies spielt auch bei der Modellierung des gesamten Lebensweges eine Rolle (vgl. Kapitel 7.7.2). Werden Zusatzprodukte erzeugt, erhält das Produktsystem mitunter einen weiteren Nutzen, der ebenfalls zu berücksichtigen ist (vgl. Kapitel 7.7.3). Durch die Datenerhebung der Sachbilanz ändert sich der Graphentyp: dieser ist nun auch gewichtet (vgl. Kapitel 7.7.4). Insgesamt ergeben sich diverse Anforderungen an die Modellierung, die in Kapitel 7.7.5 aufgelistet werden.

7.7.1 Herstellende und verarbeitende Produktsysteme

Ein Produktsystem, dessen Nutzen es ist, ein Produkt zu erzeugen, weist in der Regel als Referenzfluss und funktionelle Bezugseinheit einen Output auf; ein Produktsystem, dessen Zweck es ist, ein Produkt zu verarbeiten, weist in der Regel als Referenzfluss und funktionelle Bezugseinheit einen Input auf. In Abbildung 7.27 ist zur Veranschaulichung ein Produktsystem mit drei Prozessmodulen und Produktflüssen dargestellt, durch welches ein Produkt P_3 erzeugt wird und die Grenze des Produktsystems, dargestellt als gestrichelte Linie, überschreitet. Analog ist in Abbildung 7.28 zur Veranschaulichung ein Produktsystem mit drei Prozessmodulen und Produktflüssen dargestellt, durch welches das Produkt P_0 verarbeitet wird.

In der grafischen Darstellung unterscheiden sich herstellende Produktsysteme von verarbeitenden Produktsystemen durch einen Wechsel der Pfeilrichtung - in der tabellarischen Darstellung des Produktsystems als Produktmatrix zeigt sich dies durch die Umkehrung des Vorzeichens.

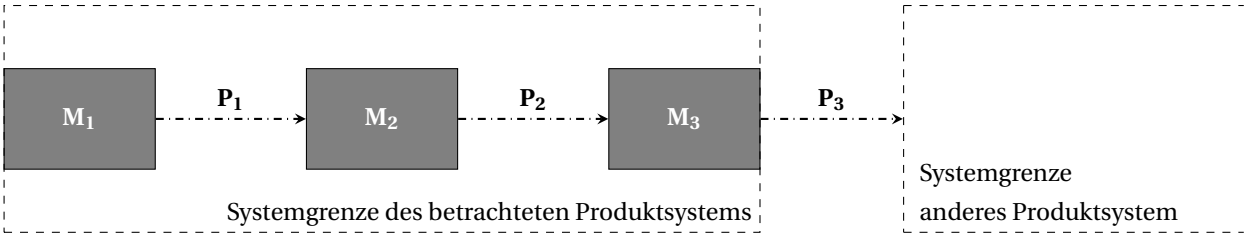


Abbildung 7.27: Produktsystem mit drei Prozessmodulen, durch welches das Produkt P_3 hergestellt wird.

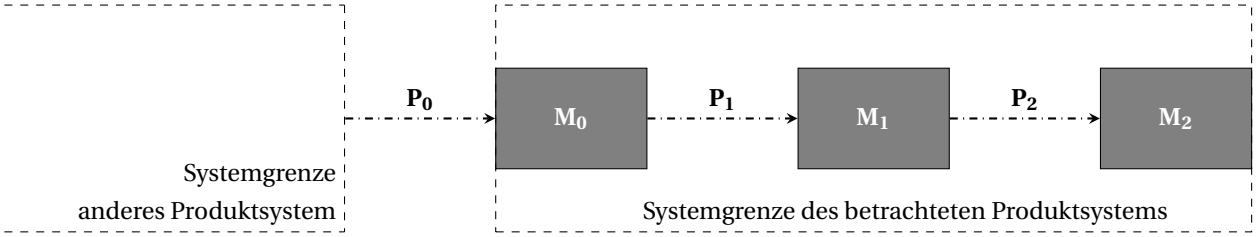


Abbildung 7.28: Produktsystem mit drei Prozessmodulen, durch welches das Produkt P_0 verarbeitet wird.

7.7.2 Betrachtung des gesamten Lebensweges

Wenn Produktsysteme Prozesse beinhalten, durch welches ein Produkt erzeugt wird *und* zugleich Prozesse, durch welche das erzeugte Produkt wieder vollständig behandelt wird, weisen die entsprechenden Produktsysteme kein Endprodukt auf, welches die Systemgrenze überschreitet. Dies kann der Fall sein, wenn der gesamte Lebensweg eines Produkts berücksichtigt wird. Zusätzlich kann es vorkommen, dass die durch die Hauptprodukte gebildete Produktsystemstruktur einem geschlossenen Zyklus entspricht. Dies kommt vor, wenn Systeme mit geschlossenen Stoff- oder Energiekreisläufen modelliert werden, was politisch und gesellschaftlich zunehmend angestrebt wird. Auch diese Produktsysteme weisen dann bei vollständiger Rückführung in das System kein Endprodukt auf, welches die Systemgrenze überschreitet.

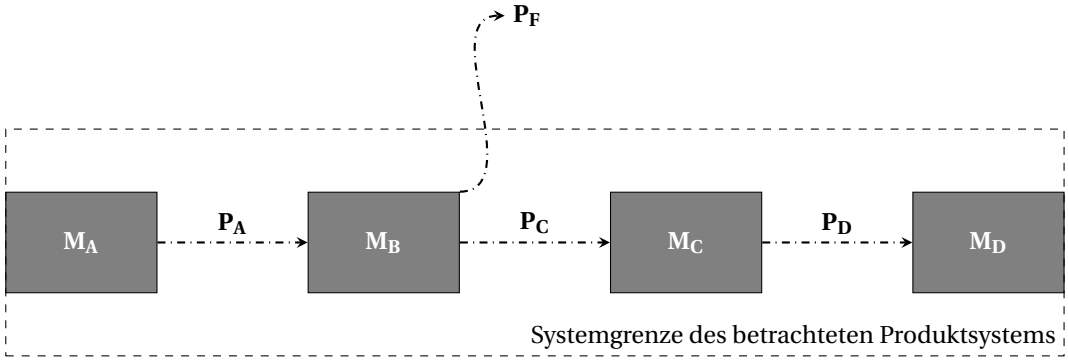


Abbildung 7.29: Lebensweg eines Produkts, funktionelle Einheit ist Nutzung von P_F.

Zur Berechenbarkeit des Systems spielt jedoch die funktionelle Einheit bzw. der Bedarf als „Output auf der rechten Seite“ eine entscheidende Rolle. Einem Produktsystem, welches einzig das Informationsmodul *A* abbildet, wird als funktionelle Einheit im Wesentlichen die *Erzeugung* eines quantifizierten Produkts zugeordnet (Defini-

tion „Informationsmodul “ vgl. Kapitel 5.2.19). Am Ende des Herstellungsprozesses überschreitet das Produkt als Output die Systemgrenze. Dieses Produkt ist der „Bedarf“, welcher im Bedarfsvektor als positiver Wert quantifiziert wird und somit ungleich null ist.

Einem Produktsystem, welches einzig das Informationsmodul *C* abbildet, wird als funktionelle Einheit im Wesentlichen die *Verwertung bzw. Beseitigung* von Abfall zugeordnet. Der Abfall entsteht außerhalb des Produktsystems und gelangt als *Input* in das Produktsystem. Die Verarbeitung dieses Produkts ist der Zweck des Prozessmoduls. Da hierbei eine Verarbeitung erzielt werden soll und es sich nicht um die Herstellung eines Produkts handelt, weist der entsprechende Wert im Bedarfsvektor ein negatives Vorzeichen auf. Einem Produktsystem, welches einzig das Informationsmodul *D* abbildet, wird als funktionelle Einheit im Wesentlichen die *Substitution* eines quantifizierten Zusatzprodukts zugeordnet. Das Zusatzprodukt entsteht außerhalb des Produktsystems und gelangt als *Input* in das Produktsystem. Die Substitution dieses Produkts ist der „Bedarf“, welcher im Bedarfsvektor ebenfalls als negativer Wert quantifiziert wird und somit ungleich null ist.

Einem Produktsystem, welches einzig das Informationsmodul *B* abbildet, wird als funktionelle Einheit im Wesentlichen die *Nutzung* des abgebildeten Produkts zugeordnet. Der Referenzfluss überschreitet als *Output* die Systemgrenze. Dieser unterscheidet sich jedoch von den Referenzflüssen der übrigen Informationsmodule. Wird z. B. ein physisches Produkt im Bauwesen modelliert, stellen die Referenzflüsse der Informationsmodule *A*, *C* und *D* physische Vorgänge dar, bei denen das Produkt stofflich erzeugt, aufbereitet oder beseitigt wird. Im Gegensatz hierzu stellt der Referenzfluss des Informationsmoduls B im Allgemeinen vielmehr abstrakt die Nutzung des Produkts dar. Wird der gesamte Lebenszyklus mit allen Informationsmodulen *A* bis *D* betrachtet, ist der Zweck des betrachteten Gesamtsystems die Nutzung des Produkts außerhalb des Produktsystems und wird im Produktsystem in Form eines abstrakten Produktflusses als *Output* in Modul B modelliert (vgl. Abbildung 7.29). Insgesamt sind alle übrigen Produktflüsse Zwischenproduktflüsse, die durch das folgende Prozessmodul verarbeitet werden: das hergestellte Produkt aus Modul *A* wird zur Nutzung benötigt, während das verwendete Produkt nach der Nutzung in Modul *C* behandelt wird; etwaige Zusatzprodukte daraus können in Modul *D* substituiert werden.

7.7.3 Modellierung von Zusatzprodukten

Beim Auftreten von Zusatzprodukten ist die vollständige Modellierung dieser zu beachten. Relevante Methoden hierzu sind in Kapitel 5.3.5 aufgeführt. Folgende Szenarien sind beim Auftreten von Zusatzprodukten denkbar:

- Das Zusatzprodukt wird gemäß der Abfalleigenschaft als Abfall eingeordnet. Es muss in einem Prozess endgültig entsorgt werden. Dafür wird ein Prozessmodul im Produktsystem modelliert, welches die Abfallbeseitigung repräsentiert.

- Das Zusatzprodukt erfüllt die Anforderungen zum Ende der Abfalleigenschaft und gilt nach geeigneten Behandlungsprozessen nicht mehr als Abfall. Im Produktsystem sollten die entsprechenden Behandlungsprozesse modelliert werden. Die (zurück)gewonnenen Produkte können intern oder extern verwendet werden.

- Das Zusatzprodukt kann weitgehend direkt innerhalb des Produktsystems verwendet werden und stellt daher ein internes Zusatzprodukt dar. Deckt das interne Zusatzprodukt nicht vollständig den Bedarf, ist die Modellierung eines weiteren Prozessmoduls notwendig, welches das Produkt als Hauptprodukt herstellt. Durch die Bedarfsdeckung eines Inputs von mehreren Prozessmodulen des Produktsystems entstehen Wechselbeziehungen. Diese werden durch zusammenfließende Hyperkanten dargestellt. Da die Kanten dasselbe Produkt mit identischer Funktion repräsentieren und in einer abhängigen Wechselbeziehung zueinander stehen, können sie nicht durch mehrere Kanten, d. h. als unabhängige Flüsse dargestellt werden.

Deckt das interne Koppelprodukt vollständig den Bedarf, kann auf das Prozessmodul verzichtet werden - in diesem Fall wird der Zusatzproduktfluss bei der Kalkulation der Sachbilanz nicht berücksichtigt. Etwaige Überschüsse, welche im Produktsystem keine Verwendung haben, werden den externen Zusatzprodukten zugeordnet. Sie unterscheiden sich in ihrer Funktion von den intern verwendeten Zusatzprodukten und werden daher gesondert dargestellt.

- Das Zusatzprodukt hat intern keine Verwendung, stellt aber einen Nutzen für externe Produktsysteme dar und wird daher als externes Zusatzprodukt betrachtet. Das Zusatzprodukt kann durch Allokation oder Systemerweiterung mittels eines Prozessmoduls, welches die Gutschrift repräsentiert, substituiert werden.

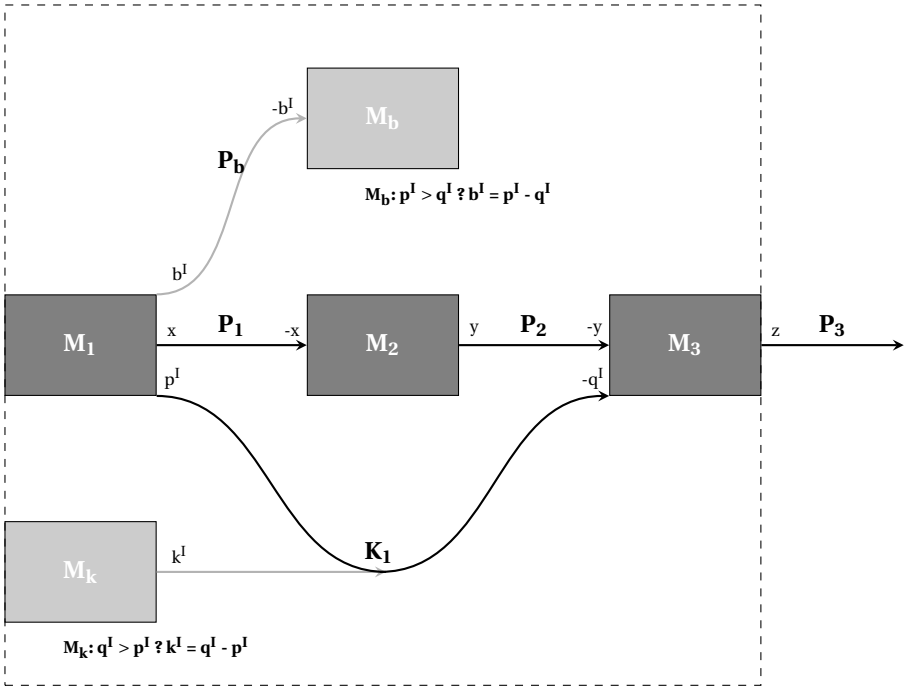


Abbildung 7.30: Produktsystem mit fünf Prozessmodulen und vollständiger Modellierung des Koppelprodukts K_1.

In Abbildung 7.30 ist ein Produktsystem dargestellt, indem das interne Koppelprodukt K_1 entsteht. Es wird mit der Menge p^I im Prozessmodul M_1 neben dem Hauptprodukt P_1 produziert. P_1 wird im Prozessmodul M_2 mit einer Menge von x benötigt, dessen Output wiederum M_3 versorgt, während K_1 in Prozessmodul M_3 mit einer Menge von q^I benötigt wird. Daraus folgt eine doppelte Abhängigkeit zwischen den beiden Prozessmodulen M_1 und M_3. Das Produktsystem ist ohne das Prozessmodul M_k funktionsfähig, wenn $p^I - q^I = 0$ gilt, was zugleich bedeutet, dass p und q identische Werte mit Intervallweiten von null aufweisen müssen. Bei der Erzeugung als auch der Verwendung von Koppelprodukten sind Intervallweiten ungleich null möglich und gemäß der Modellierungskriterien zulässig („Modellierungskriterien“ vgl. auch Kapitel 8.4.3). Damit das Gesamtsystem auch bei Intervallweiten ungleich null lösbar bleibt, ist in den Modellierungskriterien festgelegt, dass in diesem Fall das Produktsystem um ein Prozessmodul zur Produktion von K_1 als Hauptprodukt ergänzt wird *oder* auftretende Überschüsse zu modellieren sind - z. B. als Gutschrift (vgl. „Gutschriftenmethode“ in Kapitel 5.3.5).

Beide Fälle sind in Abbildung 7.30 als Prozessmodule M_k sowie M_b mit hellgrauer Füllung dargestellt. Dabei ist ein zusätzlicher Bedarf an K_1 als Hyperkante dargestellt: So fließen die beiden Produktflüsse K_1 aus M_1 und M_k zusammen und versorgen M_3, falls der Bedarf q^I von K_1 in $M_3 \geq$ der Produktion von K_1 mit der Menge p^I in

M_1 ist. Ist hingegen die Menge $p^I > q^I$, verbleibt ein Überschuss an K_1 mit einer Menge von b^I. Da dieser Überschuss im System nicht mehr denselben Zweck erfüllt wie das Koppelprodukt K_1, wird er gesondert modelliert als Produktfluss P_b. Das Prozessmodul M_b kann die Substitution eines entsprechenden Äquivalenzsystems repräsentieren (vgl. „Gutschriftenmethode“ in Kapitel 5.3.5). Stellt der Überschuss P_b keinen Nutzen mehr dar und weist Abfalleigenschaften auf, kann M_b stattdessen den Beseitigungsprozess repräsentieren.

7.7.4 Gewichtete Graphen im Zuge der Sachbilanz

Im Zuge der Sachbilanz werden für jedes Prozessmodul die Inputs und Outputs bestimmt. In der grafischen Darstellung des Produktsystems können den Pfeilen in dieser Phase Gewichtungsfaktoren am Pfeilanfang und Pfeilende zugeordnet werden. Nach dieser Datenerhebung sind die als Graphen interpretierten Produktsysteme zusammenhängend, gerichtet *und* gewichtet. Die Produktflüsse eines Produktsystems fungieren als gerichtete Kanten, welche die einzelnen Prozessmodule des Produktsystems, die Knoten des Graphen, *vorwiegend produktsystemintern* und *stets innerhalb* der Technosphäre miteinander verbinden. In Abbildung 7.31 ist das Produktsystem aus Abbildung 7.27 des Kapitels 7.7 dargestellt, dem Gewichtungsfaktoren zugewiesen worden sind. Dies bedeutet, dass die Datenerhebung der *einzelnen* Prozessmodule für dieses Produktsystem bereits erstellt wurde. Der Pfeil des Zwischenprodukts P_1 in Abbildung 7.31 beschreibt, dass durch ein einmaliges Ablaufen des Prozessmoduls M_1 eine Einheit des Produkts P_1 erzeugt wird, wobei in Prozessmodul M_2 für einen einmaligen Durchlauf tatsächlich m Einheiten des Produkts P_1 erforderlich sind. Die Produktflüsse werden als *durchgehende* Pfeile dargestellt, wenn die Gewichtungsfaktoren die Ergebnisse der Sachbilanz darstellen - sie zeigen dann die *Gesamtmenge* der im Produktsystem erforderlichen Inputs bzw. Outputs an den jeweiligen Pfeilenden bzw. Pfeilanfängen auf. Hierfür sind die einzelnen Skalierungsfaktoren unter Berücksichtigung des Bedarfsvektors zu berechnen. Dies ist erst dann möglich, wenn das Produktsystem vollständig modelliert worden ist.

Zur Unterscheidung wird daher die Pfeillinie in dieser Arbeit bewusst durch einzelne Punkte unterbrochen, wenn die Gewichtungsfaktoren der Produktflüsse noch keine absoluten Quantitäten des gesamten Produktsystems darstellen. Diese Darstellungsart soll verdeutlichen, dass sich die dargestellten Gewichtungsfaktoren auf die Bilanz *einzelner* Prozessmodule beziehen und sind nicht zu verwechseln mit den benötigten Mengen für das *gesamte* Produktsystem.

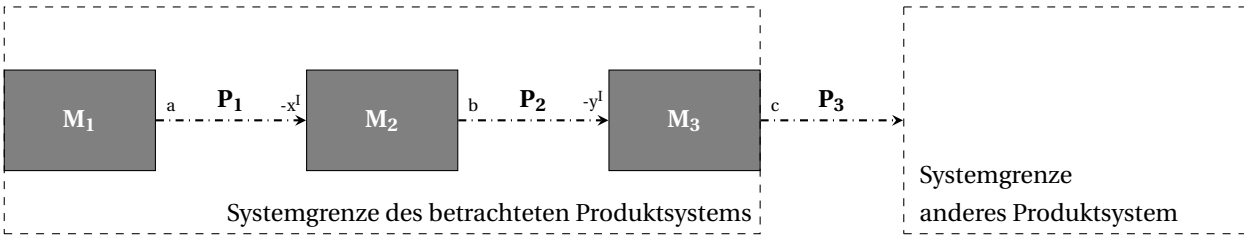


Abbildung 7.31: Produktsystem mit den je Prozessmodul benötigten Input- und hergestellten Outputmengen.

Die Gewichtungsfaktoren derlei unterbrochener Pfeillinien geben somit an, in welcher Menge die Zwischenprodukte bzw. das/die Endprodukt(e) von *einem einzelnen* Prozessmodul hergestellt (im Falle eines Outputs) bzw. benötigt (im Falle eines Inputs) werden.

7.7.5 Grundregeln zur matrixbasierten Modellierung

Bei der Modellierung eines Technologiesystems als Produktsystem sind zum einen die Rahmenbedingungen und Anforderungen der Ökobilanzierung zu beachten. Zum anderen sind bestimmte Kriterien notwendig, damit das sich durch die entsprechende Produktmatrix des Graphen und des Bedarfsvektors gegebene lineare Gleichungssystem gelöst werden kann. Durch folgende Punkte soll die Basis hierfür geschaffen werden:

- Jedes Prozessmodul muss genau einen Hauptproduktfluss aufweisen.
- Prozessmodule, deren Zweck es ist, ein Produkt zu verbrauchen oder dieses durch Systemerweiterung zu substituieren, haben einen *eingehenden* Hauptproduktfluss.
- Prozessmodule, deren Zweck es ist, ein Produkt zu erzeugen, haben einen *ausgehenden* Hauptproduktfluss.
- Inputs werden als negative Werte erfasst, Outputs werden als positive Werte erfasst.
- Die Prozessmodule sowie deren Hauptprodukt(flüss)e eines Produktsystems sollen eindeutig sein und nicht redundant.
- Produktflüsse können Output *oder* Input eines Prozessmoduls sein, jedoch nicht beides zugleich.
- Das Produktsystem soll zusammenhängend sein.
- Der Bedarfsvektor soll einen Einheitsvektor bzw. das Vielfache eines Einheitsvektors darstellen.
- Zusatzproduktflüsse, die im Produktsystem kein Hauptprodukt repräsentieren, werden in der Produktmatrix nicht erfasst.
- Bei der Gruppierung von Prozessmodulen oder Produktsystemen müssen die Hauptproduktflüsse der einzelnen Elemente identisch sein.
- Interne Koppelproduktflüsse, die als Inputs in Prozessmodule des Produktsystems fließen, jedoch den Bedarf allein nicht decken können, erfordern ein Prozessmodul, durch welches als Hauptprodukt das Koppelprodukt produziert wird.
- Interne Koppelproduktflüsse, die als Inputs in Prozessmodule des Produktsystems fließen und den Bedarf vollständig allein decken, ohne Überschüsse zu erzeugen, können optional als Produktflüsse modelliert werden, sie sind jedoch in der Berechnung des Skalierungsvektors überflüssig und werden daher in der Kalkulation nicht berücksichtigt.
- Treten Überschüsse erzeugter Koppelprodukte auf, die intern keine Verwendung finden, die aber in der Bilanzierung berücksichtigt werden sollen, muss der Umgang mit diesen externen Zusatzprodukten durch Substitution oder Allokation entsprechend modelliert werden.

Im Zuge der Entwicklung des intervallbasierten Verfahrens wurden die genannten Kriterien zur Berechnung näher spezifiziert sowie weitere Modellierungskriterien formuliert, die für den intervallbasierten Ansatz erforderlich sind. Diese Kriterien sind in Kapitel 8.4 aufgeführt.

Kapitel 8

Entwicklung des intervallbasierten Berechnungsverfahrens

In diesem Kapitel werden alle wesentlichen Aspekte erläutert, die für die Entwicklung des intervallbasierten Berechnungsverfahrens als wichtig erachtet wurden. Hierzu werden die Zielsetzung des Berechnungsverfahrens in Kapitel 8.1 sowie die Gründe für das intervallbasierte Verfahren in Kapitel 8.2 aufgeführt. Dies bedarf auch grundlegender Annahmen, die in Kapitel 8.3 aufgezeigt werden. Daneben werden in Kapitel 8.4 die spezifischen Anforderungen an das Berechnungsverfahren vorgestellt, die zur Durchführbarkeit der matrixbasierten Methode unter Anwendung der Intervallarithmetik notwendig sind. Anschließend wird in Kapitel 8.5 die Vorgehensweise bei der Konzepterstellung erläutert und zuletzt in Kapitel 8.6 die Vorgehensweise zur Analyse und Verifikation der Methoden beschrieben, auf denen die Entwicklung des Berechnungsverfahrens basiert.

8.1 Zielsetzung des Berechnungsverfahrens

Mit dem im Rahmen dieser Dissertation entwickelten und auf der Intervallarithmetik basierenden Verfahren zur ökologischen Bilanzierung sollen belastbare, transparente und möglichst präzise Aussagen über die potentiellen Umweltwirkungen getroffen werden können. Dabei sollen Unsicherheiten aller Art, welche trotz sorgfältiger Vorgehensweise nicht eliminiert werden können, direkt im Zuge der Berechnung praxisgerecht berücksichtigt werden. Aus dem Anspruch einer möglichst hohen Aussagekraft bei gleichzeitiger Transparenz und Belastbarkeit der Ergebnisse resultiert auf mathematisch-numerischer Ebene das Ziel, möglichst schmale Ergebnisintervalle zu generieren, ohne dabei *mögliche und relevante* Unsicherheiten - die in Form von Intervallen in der Bilanzierung berücksichtigt werden - außer Acht zu lassen. Hierfür sollen verschiedene Gleichungslöser implementiert und erprobt werden. Um dem Anspruch der Praktikabilität gerecht zu werden, soll auch der Rechenzeitaufwand des Verfahrens berücksichtigt werden. Um die Ergebnisse des intervallbasierten Verfahrens bewerten und miteinander vergleichen zu können, ist ein geeigneter Unsicherheitsindex zu entwickeln. Mit diesem soll die Größe der Unsicherheit in den Bilanzierungsergebnissen abgeschätzt werden können, um vereinfacht einordnen zu können, wie unsicher die Ergebnisse in Relation zu anderen Ergebnissen sind.

8.2 Gründe für den intervallbasierten Ansatz

Der intervallarithmetische Ansatz erlaubt allgemein einen konformen Umgang zur transparenten und praktikablen Behandlung von Unsicherheit. Im Folgenden werden die spezifischen Vorteile aufgelistet, die eine intervallbasierte Ökobilanzierung ermöglicht:

- Die Definition unsicherer Werte in Form von Intervallen ist für alle in der Ökobilanzierung auftretenden Arten von Unsicherheit möglich. Dies ist besonders von Vorteil, wenn keine Informationen darüber vorliegen, welche Art von Unsicherheit vorliegt oder sich mehrere Arten von Unsicherheit überlagern.

- Die Erfassung von Unsicherheit in Form von Intervallen wird insbesondere für unscharfe Daten empfohlen (vorgeschlagene Verfahren für unscharfe Daten vgl. Kapitel 6.3). Dazu zählen auch Daten, die unsicher aufgrund mangelnder oder fehlender Informationen sind, wie es in der Ökobilanzierung häufig der Fall sein kann (Bedeutung von Unsicherheit in der Ökobilanzierung vgl. Kapitel 6.2.1).

- Der vergleichende, relative Ansatz in der Ökobilanzierung hat zur Folge, dass Sicherheitsfaktoren wie bei statischen Berechnungen für bspw. unsichere Daten nicht geeignet sind (Problematik der fehlenden, sicheren Seite in der Ökobilanzierung vgl. Kapitel 6.4.2). Eine Erfassung unsicherer Werte als Intervall erlaubt eine Berücksichtigung von Unsicherheit, ohne den relativen Bezug zu anderen Werten zu beeinflussen.

- Liegen begrenzte Kenntnisse seitens der Hersteller zu vorgelagerten Lieferketten sowie zu nachgelagerten Entsorgungsprozessen vor, erlaubt eine „Gruppierung“ (Definition von „Gruppen“ vgl. Kapitel 5.2.18) diverser Prozesse eine Berücksichtigung der Ungewissheit, ohne dabei die Problematik der fehlenden sicheren Seite durch den relativen Bezug außer Acht zu lassen.

- Diverse, qualitativ oder quantitativ abweichende Datensätze, welche dieselbe funktionelle Einheit aufweisen, können ebenfalls „gruppiert“ werden und sind somit als Intervall erfassbar. Dadurch bleiben die festgestellten Abweichungen berücksichtigt.

- Der Aufwand zur Erfassung von Unsicherheit in Form von Intervallen ist begrenzt, da nur wenige zusätzliche Informationen erforderlich sind.

- Der Aufwand zur Auswertung von Unsicherheit ist bei Anwendung des entwickelten Unsicherheitsindexes *UnsI* (Erläuterung zum Unsicherheitsindex vgl. Kapitel 8.5.4) überschaubar. Dieser wird über die Intervallweiten berechnet, welche direkt im Zuge der Bilanzierung berechnet worden sind. Es sind folglich keine gesonderten Analysen im Nachgang notwendig.

- Der aktiv aufzuwendende Zeitaufwand der Ökobilanzierenden ist aus oben aufgeführten Punkten ebenfalls vertretbar. Der Rechenzeitaufwand der Algorithmen wurde bei der Verfahrensentwicklung berücksichtigt.

- Politisch und gesellschaftlich rücken ökologische Themen zunehmend in den Fokus. Das setzt zum einen Unternehmen unter Druck. Zudem werden auch von Kunden vermehrt Informationen zur Nachhaltigkeit der Produkte eingefordert und in die Entscheidungsfindung einbezogen. Eine transparente Darstellung der Unsicherheit direkt im Ergebnis ist auch oder *besonders* außerhalb des Expertenkreises von Ökobilanzen von großer Bedeutung. Der Wegfall präziser Werte im Ergebnis kann Fehldeutungen über die Sicherheit von Ökobilanzen entgegenwirken, gleichzeitig bleibt die Unsicherheit zwingend bei vergleichenden Analysen berücksichtigt, da sie direkt in den Ergebnissen enthalten ist. Dazu gibt es über den Unsicherheitsindex *UnsI* die Möglichkeit, die Sicherheit der Ergebnisse einzuordnen.

8.3 Grundlegende Annahmen

Beim intervallbasierten Ansatz der Ökobilanzierung sollen unsichere Informationen eingegrenzt werden können. Die Intervalle unsicherer Kennwerte beschreiben den *möglichen* und *relevanten* Bereich, den sie annehmen können; gleiches gilt für die aus der ökologischen Bilanzierung resultierenden Ergebnisse. Dabei ist es äußerst wichtig, die Intervalle auch als solche zu interpretieren. So sind die Untergrenzen der Ergebnisintervalle *nicht* als das „bestmöglich“ zu Erreichende zu deuten, die z. B. durch eine Optimierung der Produktionsprozesse erzielbar wären. Gleichsam sind aus den Obergrenzen der Intervalle *nicht* etwaige „worst Case“-Szenarios abzuleiten. Die Intervallweiten müssen als *unsichere* Daten verstanden werden.

In Abbildung 8.1 ist exemplarisch ein Intervall dargestellt. Dabei ist auch ein Hauptwert innerhalb des möglichen Wertebereichs abgebildet, der für den „typischen“ Wert steht, der bei der herkömmlichen Bilanzierung verwendet wird bzw. verwendet worden wäre.

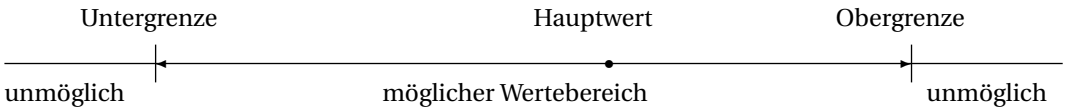


Abbildung 8.1: Intervall zur Abgrenzung des Möglichen vom Unmöglichen bzw. des Relevanten vom Irrelevanten; Hauptwert zur Repräsentation des „typischen“ Werts.

Die Intervallgrenzen der Ergebnisintervalle, die mit dem in dieser Arbeit entwickelten Verfahren berechnet werden, zeigen somit die Grenze zum *Unmöglichen* oder *Irrelevanten* auf. Auch bzw. besonders im unmittelbaren Vergleich zu anderen Ergebnisintervallen muss dies Beachtung finden: solange sich zwei Intervalle überschneiden, kann keine Aussage darüber getroffen werden, welches der beiden Ergebnisse günstiger ist.

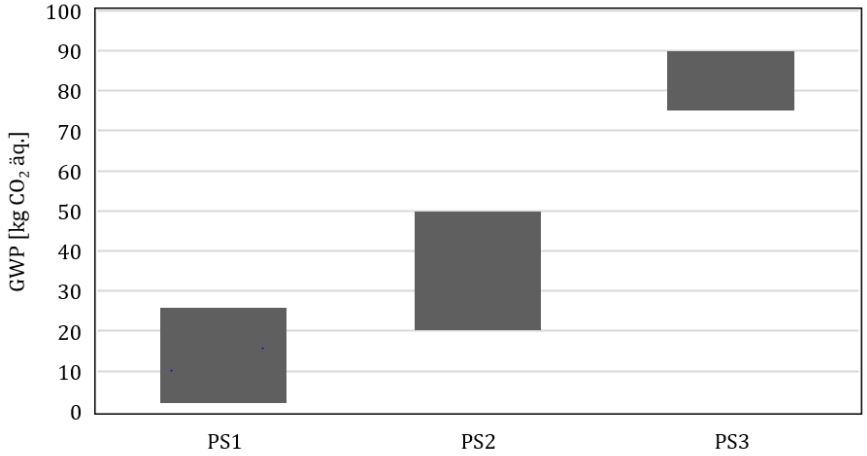


Abbildung 8.2: *GWP*-Intervalle [kg CO_2 äq.] von drei fiktiven Produktsystemen PS1, PS2 und PS3.

Ein Beispiel hierzu ist in Abbildung 8.2 dargestellt mit den *GWP*-Intervallen der drei fiktiven Produktsyste-

me PS1, PS2 und PS3. Dabei verfügt PS1 über ein *GWP*-Intervall von [2; 26] kg CO_2 äq., PS3 hingegen weist ein *GWP*-Intervall von [75; 90] kg CO_2 äq. auf. Da die Obergrenze des *GWP*-Intervalls von PS1 *unter* der Untergrenze des *GWP*-Intervalls von PS3 liegt, ist die Aussage zulässig, dass das Produktsystem PS1 hinsichtlich des *GWP* vergleichsweise niedriger einzustufen ist als das Produktsystem PS3. Das Produktsystem PS2 hingegen misst ein *GWP*-Intervall von [20; 50] kg CO_2 äq., womit sich die Ergebnisintervalle von PS1 und PS2 überlappen. Damit ist gemäß den intervallarithmetischen Vergleichsoperatoren nicht abschließend zu beurteilen, welche der beiden Produktsysteme hinsichtlich des *GWP* als geeigneter zu bewerten ist (intervallarithmetische Vergleichsoperatoren vgl. auch Kapitel 3.4).

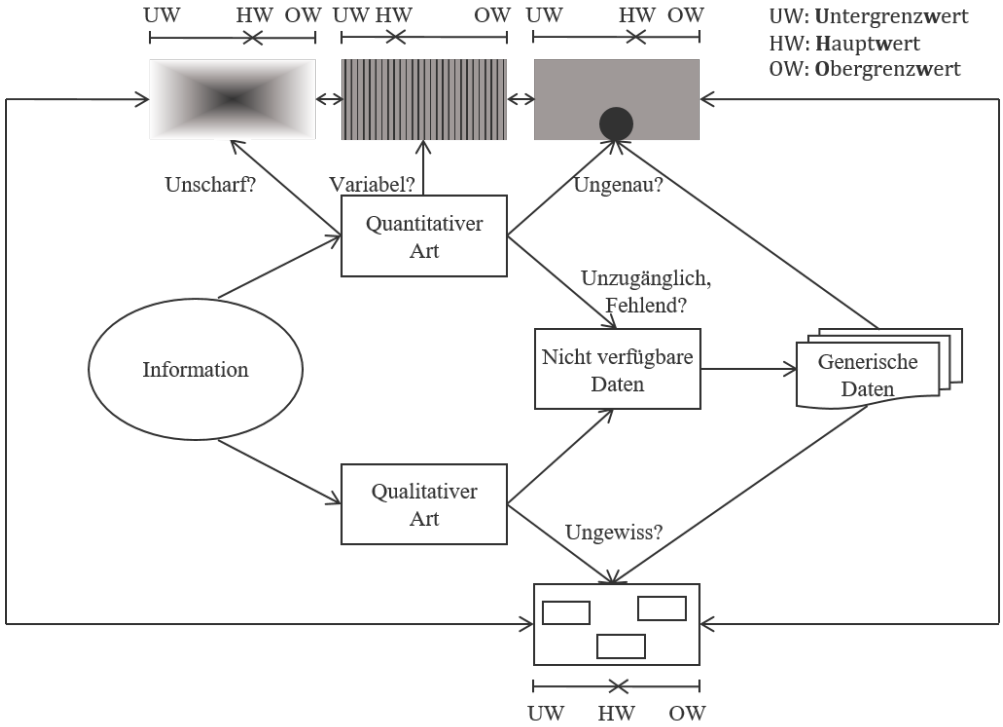


Abbildung 8.3: Alle Arten von Unsicherheit werden im Zuge der Ökobilanzierung als Intervalle erfasst.

Bei dem in dieser Arbeit entwickelten Verfahren wird davon ausgegangen, dass die in einer Ökobilanzierung auftretenden Ungewissheiten, Unpräzisionen und Ungenauigkeiten als Intervalle erfasst bzw. zusammengefasst werden können (vgl. Abbildung 8.3). Im Gegensatz zu einem stochastischen Ansatz werden beim intervallbasierten Verfahren keine Wahrscheinlichkeitsverteilungen für die einzelnen Zahlenwerte innerhalb des Intervalls ermittelt. Im übertragenen Sinn sind somit die Wahrscheinlichkeitsmaße aller reellen Zahlenwerte innerhalb des Intervalls gleich und ihr Auftreten ist damit gleich wahrscheinlich - es erfolgt jedoch diesbezüglich beim intervallbasierten Ansatz keine Wertung.

8.4 Anforderungen an das Verfahren

Für das matrix- und intervallbasierte Berechnungsverfahren werden verschiedene Kriterien festgelegt, damit eine Berechnung möglich ist und diese plausible Ergebnisse liefert. Sie berücksichtigen dabei sowohl die Anforderungen, die durch die Methodik der Ökobilanzierung gestellt werden, als auch die Voraussetzungen, die sich aus einer intervallbasierten Berechnung ergeben.

8.4.1 Berechnungskriterien

Dem matrixbasierten Berechnungsverfahren liegen Vorbedingungen (Definition „Vorbedingungen“ vgl. Kapitel 2.2.4) zugrunde, welche zur Anwendbarkeit der Algorithmen erfüllt sein müssen. Diese haben wesentlichen Einfluss auf die Modellierung des Produktsystems. Diese Bedingungen lauten wie folgt:

1. Die Produktmatrix P muss regulär sein.
2. Eine Division durch ein Punktintervall mit einem Wert von Null oder durch ein Intervall, welches die Null impliziert, ist nicht zulässig.
3. Die Spaltenanzahl der Produktmatrix P muss der Zeilenanzahl des Bedarfsvektors f entsprechen.
4. Kehrwertbildungen von Intervallen sind nur zulässig, wenn diese die Null nicht implizieren.

8.4.2 Gültigkeitskriterien

Den intervallbasierten Gleichungslösern liegen Nachbedingungen zugrunde, die zur Verifikation der Algorithmen erfüllt sein müssen (Definition „Nachbedingungen“ vgl. Kapitel 2.2.4). Diese Bedingungen lauten wie folgt:

1. Die Intervalluntergrenze darf den ermittelten, referenziellen Untergrenzwert aus der vollständigen Senzarioanalyse nicht überschreiten.
2. Die Intervallobergrenze darf den ermittelten, referenziellen Obergrenzwert aus der vollständigen Senzarioanalyse nicht überschreiten.
3. Die Ergebnisintervalle müssen die Intervalle der vollständigen Szenarioanalyse vollständig umhüllen.
4. Die berechneten Intervalle mit Intervallweiten ungleich Null dürfen die Null nicht implizieren.

8.4.3 Modellierungskriterien

Die folgenden Modellierungskriterien ergänzen die festgelegten, allgemeinen Grundregeln zur Modellierung in der matrixbasierten Ökobilanzierung (Grundregeln zur Modellierung vgl. Kapitel 7.7.5) um Anforderungen, die bei Anwendung des intervallarithmetischen Ansatzes berücksichtigt werden müssen. Die Bedingungen diesbezüglich lassen sich wie folgt zusammenfassen:

1. Das Hauptprodukt eines Prozessmoduls ist eindeutig bestimmt, d. h. der quantifizierte Output des zugehörigen Hauptproduktflusses ist frei von Unsicherheiten und seine Intervallweite ist somit null.
2. Die Erzeugung oder der Verbrauch von Produkten, die im Prozessmodul selbst kein Hauptprodukt darstellen, darf unsicher sein.
3. Unsicherheit darüber, ob ein Hauptprodukt im Produktsystem als Input verbraucht oder als Output erzeugt wird, ist nicht zulässig.
4. Ist unsicher, ob ein internes Zusatzprodukt den Bedarf decken kann oder ein Teil davon das Produktsystem verlässt, sind beide Fälle gesondert als Unsicherheit zu betrachten und zu modellieren.
5. Der Nutzen des Produktsystems ist mit dem Referenzfluss eindeutig zu bestimmen, d. h. die Faktoren des Bedarfsvektors weisen Intervallweiten von null auf.

6. Der Einfluss von Elementarflüssen auf eine Umweltwirkung kann unsicher sein. Es muss jedoch gewährleistet sein, dass sich der durch den Charakterisierungsfaktor beschriebene Einfluss positiv *oder* negativ auf die entsprechende Wirkungskategorie auswirkt.

8.5 Aufbau und Vorgehensweise bei der Entwicklung

Im Zuge der Entwicklung des Verfahrens wird die Methodik der Ökobilanzierung um einen intervallarithmetischen Ansatz erweitert, Gleichungslöser implementiert, analysiert und modifiziert sowie daraus ein allgemeines Verfahren abgeleitet (vgl. auch Abbildung 8.4). Hierfür werden diverse Algorithmen zur Lösung intervallarithmetischer Gleichungssysteme berücksichtigt und die Ergebnisse mittels Parameterstudien vergleichend analysiert und bewertet.

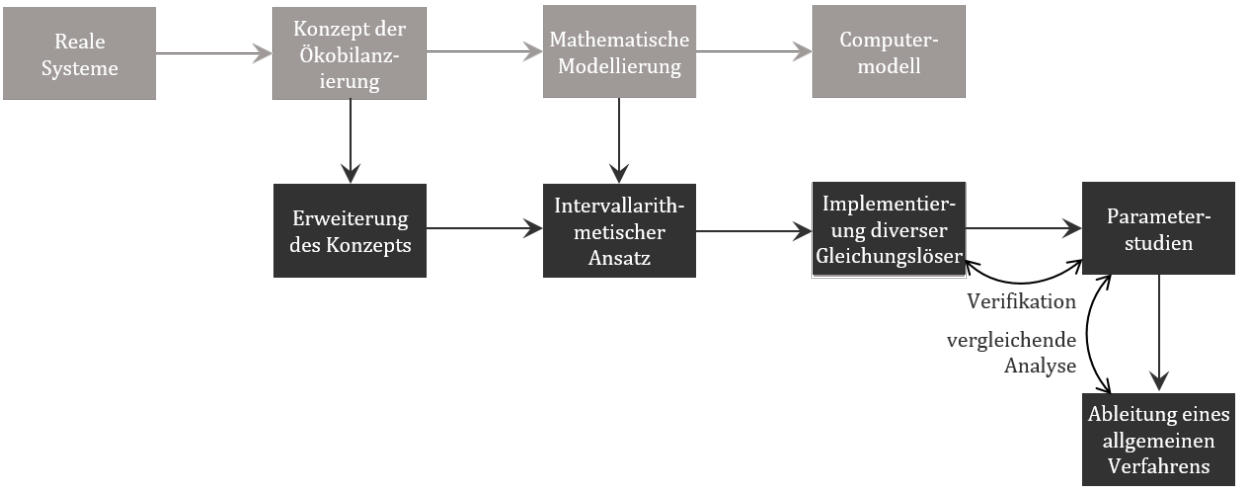


Abbildung 8.4: Vorgehensweise bei der Entwicklung des intervallbasierten Berechnungsverfahrens.

Geeignete, aus der Literaturrecherche ermittelte Verfahren werden implementiert (Auswahl dieser Algorithmen vgl. auch Kapitel 8.5.1). Zusätzlich werden das Gaußsche Eliminationsverfahren und das Adjunkten-Verfahren auf die intervallarithmetische Ebene übertragen und verschiedene Varianten zu den beiden Methoden entworfen (Erläuterung zu diesen Algorithmen vgl. Kapitel 8.5.2). Die Hülle der Lösungsmenge kann mit der vollständigen Szenarioanalyse ermittelt werden. Aus diesem Grund werden ebenfalls hierfür geeignete Algorithmen erstellt und als die Solver *brute* und *piga* implementiert. Die Ergebnisse des Solvers *piga* dienen den übrigen Gleichungslösern auch als Referenz (Erläuterung zu diesem Algorithmus vgl. ebenfalls Kapitel 8.5.2). Der Entwicklungssatz von Laplace wird in dieser Arbeit für die Adjunkten-Verfahren verwendet. Dieser ist hinsichtlich des Rechenzeitaufwands als nicht praktikabel einzustufen. Deshalb wird zusätzlich ein Verfahren auf Basis des Teile-und-Herrsche-Prinzips entwickelt, mit welchem die Produktmatrix in kleinere Matrizen unterteilt wird (Verfahren zum Splitten der Matrix vgl. Kapitel 8.5.3). In Tabelle 8.1 werden die untersuchten Berechnungsmethoden sowie deren Kurzbezeichnungen zusammenfassend dargestellt.

Die Implementierung des entwickelten Verfahrens erfolgt im Paket *Ivari* („**I**nter**v**all**ari**thmetik“). Das Paket wird als Kernel in das Java-Programm *MultiVaLCA* integriert (Erläuterung zum Programm *MultiVaLCA* vgl. Kapitel 5.5). Die Ergebnisse der implementierten Gleichungslöser werden anhand von Parameterstudien miteinander verglichen. Der Aufwand wird ebenfalls in gesonderten Testreihen erprobt. Um die Intervallweiten der Ergebnisintervalle vergleichend gegenüberstellen zu können, wird der Unsicherheitsindex *UnsI* verwendet. Bei der Aus-

Tabelle 8.1: Untersuchte Berechnungsmethoden sowie Kurzbezeichnungen der Solver.

Methode	Funktion	Ansatz	Besonderheit	Hülle	gültig	Aufwand
piga	approximations-freie Lösung	vollständige Szenarioanalyse	keine Dopplungen	✓	alle	$O(2^{n^2})$*
brute			Dopplungen möglich	✓	alle	$O(2^{n^2})$*
siga	angenäherte, schnellere Lösung	Gaußsche Elimination	partielle Pivotisierung	möglich	alle	$O(n^3)$*
salt			alternative Vorsortierung			$O(n^3)$*
sopt			beste Lösung *siga* und *salt*			$O(n^3)$*
adivaOrigin	angenäherte, alternative Lösung	Adjunkten-Verfahren	ohne Vorsortierung	möglich	alle	$O(n^n)$*
adiheu			mit Vorsortierung			$O(n^n)$*
adiperm			mit Zeilen-Permutation			$O(n^{2n})$*
beeck	etablierte Verfahren	Präkondition-ierung	Beeck-Verfahren	✓	M-Matrix	$O(n^3)$**
hansen			Verfahren nach Hansen	möglich	alle	$O(n^4)$**
rohn			Verfahren nach Rohn			$O(n^4)$**
ning22			Verfahren nach Ning	Oberhülle	H-Matrix	$O(n^3)$**
ning26				✓	inv.-pos.	$O(n^3)$**

* Der Aufwand im schlechtesten Fall beruht auf Angaben zu verwendeten Methoden aus der Literatur (Komplexität für *piga* und *brute* vgl. Kapitel 2.5.4, Komplexität zur Lösung der Gleichungssysteme mit den Gauß-Verfahren vgl. Kapitel 2.4.2, Komplexität zur Berechnung der Determinanten über den Laplaceschen Entwicklungssatz für die Adjunkten-Verfahren vgl. Kapitel 2.4.1).
** Die implementierten Algorithmen der Verfahren von Beeck, Hansen, Ning und Rohn wurden anhand der Zuweisungen abgeschätzt (Abschätzung anhand von Zuweisungen vgl. auch Kapitel 2.2.3).

wertung der Parameterstudien werden sowohl einzelne Matrixtypen als auch Produktsystemstrukturen der Ökobilanzierung berücksichtigt. Die Analyse erfolgt insbesondere im Hinblick auf möglichst schmale Intervallweiten der Ergebnisintervalle, die mit praktikablen Zeitaufwand berechenbar sein sollen.

8.5.1 Auswahl etablierter Methoden

Die aus der Literaturrecherche ermittelten, intervallbasierten Verfahren von Beeck sowie Hansen als auch die Verfahren von Ning werden als relevant erachtet (Verfahren von Beeck, Hansen und Ning vgl. Kapitel 3.7.3, Kapitel 3.7.4 und Kapitel 3.7.6). Mit einzelnen Verfahren kann mitunter für bestimmte Produktmatrizen die Hülle der Lösungsmenge ermittelt werden. In diesem Zusammenhang werden häufig die Matrizen nach mathematischer Definition unterschieden in H-, Z- und M-Matrizen (Definition der Matrixtypen vgl. Kapitel 3.5). Für das Verfahren von Hansen und das Verfahren von Rohn wird die C-Matrix der Intervallmatrix A^I bestimmt. Ein weiteres Verfahren ist das Lösungsverfahren von Beeck, mit dem bei M-Matrizen über die Untergrenzen und Obergrenzen der Intervallmatrix A^I die Hülle ermittelt werden kann. Das Verfahren von Ning, welches auf Theorem 2.2 aufbaut, soll Oberhüllen liefern. Das Verfahren von Ning nach Theorem 2.6 ist nur bei invers-positiven Matrizen gültig. Das Verfahren von Nirmala wird nicht implementiert (Verfahren von Nirmala vgl. Kapitel 3.7.7).

8.5.2 Entwicklung weiterer Gleichungslöser

Neben den im vorigen Kapitel 8.5.1 vorgestellten Verfahren werden das intervallbasierte Gauß-Verfahren sowie das intervallbasierte Adjunkten-Verfahren als weitere Gleichungslöser berücksichtigt. Die intervallarithmetischen Rechenoperationen können aufgrund abhängiger Intervalle einen Einfluss auf die Intervallweiten der Ergebnisse haben (Erläuterung zu abhängigen Intervallen vgl. Kapitel 3.6.2). Daher werden verschiedene Varianten für die beiden Verfahren entwickelt und diese vergleichend gegenübergestellt. Beim Intervall-Gauß-Verfahren werden verschiedene Varianten zur Pivotisierung und Vorsortierung der Matrix entworfen. Das Adjunkten-Verfahren

wird auf die intervallarithmetische Ebene übertragen und verschiedene Varianten zur Vorsortierung der Matrix entworfen. Beim Adjunkten-Verfahren wird die Inverse über die Cofaktoren ermittelt, wobei zur Berechnung der Unterdeterminanten der Laplacesche Entwicklungssatz verwendet wird. Dies ist sehr zeitaufwändig und gilt als nicht praktikabel. Daher wird ein Algorithmus entwickelt, mit dem der Rechenzeitaufwand durch Splitten großer Matrizen in kleinere Submatrizen reduziert werden soll.

Die Hülle der Lösungsmenge kann durch eine vollständige Szenarioanalyse ermittelt werden. Daher wird auch hierfür ein entsprechender Algorithmus entwickelt. Dieser ist aber aufgrund des Rechenzeitaufwands nur auf kleine Matrizen anwendbar.

Ansätze des Intervall-Gauß-Verfahrens

Es werden verschiedene Varianten des Intervall-Gauß-Verfahrens entwickelt. Diese lauten wie folgt:

- Der Gleichungslöser *siga*, bei dem das Intervall-Gauß-Verfahren mit Vorsortierung und einer anschließenden, partiellen Pivotisierung mit größtem Pivotelement implementiert wird.
- Der Gleichungslöser *salt*, bei dem das Intervall-Gauß-Verfahren mit alternativer Vorsortierung und einer anschließenden partiellen Pivotisierung mit größtem Pivotelement implementiert wird.
- Der Gleichungslöser *sopt* (Zusammenführung von *siga* und *salt*), bei dem sowohl *siga* als auch *salt* verwendet werden und das beste Ergebnis der beiden Solver zurückgegeben wird.

Die Vorsortierung, welche beim Gleichungslöser *siga* angewandt wird, formt das Gleichungssystem derart um, dass die Produktmatrix soweit möglich einer oberen Dreiecksmatrix entspricht (vgl. auch Abbildung 8.5). Bei der reinen Umsortierung der Produktmatrix zu einer angenäherten oberen Dreiecksmatrix werden noch keine Rechenoperationen durchgeführt. Die Vorsortierung soll unnötige Rechenschritte verhindern.

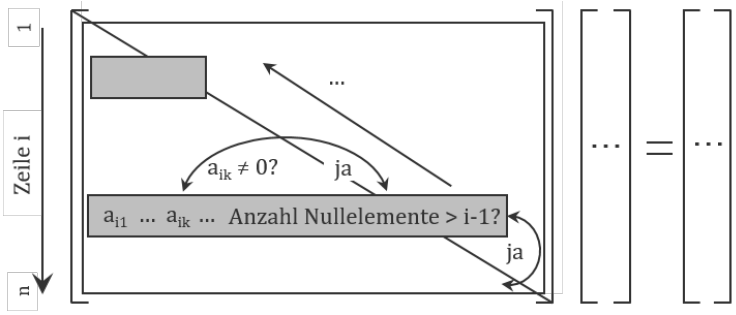


Abbildung 8.5: Vorgehensweise des Algorithmus zur Vorsortierung.

Diese Struktur entspricht der natürlichen Matrixstruktur linearer und baumartiger Produktsysteme, womit es möglich ist, dass die Prozessmodule und Hauptprodukte weitgehend gemäß ihrer Reihenfolge im zu repräsentierenden Produktsystem sortiert werden.

Die partielle Pivotisierung wird beim Gleichungslöser *siga* und *salt* angewandt. Sie wird derart implementiert, dass spaltenweise das betragsmäßig größte Element als Pivotelement definiert wird (vgl. auch Abbildung 8.6). Dies ist bei Anwendung des Gaußschen Algorithmus auf lineare Gleichungssysteme eine übliche Vorgehensweise.

Daneben wird ein Verfahren implementiert, mit dem eine alternative Vorsortierung der Matrix vorgenommen werden kann (vgl. auch Abbildung 8.7). Die alternative Vorsortierung wird analog zur bereits vorgestellten Vorsortierung durchgeführt und beim Gleichungslöser *salt* angewandt. Bei dieser Vorsortierung dürfen jedoch Elemente, die Werte kleiner Null *oder* gleich Null aufweisen, unterhalb der Hauptdiagonalen verbleiben.

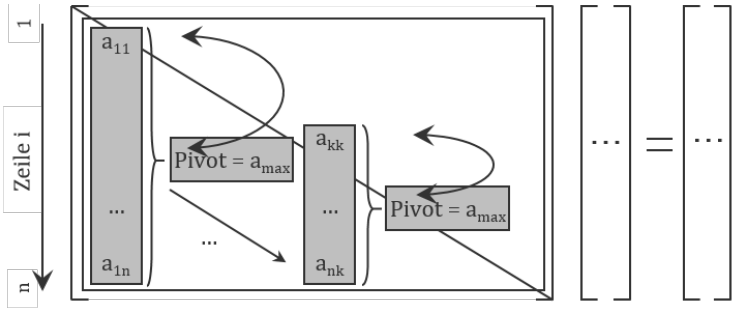


Abbildung 8.6: Vorgehensweise des Algorithmus zur partiellen Pivotisierung.

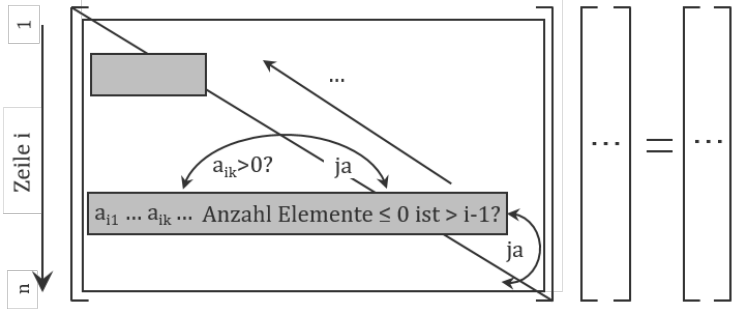


Abbildung 8.7: Vorgehensweise des Algorithmus zur alternativen Vorsortierung.

Ansätze des Intervall-Adjunkten-Verfahrens

Es werden verschiedene Varianten des Intervall-Adjunkten-Verfahrens entwickelt und miteinander verglichen. Die Berechnung der Unterdeterminanten erfolgt mit dem Entwicklungssatz von Laplace. Ziel dabei ist es, eine alternative Berechnung zu implementieren - daher wurde auf andere, effizientere Methoden zur Determinantenberechnung wie bspw. über das Gauß-Verfahren verzichtet. Um den hohen Rechenzeitaufwand bei Anwendung des Laplaceschen Entwicklungssatzes zu begrenzen, wurde eine Methode zum Splitten großer Matrizen entwickelt (Verfahren zum Splitten großer Matrizen vgl. Kapitel 8.5.3). Folgende Varianten des Intervall-Adjunkten-Verfahrens wurden entwickelt:

- Der Gleichungslöser *adivaOrigin*, bei dem das Adjunkten-Verfahren auf die intervallarithmetische Ebene übertragen wird.
- Der Gleichungslöser *adiheu*, bei dem das Intervall-Adjunkten-Verfahren mit alternativer Vorsortierung implementiert wird.
- Der Gleichungslöser *adiperm*, bei dem das Intervall-Adjunkten-Verfahren mit vorangehender Permutation der Zeilen implementiert wird. Als Bewertungskriterium für die Auswahl derjenigen Matrix für die Berechnung wird diejenige permutierte Matrix verwendet, deren Determinantenintervall die schmalste Intervallweite aufweist.

Die alternative Vorsortierung des Gleichungslösers *adiheu* entspricht der alternativen Vorsortierung des vorherigen Abschnitts (vgl. auch Abbildung 8.7). Die zeilenweise und/oder spaltenweise Permutation der Matrix zielt darauf ab, die Zeilen und Spalten der Matrix derart zu vertauschen, dass die Intervallweiten der resultierenden Determinanten möglichst gering sind. Dabei wird davon ausgegangen, dass bei einer möglichst schmalen Inter-

vallweite der Determinante die Intervalle der Inverse so wenig wie möglich aufgeweitet werden und somit das Ergebnis des Gleichungssystems ebenfalls vergleichsweise schmale Ergebnisintervalle berechnet.

Verfahren zur vollständigen Szenarionanalyse

In dieser Arbeit wird die vollständige Szenarioanalyse verwendet, um die Hülle der Lösungsmenge zu ermitteln. Bei der Entwicklung dieses Verfahrens werden die einzelnen Grenzwerte der in einem Produktsystem existierenden Intervalle als je zwei mögliche Szenarios definiert. Dafür wird auf die Unter- und Obergrenzen der Intervalle zurückgegriffen und alle Variationen, die sich ergeben, berechnet. Bei m Intervallen, denen jeweils die zwei Elemente „Untergrenze“ und „Obergrenze“ zugeordnet werden können, beträgt die Anzahl der Variationen:

$$V_{Anz} = 2^m$$

Als Ergebnis wird ein Intervallvektor ausgegeben, dessen Untergrenze die geringste untere Grenze aller Variationen darstellt, während gleichermaßen die Obergrenze die höchste obere Grenze aller Variationen repräsentiert. Im besten Fall ergeben die übrigen Gleichungslöser exakt die Unter- und Obergrenzen der vollständigen Szenarioanalyse wieder. Anderenfalls stellen die Ergebnisse der übrigen Gleichungslöser Oberhüllen dar. Liegen die Ergebnisintervalle der Gleichungslöser hingegen *innerhalb* des Ergebnisses aus der vollständigen Szenarioanalyse, sind diese nicht für den Ansatz der intervallbasierten Ökobilanzierung geeignet und werden ausgeschlossen.

8.5.3 Verfahren zum Splitten der Matrix

Der Algorithmus *adisplit* wird auf Basis des Teile-und-Herrsche-Prinzips entwickelt. Mit ihm sollen große Produktmatrizen in kleinere Submatrizen gesplittet werden. Die Methode *adisplit* wurde insbesondere für das zeitaufwändige Intervall-Adjunkten-Verfahren entwickelt, ist jedoch mit jedem Gleichungslöser kombinierbar. Bei der Entwicklung wurden die Kenntnisse über die identifizierten Systemstrukturen von Produktsystemen (Systemstrukturen von Produktsystemen vgl. Kapitel 7.1) genutzt. Die Produktmatrix wird vor der Berechnung nach Submatrizen linearer Produktsysteme sowie gegenströmiger Bäume durchsucht. Grund hierfür ist, dass diese Art von Produktsystemen zum einen in der Produktmatrix leicht idenzifizierbar sind, zum anderen ist hier die Abkapselung und gesonderte Berechnung mit vergleichsweise effizienten Verfahren möglich, welche die Hülle der Lösungsmenge berechnen.

Die Methode wird solange rekursiv aufgerufen, bis keine Matrizen dieser Art mehr gefunden werden. Es ist auch möglich, baumartige Subsysteme *innerhalb* des Produktsystems zu trennen. Dabei ist je Produktsystem genau ein Vorgänger-Prozessmodul zulässig. Die identifizierten Submatrizen werden von der Produktmatrix entfernt, gesondert berechnet und als aggregierte Prozessmodule (Definition „aggregierte Prozessmodule“ vgl. Kapitel 5.2.12) in die Produktmatrix eingetragen - was ihre Matrixdimension reduziert. Dies ist auch in Abbildung 8.8 exemplarisch anhand eines Produktsystems dargestellt. Während das dargestellte Produktsystem zunächst aus 20 Prozessmodulen besteht, werden bei der ersten Durchsuchung der Produktmatrix vier Subsysteme identifiziert und komprimiert. Die komprimierten Prozessmodule sind im darunterliegenden, überarbeiteten Produktsystem in grau dargestellt. Gleichsam werden in diesem so komprimierten Produktsystem zwei weitere Subsysteme identifiziert. Diese werden wiederum im nächsten Schritt komprimiert und nachfolgend in hellgrau dargestellt. Durch die Methode kann das Produktsystem insgesamt in drei Schritten auf acht Prozessmodule reduziert werden. Die Berechnung der reduzierten Produktmatrix erfolgt auf Basis der in den vorigen Kapiteln beschriebenen intervallbasierten Gleichungslöser. Es ist möglich, die Submatrizen mit anderen Gleichungslösern zu lösen als mit dem Gleichungslöser, der für die Produktmatrix verwendet wird.

Abbildung 8.8: Vorgehensweise des Algorithmus *adisplit* zum Verkleinern großer Matrizen.

8.5.4 Konzeption des Unsicherheitsindexes *UnsI*

Zur Bewertung und vergleichenden Analyse der Ergebnisintervalle wird der **Uns**icherheits**I**ndex) *UnsI* entwickelt, mit dem die Größe der Unsicherheit der jeweiligen Methoden eingeschätzt werden kann. *UnsI* berechnet sich aus den Grenzwerten des Intervalls wie folgt:

$$\mathrm{U}nsI = \frac{\overline{x} - \underline{x}}{\overline{x} + \underline{x}} \cdot 100\ [\%] \tag{8.1}$$

Ein sicherer Parameter, der somit als Punktintervall angegeben wird, weist einen *UnsI*-Wert von *0* Prozent auf. Ein unsicherer Parameter, der einen Bereich von *0* bis *x* erfasst, verfügt hingegen über einen *UnsI*-Wert von *100* Prozent. Da die Intervalle nach den festgelegten Kriterien des Verfahrens (Gültigkeitskriterien vgl. Kapitel 8.4.2) die Null nicht implizieren dürfen, sind die Intervallgrenzen ausschließlich positiv oder negativ. Der Bereich, den der *UnsI*-Wert daher annehmen kann, liegt daher zwischen mindestens *0* Prozent und maximal *100* Prozent.

Tabelle 8.2: Einstufung der Unsicherheit mit *UnsI*

UnsI in Prozent	**Einstufung der Unsicherheit**
UnsI-Wert ≥ 80 %	vergleichsweise sehr unsicher
80 % > *UnsI*-Wert ≥ 60 %	vergleichsweise unsicher
60 % > *UnsI*-Wert ≥ 40 %	vergleichsweise mäßig unsicher
40 % > *UnsI*-Wert ≥ 20 %	vergleichsweise sicher
20 % > *UnsI*-Wert	vergleichsweise sehr sicher

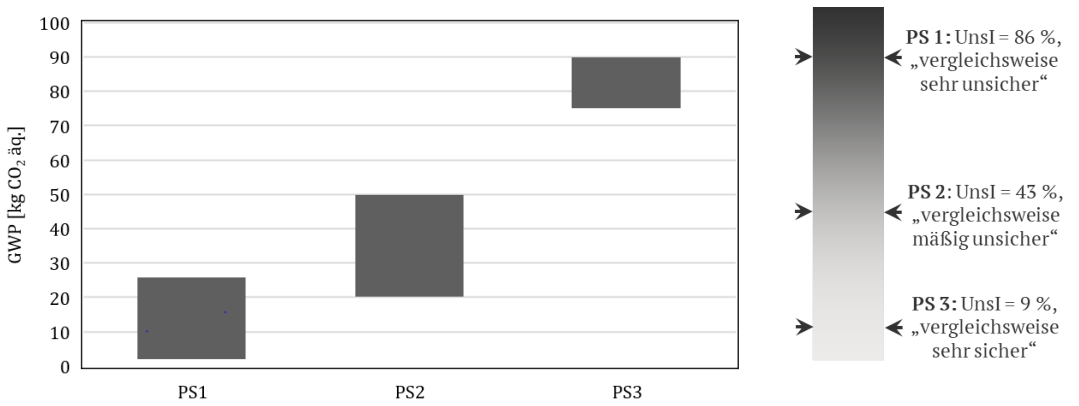
Die sprachliche Bewertung des *UnsI* ist in Tabelle 8.2 aufgelistet. Demnach kann bspw. ein *UnsI* bei einem Wert unter 20 % als „vergleichsweise sehr sicher“ bezeichnet werden, bei einem Wert von ≥ 40 % und < 60 % als „vergleichsweise mäßig unsicher“ und bei einem Wert von ≥ 80 % als „vergleichsweise sehr unsicher“.

In Abbildung 8.9a ist das Beispiel aus Abbildung 8.2 aus Kapitel 8.3 aufgegriffen. In diesem Beispiel sind diverse *GWP*-Intervalle für die fiktiven Produktsysteme PS1, PS2 und PS3 aufgeführt. Dabei weist PS1 ein *GWP*-Intervall von [2; 26] kg CO_2 äq. mit einem *UnsI*-Wert von 86 % auf, womit das Ergebnis als vergleichsweise sehr unsicher einzuordnen ist. PS3 hingegen verfügt über ein *GWP*-Intervall von [75; 90] kg CO_2 äq. mit einem *UnsI*-Wert von 9 %, womit dieses Ergebnis als vergleichsweise sehr sicher eingestuft werden kann.

Da das Intervall von PS1 gänzlich unter dem Intervall von PS3 liegt, ist das Produktsystem PS1 dem Produktsystem PS3 vorzuziehen. PS2 ist ein *GWP*-Intervall von [20; 50] kg CO_2 äq. mit einem *UnsI*-Wert von 43 % zuzuordnen; das Ergebnis ist somit als vergleichsweise mäßig unsicher zu beurteilen. Die Ergebnisintervalle von PS1 und PS2 überlappen sich; es kann folglich keine Aussage darüber getroffen werden, welches der beiden Produktsysteme hinsichtlich des Treibhauspotentials niedrigere Umweltwirkungen aufweist. Da der *UnsI*-Wert des PS1 jedoch als vergleichsweise sehr unsicher einzustufen und damit kaum aussagekräftig ist, ist für dieses Produktsystem eine vertiefte Analyse der erfassten Daten erforderlich.

8.5.5 Methodenwahl nach Hansen

Hansen schlug unter Berücksichtigung der Rechenzeit eine Vorgehensweise für die Auswahl geeigneter Methoden vor (vgl. [64], S. 102 f.). Im Falle einer M-Matrix empfiehlt er das Gaußsche Eliminationsverfahren, ansonsten die Präkonditionierung des Systems. Ist der Vektor der rechten Seite ein Nullvektor bzw. betragen die Mittelpunkte der Intervalle null, soll das präkonditionierte System ebenfalls vorrangig mit dem Gaußschen Eliminationsverfahren



(a) : Ergebnisintervalle fiktiver Produktsysteme PS1, PS2 und PS3 vom *GWP*. (b) : Zugehörige *UnsI*-Werte.

Abbildung 8.9: *GWP*-Intervalle [kg CO_2 äq.] fiktiver Produktsysteme (a) sowie zugehöriger *UnsI*-Werte (b).

gelöst werden. Versagt dieses, wird das iterative Gauß-Seidel-Verfahren vorgeschlagen. Sind die Mittelpunkte des Vektors der rechten Seite ungleich Null, soll das Verfahren nach Hansen angewandt werden. Sofern dies nicht umsetzbar ist, wird ebenfalls auf das Gauß-Seidel-Verfahren verwiesen. In der Ökobilanzierung ist mindestens eine Komponente des Bedarfsvektors auf der rechten Seite ungleich Null. Iterative Verfahren werden in dieser Arbeit nicht berücksichtigt (Erläuterung hierzu vgl. Kapitel 3.7). Bei Produktmatrizen, die keiner M-Matrix entsprechen, sollte demnach für die Testreihen dieser Arbeit vorrangig das Verfahren von Hansen angewandt werden.

8.6 Testreihen zur Analyse und Verifikation

Die implementierten Methoden werden auf Basis von Testreihen analysiert und verifiziert. Der Vergleich basiert vorwiegend auf den Gauß-Verfahren *siga*, *sopt* und *salt*, den Adjunkten-Verfahren *adivaOrigin*, *adiheu* und *adiperm* sowie den Methoden *hansen*, *beeck*, *ning26* und *piga*. Der Aufbau und die Vorgehensweise dieser Testreihen wird nachfolgend erläutert. Die Analyse und Auswertung ausgewählter Testreihen erfolgt in Kapitel 10. Zur Einordnung der Unsicherheit der Ergebnisintervalle und zur vergleichenden Analyse der Ergebnisse wird der Unsicherheitsindex *UnsI* verwendet (Erläuterung zum Unsicherheitsindex vgl. Kapitel 8.5.4). Als Referenz dienen die Ergebnisse der vollständigen Szenarioanalyse.

8.6.1 Aufbau und Ziel der Testreihen

Es werden verschiedene Tests und Testreihen durchgeführt. Diese lassen sich wie folgt zusammenfassen:

- Parameterstudien zur Bewertung der Intervallweiten der Ergebnisse und Verifikation der Gleichungslöser
- Tests zur Verifikation einzelner Methoden
- Tests zur Erprobung des Rechenzeitaufwands

Mit den Parameterstudien soll zum einen überprüft werden, ob die Solver die Gültigkeitskriterien erfüllen (Definition der Gültigkeitskriterien vgl. Kapitel 8.4.2). Zum anderen sollen die Ergebnisse der Methoden miteinander verglichen werden. Hierbei soll auch analysiert werden, ob bestimmte Solver bei H-, M- oder Z-Matrixtypen als

besonders geeignet zu betrachten sind. Ebenso soll analysiert werden, ob es Solver gibt, die bei speziellen Produktsystemstrukturen schmalere Ergebnisintervalle liefern als andere. Dabei gilt es auch zu beachten, wie groß die Abweichungen der Näherungsverfahren zur Hülle der Lösungsmenge sind.

Die Testreihen setzen sich aus Parametervariationen an Produktsystemen mit drei Prozessmodulen zusammen. Es werden die Skalierungsvektoren der Produktsysteme berechnet sowie die *UnsI*-Werte der Ergebnisintervalle. Auf fiktive Elementarflüsse sowie eine Wirkungsabschätzung dieser wird verzichtet. Grund hierfür ist, dass zur Berechnung des Umweltvektors sowie des Wirkungsvektors (Definition von „Umweltvektor“ sowie „Wirkungsvektor“ vgl. Kapitel 5.4.2) festgelegte Intervalle lediglich miteinander multipliziert und aufaddiert werden. Diese Berechnungsschritte führen nicht zum Problem abhängiger Intervalle.

Neben den Parameterstudien gibt es weitere Tests zur Überprüfung der Funktionalität der Methoden. Die Methode *adisplit* zum Splitten der Produktmatrix wird anhand einzelner, größerer Produktmatrizen erprobt. Daneben werden automatisiert große Produktsysteme erzeugt, die Matrixdimensionen zwischen 100 und 3000 aufweisen. Sie dienen zur Erprobung des Rechenzeitaufwands diverser Methoden. Fragen, die unter anderem mithilfe der Tests geklärt werden sollen, sind die folgenden:

- Erfüllen die Ergebnisse aller Methoden die Anforderungen an das Berechnungsverfahren?
- Wie groß sind die Abweichungen der Ergebnisse zwischen den einzelnen Methoden?
- Erweisen sich einzelne Matrizen bei bestimmten Produktsystemstrukturen oder Matrixtypen bzw. Matrixdimensionen als besonders geeignet bzw. ungeeignet?
- Lassen sich die Methoden kombinieren oder eine Verfahren zur Wahl einzelner Methoden ableiten, mit der möglichst schmale Ergebnisintervalle unter Berücksichtigung der Rechenzeit kalkuliert werden?

Aus den Analysen soll dann ein allgemeines Verfahren abgeleitet werden, das unter Berücksichtigung des Zeitaufwands und der Produktsystemstrukturen die jeweils geeigneten Methoden bereitstellt.

8.6.2 Ausgangssituation der Parameterstudien

Im Rahmen der Parameterstudien werden fiktive Produktsysteme modelliert und systematisch eine Vielzahl von Variationen dieser Produktsysteme entwickelt. Die Grundstruktur der Produktsystem ist linear oder baumartig. Die Produktsysteme umfassen drei Prozessmodule, die auch als komprimierte (Sub-)Systeme aufgefasst werden können. Die geringe Anzahl von Prozessmodulen ermöglicht eine vollständige Szenarioanalyse der Produktsysteme. Es werden konstante Parameter definiert, welche die Grundstruktur der Produktsysteme festlegen. Für alle übrigen Matrixelemente werden weitere Parameter definiert, die variiert werden (Beschreibung der Parameter vgl. Kapitel 8.6.3, Tabelle 8.4). Die Vorzeichen der Parameter haben einen wesentlichen Einfluss auf die Systemstruktur der Produktsysteme. Daher werden die zu variierenden Parameter wie folgt bestimmt:

- Nullintervall (Begriffserklärung „Nullintervall“ vgl. Kapitel 3.1),
- negatives Intervall mit einer Intervallweite $\neq 0$,
- dessen positives Pendant.

Sie beschreiben zusätzliche Inputs bzw. zusätzlich erzeugte Koppelprodukte, die gemäß der Modellierungskriterien Intervallweiten ungleich Null aufweisen dürfen - sie werden daher als negative bzw. positive Intervalle mit

Intervallweiten ungleich Null definiert. Die Anzahl der möglichen, variierbaren Parameter P_{var} eines Produktsystems ergibt sich aus der Anzahl der Matrixelemente abzüglich der Output- und Inputelemente, die konstant gehalten werden. Für ein Produktsystem mit linearer oder baumartiger Grundstruktur und n Prozessmodulen kann sie wie folgt berechnet werden:

$$P_{var} = 3^{n^2-2n+1}$$

Die Variationen stellen weitere Produktsysteme dar. Durch zusätzliche Parameter, die keine Nullintervalle sind, wird die lineare bzw. baumartige Struktur gestört und es entstehen Hyper-Produktsysteme unterschiedlicher Art. Insgesamt werden in dieser Arbeit zehn Methoden und 162 Produktsysteme mit jeweils drei Prozessmodulen ausgewertet, woraus sich insgesamt 4860 Skalierungsfaktoren ergeben.

8.6.3 Systemstruktur der fiktiven Produktsysteme

In dieser Arbeit werden fiktive Produktsysteme in der Testreihe *3x3-linear* mit linearer Grundstruktur sowie der Testreihe *3x3-baum* mit einer baumartigen Grundstruktur und jeweils drei Prozessmodulen M_1, M_2 und M_3 erstellt. Das Hauptprodukt von Prozessmodul M_1 ist das Produkt P_1, das von Prozessmodul M_2 ist das Produkt P_2 und durch Prozessmodul M_3 entsteht das Produkt P_3. Es werden die Skalierungsfaktoren S_1, S_2 und S_3 berechnet und die Ergebnisse zur Verifikation verwendet. Der Skalierungsfaktor S_1 gibt dabei die notwendige Skalierung des Prozessmoduls M_1 wieder, analog dazu stellen die Skalierungsfaktoren S_2 und S_3 die erforderliche Skalierung der Prozessmodule M_2 und M_3 dar. Die Grundstruktur der Produktsysteme ist linear und baumartig. Hierfür werden die Parameter x bis z definiert. Sie sind in Tabelle 8.3 aufgelistet.

Tabelle 8.3: Konstante Parameter der Testreihen.

Parameter	**Intervall**
x	[-1.05 , -0.95]
y	[-1.1 , -0.9]
z	[-1.01, -0.99]

Die Parameter a_i bis d_i mit den Indizes i von $0,1,2$ werden innerhalb der einzelnen Testreihen variiert durch Zuweisung unterschiedlicher Werte bzw. Intervalle. Sie sind in Tabelle 8.4 aufgelistet.

Tabelle 8.4: Variierte Parameter der Testreihen.

Parameter	**i=0**	**i=1**	**i=2**
a_i	[0 , 0]	[-0.3, -0.1]	[0.1, 0.3]
b_i	[0 , 0]	[-0.4, -0.2]	[0.2, 0.4]
c_i	[0 , 0]	[-0.15, -0.05]	[0.05, 0.15]
d_i	[0 , 0]	[-0.35, -0.25]	[0.25, 0.35]

Eine lineare oder baumartige Struktur ist für diejenigen Produktsysteme gegeben, bei denen die Indizes der Parameter a_i bis d_i null sind. Ist dies nicht der Fall, ergeben sich andere Systemstrukturen, wie Produktsyste-

me mit Schleifen durch Verzweigung und/oder Zusammenfluss, Produktsysteme mit Pseudoschleifen und/oder pseudo-baumartiger Struktur sowie Mischstrukturen, in denen sowohl Schleifen als auch pseudoartige Strukturen vorzufinden sind. Schleifen durch Rezyklierung kommen nicht vor, da hierfür eine Veränderung der Grundstruktur erforderlich ist.

Abbildung 8.10: Produktsystem mit linearer Grundstruktur, variierende Inputs und Outputs in grau.

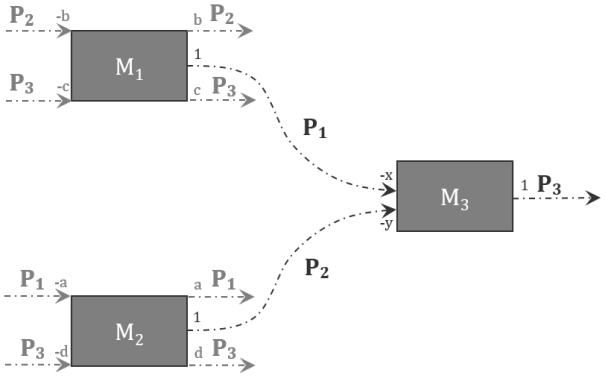


Abbildung 8.11: Produktsystem mit baumartiger Grundstruktur, variierende Inputs und Outputs in grau.

In Abbildung 8.10 ist das Produktsystem mit drei Prozessmodulen und linearer Grundstruktur dargestellt, wobei die variierenden Inputs und Outputs in grau dargestellt sind. Analog hierzu ist in Abbildung 8.11 ein Produktsystem mit baumartiger Grundstruktur dargestellt.

Tabelle 8.5: Grundstruktur der Produktsysteme mit drei Prozessmodulen.

Bezeichnung	**Produktmatrix**
3x3-linear	$\begin{array}{c} \\ \boldsymbol{P_1} \\ \boldsymbol{P_2} \\ \boldsymbol{P_3} \end{array} \begin{array}{c} \begin{array}{ccc} \boldsymbol{M_1} & \boldsymbol{M_2} & \boldsymbol{M_3} \end{array} \\ \begin{pmatrix} 1 & -x_1 & a_i \\ b_i & 1 & -y_1 \\ c_i & d_i & 1 \end{pmatrix} \end{array}$
3x3-baum	$\begin{array}{c} \\ \boldsymbol{P_1} \\ \boldsymbol{P_2} \\ \boldsymbol{P_3} \end{array} \begin{array}{c} \begin{array}{ccc} \boldsymbol{M_1} & \boldsymbol{M_2} & \boldsymbol{M_3} \end{array} \\ \begin{pmatrix} 1 & a_i & -x_1 \\ b_i & 1 & -y_1 \\ c_i & d_i & 1 \end{pmatrix} \end{array}$

In Tabelle 8.5 sind die entsprechenden Produktmatrizen der beiden Testreihen *3x3-linear* und *3x3-baum* aufgelistet. Insgesamt werden 4860 Ergebnisintervalle von zehn Methoden und 162 Produktsystemen in den beiden Testreihen *3x3-linear* und *3x3-baum* erzeugt.

8.6.4 Auswertungsmethoden

Die aus der vollständigen Szenarioanalyse ermittelten Ergebnisintervalle wurden zur Beurteilung der Gültigkeit der intervallbasierten Methoden herangezogen. Die Grenzwerte der über die Näherungsverfahren ermittelten Ergebnisse der Gleichungslöser dürfen die Untergrenze des Referenzintervalls *nicht überschreiten* und gleichzeitig die Obergrenze des Referenzintervalls *nicht unterschreiten*. Ansonsten sind sie für das in dieser Arbeit zu entwickelnde Verfahren ungeeignet. Durch Vergleich der Ergebnisintervalle wird identifiziert, welche Methoden die engsten Ergebnisintervalle liefern und wie sehr sie von der Hülle der Lösungsmenge abweichen.

Zur Beurteilung der Ergebnisse wird der Unsicherheitsindex *UnsI* verwendet. Er dient bei der Auswertung der Testreihen auch zum Vergleich der Ergebnisintervalle, ohne dass die tatsächlichen Intervallgrenzen der Resultate betrachtet werden müssen. Grund hierfür ist, dass alle Ergebnisintervalle die Hülle der Lösungsmenge umhüllen müssen; dies ist in den Gültigkeitskriterien (Gültigkeitskriterien vgl. Kapitel 8.4.2) festgelegt. Verfahren, welche dieses Kriterium nicht erfüllen, sind daher nicht für das in dieser Arbeit entwickelte Verfahren geeignet. Je höher der *UnsI*-Wert eines Ergebnisses im Vergleich zu einem anderen ist, desto größer ist die Oberhülle - damit werden in der Regel auch die Ergebnisintervalle der Ergebnisse mit niedrigeren *UnsI*-Werten umhüllt. Mitunter kann es aber auch Ausnahmen geben: so kann es sein, dass sich Intervalle der Oberhüllen auch überschneiden, nicht aber das Intervall, welches die Hülle der Lösungsmenge beschreibt. Zur besseren Lesbarkeit wird vorwiegend auf die Angabe des *UnsI* in Prozent verzichtet und dieser stattdessen als Dezimalzahl aufgeführt. Die Auswertung gleichartiger Produktsysteme aufgrund desselben Matrixtyps oder gleichartiger Prozessmodule aufgrund derselben Anordnung im Produktsystem erfolgt über das arithmetische Mittel und/oder den Median; die Abweichung zwischen den *UnsI*-Werten der Hüllen der Lösungsmengen und den Näherungsverfahren wird absolut und prozentual erfasst.

Kapitel 9

Java-basierte Implementierung

Die Implementierung des in dieser Arbeit vorgestellten Entwicklungsverfahrens beruht auf der Programmiersprache Java. Das Java-Paket heißt *Ivari* („**I**nter**v**all**ari**thmetik“) und enthält alle notwendigen Klassen und Methoden zur matrix- und intervallbasierten Berechnung. Die Erzeugung von intervallbasierten Skalaren, Vektoren und Matrizen werden über die Klassen IvariScalar, IvariVector und IvariMatrix ermöglicht. *Ivari* wird als .jar-File in das Ökobilanzierungsprogramm *MultiVaLCA* („*Multi Va*lue **L**ife **C**ycle **A**ssessment“) (Erläuterungen zu *MultiVaLCA* vgl. Kapitel 5.5.2) importiert, welches die grafische Benutzeroberfläche (engl. „**G**raphical **U**ser **I**nterface“, kurz GUI) für die Durchführung von Ökobilanzen bereitstellt.

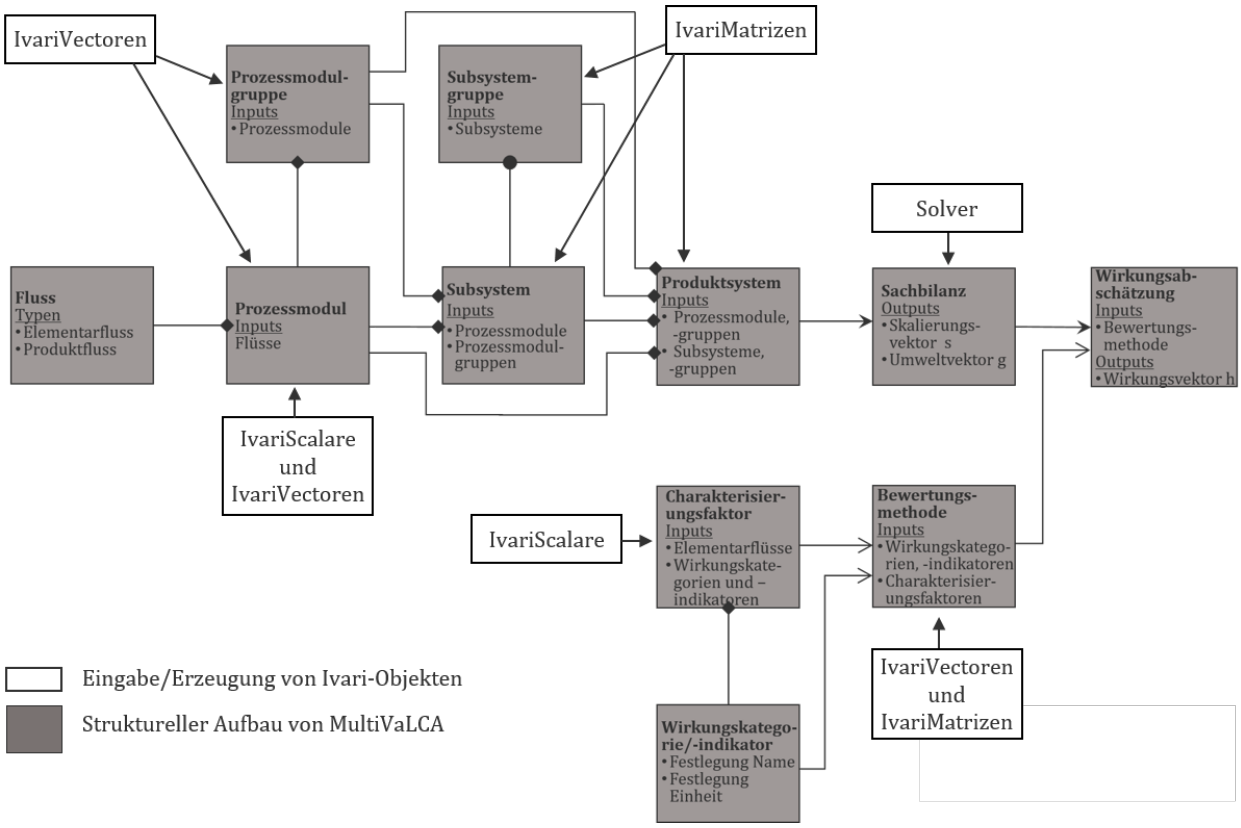


Abbildung 9.1: *Ivari*-Objekte werden an verschiedenen Stellen von *MultiVaLCA* eingegeben bzw. erstellt.

Die IvariScalare können über die GUI von *MultiVaLCA* bei der Quantifizierung von Flüssen und Charakterisierungsfaktoren eingegeben werden (vgl. auch Abbildung 9.1). Durch Zusammenfügen mehrerer Komponenten ergeben sich IvariVectoren und IvariMatrizen. Im Folgenden werden einige der durchgeführten Implementierungsarbeiten erläutert. Es werden Tests durchgeführt, um die Methoden und Klassen zu erproben. Beispiele hierzu sind im Anhang aufgeführt (vgl. Anhang A). In Kapitel 9.1 wird der grundlegende Aufbau des Java-Pakets erläutert. Kapitel 9.2 wird die Verwaltung der implementierten Berechnungsalgorithmen vorgestellt. Kapitel 9.3 enthält Informationen zu den Methoden, welche auf Basis einer vollständigen Szenarioanalyse die Hülle der Lösungsmenge ermitteln können. In Kapitel 9.4 werden die Implementierungsarbeiten weiterer Solver behandelt. Abschließend werden die Methoden zum Splitten der Matrix in Kapitel 9.5 vorgestellt.

9.1 Aufbau von Ivari

Das Paket *Ivari* besteht aus zwei Java-Packages. In einem Package sind diverse Klassen zur Definition von Intervall-Skalaren, Intervall-Vektoren und Intervall-Matrizen sowie die notwendigen arithmetischen Methoden implementiert. Das Paket *Ivari* verbindet das Matrix-Konstrukt, wie es im Java-Paket *JAMA* verfügbar ist, mit der Intervallarithmetik, welche durch diverse Methoden im Java-Paket *IAMath* unterstützt wird (Paket *IAMath* und Paket *JAMA* vgl. auch Kapitel 2.2.4, Abschnitt „Programmiersprache Java“). Das andere Package beinhaltet diverse Klassen mit Tests zur Verifikation der implementierten Methoden. Im folgenden Kapitel werden die grundlegenden Aspekte und Methoden der drei grundlegenden Klassen IvariScalar, IvariVector und IvariMatrix erläutert, welche die Basis des intervallbasierten Matrix-Konstrukts bilden (vgl. auch Abbildung 9.2).

9.1.1 IvariScalar.java

Die Klasse IvariScalar (vgl. Code 9.1) dient zur Erzeugung und dem Umgang mit Objekten, welche eine Intervall-Skalar repräsentieren. Sie enthält zwei Instanzvariablen: die Variable *lowerBound* (vgl. Code 9.1, Zeile 5) des primitiven Datentyps double, welche zur Definition der Untergrenze des Intervall-Skalars dient und analog *upperBound* (vgl. Code 9.1, Zeile 6) zur Beschreibung der Obergrenze des repräsentierten Intervalls. Es werden zwei Konstruktoren bereitgestellt: ein parameterloser Konstruktor (vgl. Code 9.1, Zeile 9 f.) und ein Konstruktor mit zwei Parametern (vgl. Code 9.1, Zeile 12 ff.) vom Typ double, durch die dem zu erzeugenden IvariScalar-Objekt direkt ein *lowerBound* und *upperBound* zugewiesen werden kann.

Code 9.1: Instanzvariablen und Konstruktoren von IvariScalar.java

```
1
2 public class IvariScalar {
3
4   // Instanzvariablen
5   private double lowerBound;
6   private double upperBound;
7
8   // Konstruktoren
9   public IvariScalar() {
10  }
11
12  public IvariScalar(double lowerBound, double upperBound) {
13    this.lowerBound = lowerBound;
14    this.upperBound = upperBound;
15  }
```

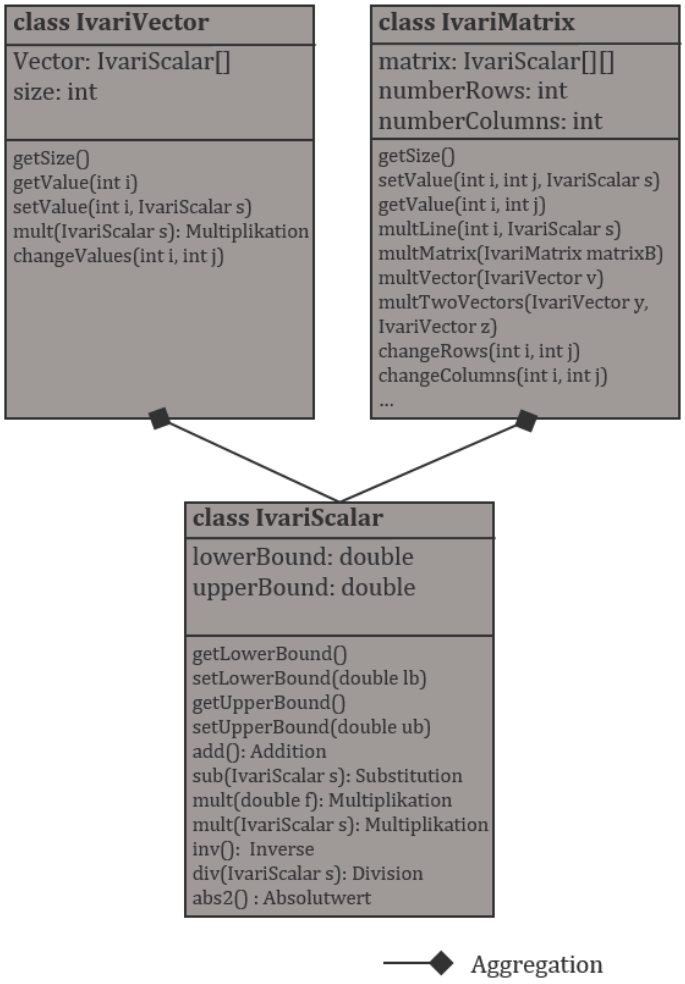


Abbildung 9.2: Die Klassen IvariScalar, IvariVector und IvariMatrix mit einzelnen Instanzvariablen und Methoden.

```
16   //...
17 }
```

Wesentliche Methoden der Klasse sind die grundlegenden intervallarithmetischen Operationen zur Addition, Subtraktion, Multiplikation und Division (Informationen zu intervallarithmetischen Operationen vgl. auch Kapitel 3.2). Der folgende Code (vgl. Code 9.2) zeigt exemplarisch die Berechnung der Inversen sowie der Division von Skalaren durch die Methoden *inv()* zur Kehrwertbildung (vgl. Code 9.2, Zeile 2 ff.) sowie *div(IvariScalar scalar)* zur Division (vgl. Code 9.2, Zeile 14 ff.), wobei durch beide ein Objekt des Typs IvariScalar zurückgegeben wird.

Code 9.2: Methoden inv() und div(IvariScalar scalar)

```
1 // Methode zur Kehrwertbildung eines IvariScalar
2 public IvariScalar inv() throws ArithmeticException {
3   IvariScalar result = new IvariScalar(1.,1.);
4   if (this.upperBound*this.lowerBound <= 0) {
5     throw new ArithmeticException("Fehler bei Kehrwertberechnung");
6   } else {
7     result.setLowerBound(1/this.upperBound);
8     result.setUpperBound(1/this.lowerBound);
9   }
10  return result;
11 }
12
13 // Methode zur Division des IvariScalars this durch das IvariScalar scalar
14 public IvariScalar div(IvariScalar scalar) throws ArithmeticException  {
15   IvariScalar result = this.mult(scalar.inv());
16   return result;
17 }
```

Die Kehrwertbildung ist nur zulässig, wenn die Ober- und Untergrenze eines Skalars keine unterschiedlichen Vorzeichen aufweisen, d. h. die Null nicht innerhalb der Intervallgrenzen liegt. Ansonsten wird der Programmablauf durch das Auslösen einer Exception beendet und in der Konsole wird die Fehlermeldung „Fehler bei Kehrwertberechnung“ (vgl. Code 9.2, Zeile 5) ausgegeben. Die Division erfolgt durch Multiplikation mit dem Kehrwert von IvariScalar *scalar*.

9.1.2 IvariVector.java

Die Klasse IvariVector (vgl. Code 9.3) dient zur Erzeugung und dem Umgang mit Objekten, welche einen Intervall-Vektor repräsentieren. Sie enthält zwei Instanzvariablen: die Variable *vector* vom Typ eines Arrays (vgl. Code 9.3, Zeile 4), welches Objekte vom Typ IvariScalar enthält, sowie die Variable *size* (vgl. Code 9.3, Zeile 5) vom primitiven Integer-Typ.

Code 9.3: Instanzvariablen und Konstruktoren von IvariVector.java

```
1 public class IvariVector {
2
3   // Instanzvariablen
4   private IvariScalar[] vector;
5   private int size;
6
7   // Konstruktoren
8   public IvariVector(int n) {
9     vector = new IvariScalar[n];
```

```
10      size = n;
11      for (int j=0; j<size; j++) {
12        vector[j] = new IvariScalar(.0, .0);
13      }
14    }
15
16    public IvariVector(IvariVector iVector) {
17      size = iVector.size;
18      vector = new IvariScalar[size];
19      for (int j=0; j<size; j++) {
20        vector[j] = iVector.getValue(j);
21      }
22    }
23
24    public IvariVector(double[] a, double[] b) {
25      size = a.length;
26      vector = new IvariScalar[size];
27      for (int i=0; i<size; i++) {
28        IvariScalar s = new IvariScalar(a[i], b[i]);
29        vector[i] = s;
30      }
31    }
32    //...
33  }
```

IvariVector enthält diverse Konstruktoren. Ein IvariVector kann mit einer bestimmten Dimension n erzeugt werden (vgl. Code 9.3, Zeile 8 ff.). Daneben kann der Konstruktor durch Eingabe eines Parameters vom Typ IvariVector aufgerufen werden (vgl. Code 9.3, Zeile 16 ff.). Eine weitere Möglichkeit zur Erzeugung eines Objekts des Typs IvariVector ist die Angabe zweier eindimensionaler Arrays double[] a sowie double[] b (vgl. Code 9.3, Zeile 24 ff.), wobei a die Untergrenzwerte und b die Obergrenzwerte des zu erzeugenden Objekts enthält.

Code 9.4: Methoden mult(IvariScalar s) und changeValues(`int i, int j`)

```
1  // Methode zur Multiplikation eines IvariVectors mit einem IvariScalar s
2  public IvariVector mult(IvariScalar scalar) {
3    IvariVector result = new IvariVector(size);
4    for (int i=0; i<size; i++) {
5      result.setValue(i,vector[i].mult(scalar));
6    }
7    return result;
8  }
9
10 // Methode zum Vertauschen zweier Zeilen i und j eines IvariVectors
11 public void changeValues(int i, int j) {
12   IvariScalar h = this.getValue(i);
13   this.setValue(i, this.getValue(j));
14   this.setValue(j, h);
15 }
```

Zwei wesentliche Methoden zur Änderung von Objekten des Typs IvariVector sind die Methode *mult(IvariScalar scalar)* und *changeValues(int i, int j)* (vgl. Code 9.4). Die Methode *mult(IvariScalar scalar)* (vgl. Code 9.4, Zeile 2 ff.) dient zur Multiplikation eines IvariVectors mit einem IvariScalar, zurückgegeben wird der resultierende IvariVector. Die Methode *changeValues(int i, int j)* (vgl. Code 9.4, Zeile 11 ff.) vertauscht die i-te Zeile eines IvariVectors mit der j-ten Zeile.

9.1.3 IvariMatrix.java

Die Klasse IvariMatrix (vgl. Code 9.5) dient zur Erzeugung und dem Umgang mit Objekten, welche eine Intervall-Matrix repräsentieren. Sie enthält drei Instanzvariablen: die Variable *matrix* vom Typ eines zweidimensionalen Arrays (vgl. Code 9.5, Zeile 4), welches Objekte vom Typ IvariScalar enthält, sowie die Variablen *numberRows* und *numberColumns* (vgl. Code 9.5, Zeile 5 und Zeile 6) vom primitiven Integer-Typ.

Code 9.5: Instanzvariablen und Konstruktoren von IvariMatrix.java

```
1  public class IvariMatrix {
2
3    //Instanzvariablen
4    private IvariScalar[][] matrix;
5    private int numberRows;
6    private int numberColumns;
7
8    //Konstruktoren
9    public IvariMatrix() {
10   }
11
12   public IvariMatrix(int n) {
13     matrix = new IvariScalar[n][n];
14     numberRows = n;
15     numberColumns = n;
16   }
17
18   public IvariMatrix(int i, int j) {
19     matrix = new IvariScalar[i][j];
20     numberRows = i;
21     numberColumns = j;
22   }
23
24   public IvariMatrix(double[][] a, double[][] b){
25     numberRows = a.length;
26     numberColumns = a[0].length;
27     matrix = new IvariScalar[numberRows][numberColumns];
28     for (int i=0; i<numberRows; i++) {
29       for (int j=0; j<numberColumns; j++) {
30         IvariScalar s = new IvariScalar(a[i][j], b[i][j]);
31         matrix[i][j] = s;
32       }
33     }
34   }
35   //...
36 }
```

IvariMatrix enthält diverse Konstruktoren. Ein Objekt vom Typ einer IvariMatrix kann mit einer bestimmten Dimension $\in \mathbb{IR}^{nxn}$ erzeugt werden (vgl. Code 9.5, Zeile 12 ff.); ein Objekt mit der Dimension mxn kann durch die Angabe zweier Integers generiert werden (vgl. Code 9.5, Zeile 18 ff.). Eine weitere Möglichkeit zur Erzeugung eines Objekts des Typs IvariMatrix ist die Angabe zweier Matrizen in Form von zweidimensionalen Arrays double[][] a sowie double[][] b (vgl. Code 9.5, Zeile 24 ff.), wobei a die Untergrenzwerte und b die Obergrenzwerte des zu erzeugenden IvariMatrix-Objekts enthält. Die grundlegenden Methoden der Klasse IvariMatrix dienen zur Durchführung arithmetischer Operationen sowie elementarer Matrixumformungen (vgl. Code 9.6).

Code 9.6: Methoden multMatrix(IvariMatrix matrixB), multVector(IvariVector vector), changeRows(int i, int j) und changeColumns(int i, int j)

```
1  // Methode zur Multiplikation einer IvariMatrix mit einer IvariMatrix matrixB
2  public IvariMatrix multMatrix(IvariMatrix matrixB) {
3    if (numberColumns != matrixB.getSize()) {
4      throw new ArithmeticException("Abweichende␣Dimensionen");
5    }
6    IvariMatrix result = new IvariMatrix(numberRows);
7    IvariScalar nullScalar = new IvariScalar(0,0);
8    for (int m = 0; m< numberRows; m++) {
9      for (int n = 0; n< numberRows; n++) {
10       result.setValue(m, n, nullScalar);
11     }
12   }
13   for (int k=0; k<numberRows; k++) {
14     for (int i=0; i<numberColumns; i++) {
15       for (int j = 0; j < numberRows; j++){
16         result.setValue(i,k, result.getValue(i,k).add(matrixB.getValue(i,j).mult(this.getValue(j,
               k))));
17       }
18     }
19   }
20   return result;
21 }
22
23
24 // Methode zur Multiplikation einer IvariMatrix this mit einem IvariVector vector
25 public IvariVector multVector(IvariVector vector) throws ArithmeticException {
26   if (numberColumns != vector.getSize()) {
27     throw new ArithmeticException("Abweichende␣Dimensionen");
28   }
29   IvariVector result = new IvariVector(numberRows);
30   for (int j=0; j<numberRows; j++) {
31     result.setValue(j, new IvariScalar(0., 0.));
32     for (int k=0; k<numberColumns; k++) {
33       result.setValue(j, result.getValue(j).add(getValue(j,k).mult(vector.getValue(k))));
34     }
35   }
36   return result;
37 }
38
39 // Methode zum Vertauschen zweier Zeilen einer IvariMatrix
40 public void changeRows(int i, int j) {
41   for (int k=0; k<numberRows; k++) {
42     IvariScalar h = this.getValue(i, k);
43     this.setValue(i, k, this.getValue(j, k));
44     this.setValue(j, k, h);
45   }
46 }
47
48 // Methode zum Vertauschen zweier Spalten einer IvariMatrix
49 public void changeColumns(int i, int j) {
50   for (int k=0; k<numberRows; k++) {
51     IvariScalar h = this.getValue(k, i);
```

```
52      this.setValue(k, i, this.getValue(k, j));
53      this.setValue(k, j, h);
54    }
55  }
```

Die Methode *multMatrix(IvariMatrix matrixB)* (vgl. Code 9.6, Zeile 2 ff.) dient zur Multiplikation zweier Objekte des Typs IvariMatrix, zurückgegeben wird die IvariMatrix *result*. Mit der Methode *multVector(IvariVector vector)* (vgl. Code 9.6, Zeile 25 ff.) wird ein IvariMatrix-Objekt mit einem Objekt des Typs IvariVector multipliziert, zurückgegeben wird der resultierende IvariVector *result*. Durch die Methoden *changeRows(int i, int j)* (vgl. Code 9.6, Zeile 40 ff.) bzw. *changeColumns(int i, int j)* (vgl. Code 9.6, Zeile 49 ff.) können die i-te Zeile bzw. Spalte einer IvariMatrix mit der j-ten Zeile bzw. Spalte vertauscht werden.

9.2 Verwaltung der implementierten Berechnungsalgorithmen

Es wurden verschiedene intervallbasierte Gleichungslöser programmiert. Die Verwaltung dieser erfolgt über die Enumeration Solver und die Methode *solve(Solver chosen, IvariVector vector)*, mit der die entsprechenden Gleichungslöser aufgerufen werden können (vgl. auch Abbildung 9.3).

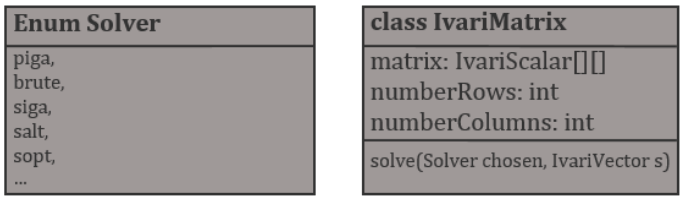


Abbildung 9.3: Die Enumeration Solver sowie die Methode zum Aufruf der Gleichungslöser in IvariMatrix.

9.2.1 Solver.java

Die Enumeration Solver (vgl. Code 9.7) dient als Aufzählungstyp zur Verwaltung der verschiedenen implementierten Algorithmen, mit denen das Intervall-Gleichungssystem gelöst werden kann.

Code 9.7: Enumeration Solver.java zur Verwaltung der Berechnungsalgorithmen

```
1  public enum Solver {
2    piga,
3    brute,
4    siga,
5    salt,
6    sopt,
7    adivaOrigin,
8    adiheu,
9    adiperm,
10   adisplit,
11   hansen,
12   rohn,
13   beeck,
14   ning22,
15   ning26,
16   picky,
```

```
17   quickpick,
18   quickily;
19 }
```

9.2.2 Methode solve(Solver chosen, IvariVector s)

Die Methoden zur Ausführung der entsprechenden Solver wurde in der Klasse IvariMatrix implementiert. Der ausgewählte Solver sowie der Bedarfsvektor vom Typ IvariVector werden der Methode *solve(Solver chosen, IvariVector s)* (vgl. Code 9.8) übergeben, wodurch dann die entsprechenden Methoden aufgerufen werden.

Code 9.8: Methode solve(Solver chosen, IvariVector s)

```
1 public IvariVector solve(Solver chosen, IvariVector s) throws Exception {
2   IvariVector result = new IvariVector(s);
3   switch (chosen) {
4   case piga:
5     result = gaussPlu2(s);
6     break;
7   case brute:
8     result = gaussPlus(s);
9     break;
10  case siga:
11    result = gauss(chosen, s);
12    break;
13  case salt:
14    result = gauss(chosen, s);
15    break;
16  case sopt:
17    result = gaussOpt(s);
18    break;
19  case adivaOrigin:
20    result = adivaOrigin(s);
21    break;
22  case adiheu:
23    result = adiheu(s);
24    break;
25  case adiperm:
26    result = adiperm(s);
27    break;
28  case adisplit:
29    result = adisplit(s);
30    break;
31  case hansen:
32    result = solverHansen(s);
33    break;
34  case rohn:
35    result = solverRohn(s);
36    break;
37  case beeck:
38    result = solverBeeck(s);
39    break;
40  case ning22:
41    result = solverNing22(s);
42    break;
```

```
43   case ning26:
44     result = solverNing26(s);
45     break;
46     case picky:
47     result = picky(s);
48     break;
49     case quickpick:
50     result = quickpick(s);
51     break;
52     case quickily:
53     result = quickily(s);
54     break;
55   default:
56     break;
57   }
58   ...
59   return result;
60 }
```

Die Auswahl der Methode erfolgt durch die Mehrfachverzweigung *switch* (vgl. Code 9.8, Zeile 3 ff.). Unabhängig von der Auswahl der Methoden verweisen die berechneten Ergebnisse allesamt auf den IvariVector *result*, der von der Methode zurückgegeben wird.

9.3 Solver brute und piga zur vollständigen Szenarioanalyse

Zur Verifikation der Intervall-Gleichungslöser dienen die Solver *piga* und *brute* (vgl. Code 9.7, Zeile 2 f.), mit denen eine vollständige Szenarioanalyse durchgeführt werden kann. Bei diesen Solvern kommen somit nicht die intervallarithmetischen Rechenoperationen (intervallarithmetische Rechenoperationen vgl. Kapitel 3.2) zur Anwendung. Der Solver *piga* stellt eine hinsichtlich der Rechenzeit verbesserte Variante zum Solver *brute* dar. Während *brute* ungeachtet von zusammenfallenden Intervallunter- und Intervallobergrenzen des Gleichungssystems und einhergehenden Intervallweiten von *0* alle Variationen berechnet und so auch Dopplungen zulässt, ermittelt *piga* derlei Intervalle und verzichtet hier auf eine Variation. Die Solver greifen zur Berechnung auf die Methoden *gaussPlus(IvariVector s)* (vgl. Kapitel 9.3.1), *gaussPlu2(IvariVector s)* (vgl. Kapitel 9.3.2) und *gaussPlusKern(double[][] matrixP1, double[] vectorP1)* (vgl. Kapitel 9.3.3) zurück.

9.3.1 Methode gaussPlus(IvariVector s)

Die Methode *gaussPlus(IvariVector s)* (vgl. Code 9.9) ist Teil der Klasse IvariMatrix (vgl. Kapitel 9.1.3). Zunächst werden alle Untergrenzwerte der zu variierenden IvariMatrix *this* sowie des zugehörigen IvariVectors *s* gespeichert im zweidimensionalen Array *matrixP* und im eindimensionalen Array *vectorP* (vgl. Code 9.9, Zeile 2 f.) vom primitiven Datentyp double (vgl. Code 9.9, Zeile 6 ff.). Durch Aufruf der Methode *gaussPlusKern(double[][] matrixP1, double[] vectorP1)* wird das sich ergebene Gleichungssystem gelöst und die Ergebnisse sowohl als Unter- als auch Obergrenzwert dem IvariVector *result* übergeben (vgl. Code 9.9, Zeile 12 ff.). Durch eine Schleife mit einem Schleifenzähler *k* von *0* bis zur Anzahl der Variationen $2^{(n+1)\cdot(n)}$ werden alle möglichen Variationen der Matrix *matrixP* sowie des Vektors *vectorP* gebildet (vgl. Code 9.9, Zeile 17 ff.). Dies erfolgt durch Teilung des Schleifenzählers *k* bei jedem Schleifendurchlauf durch *2*. Bleibt ein Rest, wird den Objekten der Obergrenzwert der IvariMatrix *this* bzw. des IvariVectors *s* übergeben, anderenfalls der Untergrenzwert (vgl. Code 9.9, Zeile 19 ff. und Code 9.9, Zeile 26 ff.)).

Das resultierende Gleichungssystem wird erneut berechnet (vgl. Code 9.9, Zeile 35). Unterschreiten die Ergebnisse die bisherigen Untergrenzwerte bzw. überschreiten die Ergebnisse die bisherigen Obergrenzwerte, werden dem IvariVector *result* die geringeren Werte als Untergrenzwert und die höheren Werte als Obergrenzwert übergeben (vgl. Code 9.9, Zeile 36 ff).

Code 9.9: Methode gaussPlus(IvariVector s)

```
1  private IvariVector gaussPlus(IvariVector s) {
2    double[][] matrixP = new double[numberRows][numberRows];
3    double[] vectorP = new double[numberRows];
4    double[] resultP = new double[numberRows];
5    IvariVector result = new IvariVector(numberRows);
6    for (int i=0; i<numberRows; i++) {
7      vectorP[i]=s.getValue(i).getLowerBound();
8      for (int j=0; j<spalAnz; j++) {
9        matrixP[i][j]=this.getValue(i,j).getLowerBound();
10     }
11   }
12   resultP = gaussPlusKern(matrixP, vectorP);
13     for (int i=0; i<numberRows; i++) {
14       IvariScalar is = new IvariScalar(resultP[i],resultP[i]);
15       result.setValue(i,is);
16     }
17     for (int k = 0; k < Math.pow(2,numberRows*(numberRows+1)); k++) {
18       int k1 = k; int k2 = 0;
19     for (int i=0; i<numberRows; i++) {
20       k2 = k1 % 2; k1 = k1 / 2;
21       if (k2 == 0) {
22         vectorP[i]=s.getValue(i).getLowerBound();
23       } else {
24         vectorP[i]=s.getValue(i).getUpperBound();
25       }
26       for (int j=0; j<spalAnz; j++) {
27         k2 = k1 % 2; k1 = k1 / 2;
28         if (k2 == 0) {
29           matrixP[i][j]=this.getValue(i,j).getLowerBound();
30         } else {
31           matrixP[i][j]=this.getValue(i,j).getUpperBound();
32         }
33       }
34     }
35     resultP = gaussPlusKern(matrixP, vectorP);
36       for (int i=0; i<numberRows; i++) {
37           IvariScalar is = result.getValue(i);
38           if (resultP[i] < is.getLowerBound()) {
39             is.setLowerBound(resultP[i]);
40           }
41           if (resultP[i] > is.getUpperBound()) {
42             is.setUpperBound(resultP[i]);
43           }
44           result.setValue(i,is);
45       }
46   }
47   return result;
```

```
48 }
```

Die Methode gibt somit die minimalen Ergebnisse aller Variationen als Untergrenzwerte und die maximalen Ergebnisse aller Variationen als Obergrenzwerte, die im IvariVector *result* gespeichert wurden, zurück (vgl. Code 9.9, Zeile 47). Der Algorithmus wurde in der Methode *gaussPlu2(IvariVector s)* verbessert, sodass nur Intervalle mit Intervallweiten $\neq 0$ berücksichtigt werden, wodurch die Anzahl der zu berechnenden Variationen reduziert werden kann.

9.3.2 Methode gaussPlu2(IvariVector s)

Die Methode *gaussPlu2(IvariVector s)* stellt die optimierte Variante der Methode *gaussPlus(IvariVector s)* dar. Dafür wird ein Objekt vom Typ eines Arrays mit der Dimension $\in \mathbb{R}^{((n)*(n+1)x2)}$ instanziiert (vgl. Code 9.10, Zeile 2) wobei n der Zeilenanzahl der betrachteten Intervallmatrix *this* und des zugehörigen Intervallvektors s entspricht (vgl. Kapitel 9.1.3). Zusätzlich wird ein Zähler *swapCounter* vom primitiven Datentyp Integer eingeführt (vgl. Code 9.10, Zeile 3), um die Eintragung der *swapList* zu verwalten.

Wenn sich die einzelnen unteren und oberen Grenzwerte der durchsuchten Intervalle um 0,01 % unterscheiden, werden diese in den Objekten *vectorP* sowie *matrixP* gespeichert, anderenfalls wird der Mittelwert dieser eingetragen (vgl. Code 9.10, Zeile 8 ff.). Im ersten Fall wird die Zeilennummer des IvariVectors s in der ersten Spalte der *swapList* eingetragen und der Wert *-1* in der zweiten Spalte (vgl. Code 9.10, Zeile 16 f.) - beim IvariMatrix-Objekt *this* wird analog in der ersten Spalte die Zeilennummer übergeben und in der zweiten Spalte die zugehörige Spaltennummer des aktuell abgefragten Matrixelements (vgl. Code 9.10, Zeile 31 f.). Nach jedem Eintrag in der *swapList* wird der Zähler *swapCounter* um *1* erhöht (vgl. Code 9.10, Zeile 18 und Code 9.10, Zeile 33).

Code 9.10: Methode gaussPlu2(IvariVector s)

```
1  private IvariVector gaussPlu2(IvariVector s){
2    int[][] swapList = new int[numberRows*(numberRows+1)][2];
3    int swapCounter = 0;
4    double[][] matrixP = new double[numberRows][numberRows];
5    double[] vectorP = new double[numberRows];
6    double[] resultP = new double[numberRows];
7    IvariVector result = new IvariVector(numberRows);
8    for (int i=0; i<numberRows; i++) {
9        if ((s.getValue(i).getLowerBound() > 0.
10         && s.getValue(i).getLowerBound()
11           < s.getValue(i).getUpperBound() * .9999)
12         || (s.getValue(i).getLowerBound() < 0.
13         && s.getValue(i).getLowerBound()
14           < s.getValue(i).getUpperBound() * 1.0001)){
15         vectorP[i]=s.getValue(i).getLowerBound();
16         swapList[swapCounter][0]=i;
17         swapList[swapCounter][1]=-1;
18         swapCounter++;
19       } else {
20         vectorP[i]=(s.getValue(i).getLowerBound()
21           + s.getValue(i).getUpperBound())/2;
22       }
23     for (int j=0; j<numberColumns; j++){
24       if ((this.getValue(i,j).getLowerBound() > 0.
25         &&  this.getValue(i,j).getLowerBound()
26           < this.getValue(i,j).getUpperBound()*.9999)
```

```
27          || (this.getValue(i,j).getLowerBound() < 0.
28            && this.getValue(i,j).getLowerBound()
29              <this.getValue(i,j).getUpperBound()*1.0001)){
30          matrixP[i][j]=this.getValue(i,j).getLowerBound();
31          swapList[swapCounter][0]=i;
32              swapList[swapCounter][1]=j;
33              swapCounter++;
34        } else {
35          matrixP[i][j]=(this.getValue(i,j).getLowerBound()
36            + this.getValue(i,j).getUpperBound())/2;
37        }
38      }
39    }
40    resultP = gaussPlusKern(matrixP, vectorP);
41      for (int i=0; i<numberRows; i++) {
42          IvariScalar is = new IvariScalar(resultP[i],resultP[i]);
43          result.setValue(i,is);
44      }
45    for (int k = 0; k < Math.pow(2,swapCounter); k++){
46      int k1 = k; int k2 = 0;
47      for (int i=0; i<swapCounter; i++) {
48        k2 = k1 % 2; k1 = k1 / 2;
49        if (k2 == 0) {
50          if (swapList[i][1] == -1) {
51            vectorP[swapList[i][0]]
52              = s.getValue(swapList[i][0]).getLowerBound();
53          } else {
54            matrixP[swapList[i][0]][swapList[i][1]]
55              = this.getValue(swapList[i][0],swapList[i][1]).getLowerBound();
56          }
57        } else {
58          if (swapList[i][1] == -1) {
59            vectorP[swapList[i][0]]
60              = s.getValue(swapList[i][0]).getUpperBound();
61          } else {
62            matrixP[swapList[i][0]][swapList[i][1]]
63              = this.getValue(swapList[i][0],swapList[i][1]).getUpperBound();
64          }
65        }
66      }
67      resultP = gaussPlusKern(matrixP, vectorP);
68          for (int i=0; i<numberRows; i++) {
69            IvariScalar is = result.getValue(i);
70            if (resultP[i] < is.getLowerBound()) {
71              is.setLowerBound(resultP[i]);
72            }
73            if (resultP[i] > is.getUpperBound()) {
74              is.setUpperBound(resultP[i]);
75            }
76            result.setValue(i,is);
77          }
78    }
79    return result;
80  }
```

Anschließend wird analog zur Methode *gaussPlus(IvariVector s)* (vgl. Kapitel 9.3.1) die erste Variation von *matrixP* und *vectorP* berechnet und dem IvariVector *result* die Ergebnisse übergeben (vgl. Code 9.10, Zeile 40). Die Erstellung aller weiterer Variationen erfolgt ebenfalls analog zur Methode *gaussPlus(IvariVector s)* (vgl. Kapitel 9.3.1), wobei der Schleifenzähler *k* nur von *0* bis zur Anzahl der eingetragenen Elemente der *swapList* läuft - womit insgesamt weniger Schleifen durchgeführt werden als bei der Methode *gaussPlus(IvariVector s)*. Ergibt die Division durch *2* keinen Rest, werden erneut die unteren Grenzwerte in die Objekte *vectorP* und *matrixP* eingetragen, anderenfalls die oberen Grenzwerte. Zusätzlich wurde eine Verzweigung eingeführt. Dies bezweckt, dass durch die Abfrage der zweiten *swapList* und den Wert *-1* erkannt wird, ob die Eintragung dem IvariVector *s* oder der IvariMatrix *this* zugeordnet ist (vgl. Code 9.10, Zeile 50 ff. und Code 9.10, Zeile 58 ff.).

Die so erstellten Variationen werden erneut berechnet (vgl. Code 9.10, Zeile 67) mit der Methode *gaussplusKern* (vgl. folgendes Kapitel 9.3.3). Dabei werden gleichsam zur Methode *gaussPlus* (vgl. Kapitel 9.3.1 die Werte des IvariVectors *result* überschrieben, wenn die jeweils berechneten und in *resultP* gespeicherten Ergebnisse die bisherigen Untergrenzwerte unterschreiten bzw. die aktuellen Obergrenzwerte überschreiten (vgl. Code 9.9, Zeile 12 ff). Die Methode gibt die Minima und Maxima aller Variationen durch Rückgabe des IvariVectors *result* zurück (vgl. Code 9.9, Zeile 79).

9.3.3 Methode gaussPlusKern(double[ ][ ] matrixP1, double[ ] vectorP1)

Das Verfahren der Methode *gaussPlusKern(double[][] matrixP1, double[] vectorP1)* (vgl. Code 9.11) entspricht im Kern der intervallbasierten Methode *gauss(Solver chosen, IvariVector s)* (vgl. Code 9.12), ohne die intervallarithmetischen Rechenregeln anzuwenden. Die elementaren Umformungen basieren auf partieller Pivotisierung, bei dem als Pivotelement der betragsmäßig größte Eintrag *matrixP[max][p]* einer Spalte für die Hauptdiagonale gesucht wird. In der lokalen Variable *max* vom primitiven Datentyp Integer wird die Zeilennummer des aktuell größten Matrixelements gespeichert. Dabei wird zu Beginn die Variable *max* mit der Zeilennummer des Elements der Hauptdiagonalen belegt (vgl. Code 9.11, Zeile 13).

Code 9.11: Methode gaussPlusKern (double[ ][ ] matrixP1, double[ ] vectorP1)

```
1  private double[] gaussPlusKern(double[][] matrixP1, double[] vectorP1) {
2    double[][] matrixP = new double[numberRows][numberRows];
3    double[] vectorP = new double[numberRows];
4    for (int i = 0; i < numberRows; i++) {
5      vectorP[i] = vectorP1[i];
6      for (int j = 0; j < numberRows; j++) {
7        matrixP[i][j] = matrixP1[i][j];
8      }
9    }
10   final double EPSILON = 1e-10;
11   double[] resultP = new double[numberRows];
12   for (int p = 0; p < numberRows; p++) {
13     int max = p;
14       for (int i = p + 1; i < numberRows; i++) {
15         if (Math.abs(matrixP[i][p]) > Math.abs(matrixP[max][p])) {
16             max = i;
17       }
18     }
19       double[] temp = matrixP[p];
20       matrixP[p] = matrixP[max];
21     matrixP[max] = temp;
```

```
22      double t = vectorP[p];
23      vectorP[p] = vectorP[max];
24      vectorP[max] = t;
25      if (Math.abs(matrixP[p][p]) <= EPSILON){
26          throw new ArithmeticException("Matrix singular or nearly singular");
27        }
28      for (int i = p + 1; i < numberRows; i++) {
29          double alpha = matrixP[i][p] / matrixP[p][p];
30            vectorP[i] -= alpha * vectorP[p];
31            for (int j = p; j < numberRows; j++) {
32              matrixP[i][j] -= alpha * matrixP[p][j];
33          }
34        }
35    }
36      for (int i = numberRows - 1; i >= 0; i--) {
37        double sum = 0.0;
38        for (int j = i + 1; j < numberRows; j++) {
39          sum += matrixP[i][j] * resultP[j];
40        }
41        resultP[i] = (vectorP[i] - sum) / matrixP[i][i];
42    }
43    return resultP;
44  }
```

Die Durchsuchung erfolgt spaltenweise und beginnt beim Matrixelement $a_{(p+1)p}$, wobei die p-te Spalte vom Matrixelement unterhalb der Hauptdiagonalen bis zur n-ten Zeile durchsucht wird (vgl. Code 9.11, Zeile12 ff.). Ggfs. werden die Zeilen vertauscht (vgl. Code 9.11, Zeile 19 ff.) und überzählige Variablen eliminiert (vgl. Code 9.11, Zeile 28 ff.). Falls der Wert des Elements der Hauptdiagonalen nahe bei *0* liegt, wird eine Exception ausgelöst und eine Fehlermeldung ausgegeben vgl. Code 9.11, Zeile 25 f.). Anschließend erfolgt die Berechnung der Lösung durch rückwärtiges Einsetzen (vgl. Code 9.11, Zeile 36 ff.), wobei die Ergebnisse durch das Objekt *resultP* vom Typ eines eindimensionalen Arrays zurückgegeben werden (vgl. Code 9.12, Zeile37 ff.).

9.4 Implementierung diverser Gleichungslöser

Im Folgenden werden die implementierten Gleichungslöser auf Basis des intervallbasierten gaußschen Eliminationsverfahrens in Kapitel 9.4.1 sowie auf Basis der Inversenberechnung über die Intervall-Adjunkten in Kapitel 9.4.2 vorgestellt. Daneben wurden weitere Gleichungslöser implementiert, wie z. B. die Solver *hansen, beeck, ning22* und *ning26*. Sie stellen die implementierten Verfahren von Hansen, Beeck und Ning dar (Verfahren vgl. Kapitel 3.7.4, 3.7.3 und 3.7.6). Die Implementierungen dieser Methoden sind im Anhang (vgl. Anhang A) zu finden.

9.4.1 Solver auf Basis des Intervall-Gauß-Verfahrens

Die Solver *siga, salt* und *sopt* basieren auf dem Intervall-Gauß-Verfahren. Die Implementierung beruht auf den Grundsätzen des Codes von [34], welche auf die Intervallebene übertragen wurden. Im Folgenden werden die verwendeten Methoden der Solver *siga, salt* und *sopt* vorgestellt. Die Solver *siga* und *salt* kommen durch die Methode *gauss(Solver chosen, IvariVector s)* (vgl. Code 9.12) zur Ausführung. Der Solver *sopt* (vgl. Code 9.7, Zeile 6) wird durch die Methode *gaussOpt(IvariVector s)* (vgl. Code 9.16) ausgeführt; er bedient sich der beiden Gleichungslöser *siga* und *salt* und gibt als Lösung den jeweils höchsten Intervalluntergrenzwert sowie den niedrigsten Intervallobergrenzwert aus.

Methode gauss(Solver chosen, IvariVector s)

Die elementaren Umformungen der Methode *gauss(Solver chosen, IvariVector s)* (vgl. Code 9.12) basieren auf partieller Pivotisierung, bei dem als Pivotelement der betragsmäßig größte Eintrag einer Spalte für die Hauptdiagonale gesucht wird, wobei die Durchsuchung beim Matrixelement a_{kk} der Hauptdiagonalen beginnt und dann die weiteren Einträge der k-ten Spalte bis zur n-ten Zeile durchsucht werden (vgl. Code 9.12, Zeile15 ff.). Ggfs. werden die Zeilen vertauscht (vgl. Code 9.12, Zeile 22 f.) und überzählige Variablen eliminiert (vgl. Code 9.12, Zeile 24 ff.). Anschließend erfolgt die Berechnung der Lösung durch rückwärtiges Einsetzen (vgl. Code 9.12, Zeile 32 ff.), wobei die Ergebnisse durch das Objekt *result* vom Typ IvariVector zurückgegeben werden (vgl. Code 9.12, Zeile 37 ff.).

Code 9.12: Methode gauss(Solver chosen, IvariVector s)

```
1 private IvariVector gauss(Solver chosen, IvariVector s) throws Exception {
2   LinkedList<Integer> swapList = new LinkedList<Integer>();
3   switch (chosen) {
4     case siga:
5       swapList = heuSort(s);
6       break;
7     case salt:
8       swapList = heuSort2(s);
9       break;
10    default:
11      break;
12    }
13  }
14  IvariVector result = new IvariVector(numberRows);
15  for (int pivot = 0; pivot < numberRows; pivot++) {
16      int max = pivot;
17        for (int i = pivot + 1; i < numberRows; i++) {
18          if (getValue(i,pivot).abs2() > getValue(max,pivot).abs2()) {
19              max = i;
20            }
21        }
22        changeRows(pivot, max);
23        s.changeValues(pivot, max);
24        for (int i = pivot + 1; i < numberRows; i++) {
25          IvariScalar alpha = getValue(i,pivot).div(getValue(pivot,pivot));
26            s.setValue(i, s.getValue(i).sub(alpha.mult(s.getValue(pivot))));
27            for (int j = pivot + 1; j < numberRows; j++) {
28              setValue(i,j, getValue(i,j).sub(alpha.mult(getValue(pivot,j))));
29            }
30        }
31  }
32    for (int i = numberRows - 1; i >= 0; i--) {
33      IvariScalar sum = new IvariScalar(.0, .0);
34        for (int j = i; j < numberRows; j++) {
35          sum = sum.add(getValue(i,j).mult(result.getValue(j)));
36        }
37        result.setValue(i, s.getValue(i).sub(sum).div(getValue(i,i)));
38    }
39    result = heuBack(result, swapList);
40  return result;
41 }
```

Vor der Vorwärts- und Rückwärtselimination wird die IvariMatrix *matrix* vorsortiert. Die Vorsortierung erfolgt durch den Aufruf der Methode *heuSort(IvariVector s)* bzw. *heuSort2(IvariVector s)* - je nach Wahl des Solvers *siga* bzw. *salt* (vgl. Code 9.12, Zeile4 ff.). Im ersteren Fall wird die IvariMatrix *matrix* derart umsortiert, dass diese - soweit möglich - einer oberen Dreiecksmatrix entspricht, im letzteren Fall wird die IvariMatrix *matrix* derart umsortiert, sodass sich unterhalb der Hauptdiagonalen - soweit möglich - nur Intervalle mit negativen Vorzeichen oder Nullelemente befinden. Die notwendige Rücksortierung erfolgt nach Durchführung des Intervall-Gauß-Verfahrens durch Aufruf der Methode *heuBack(IvariVector vector, LinkedList<Integer> list)* (vgl. Code 9.12, Zeile39). Die beiden Methoden werden in Code 9.13 sowie Code 9.15 dargestellt.

Methoden heuSort(IvariVector s) und heuSort2 (IvariVector s)

Mit der Methode *heuSort(IvariVector s)* (vgl. Code 9.13) kann die IvariMatrix *this* mit einer Dimension $\in \mathbb{IR}^{nxn}$ sowie der als Parameter übergebene IvariVector *s* umgeformt werden, sodass die IvariMatrix - soweit möglich - einer oberen Dreiecksmatrix entspricht. Dafür werden zeilenweise die Nullelemente gezählt (vgl. Code 9.13, Zeile 9 f.) und die Spaltennummer des zuletzt detektierten Nichtnullelements gespeichert (vgl. Code 9.13, Zeile 12 f.). Der Suchalgorithmus läuft in Schleifen nach absteigender Anzahl an Nullelementen (vgl. Code 9.13, Zeile 5 ff.): zunächst wird eine Gleichung mit *n-1* Nullelementen gesucht, wird diese gefunden, wird sie mit der Gleichung in der *n*-ten Zeile getauscht (vgl. Code 9.13, Zeile 16 f.) - ist das Nichtnullelement nicht in der passenden - hier *n*-ten - Spalte, werden die Spalten dementsprechend getauscht (vgl. Code 9.13, Zeile 19 f.).

Es folgt ein Abbruch der inneren Schleife (vgl. Code 9.13, Zeile 24), d. h. der weiteren Spaltendurchsuchung. Fortgesetzt wird mit der nächsten Zeile, bei der nach $n-2$ Nullelementen gesucht wird und bei Erfolg diese mit der Gleichung der vorletzten Zeile getauscht wird.

Code 9.13: Methode heuSort(IvariVector s)

```
1  private LinkedList<Integer> heuSort(IvariVector s) {
2    LinkedList<Integer> swapList = new LinkedList<Integer>();
3    int numberZeros = 0;
4    int notNullColumn = 0;
5    for (int i = numberRows; i>0; i--) {
6      for (int j = 0; j<i; j++) {
7        numberZeros=0;
8        for (int k = 0; k<i; k++) {
9          if (getValue(j,k).getLowerBound()==0) {
10           numberZeros++;
11         }
12         else {
13           notNullColumn=k;
14         }
15       }
16       if (numberZeros>=i-1){
17         changeRows(j,i-1);
18         s.changeValues(j,i-1);
19         if (notNullColumn != i-1) {
20           changeColumns(notNullColumn,i-1);
21           swapList.add(notNullColumn);
22           swapList.add(i-1);
23         }
24         break;
25       }
```

```
26      }
27    }
28    return swapList;
29  }
```

Der Sortieralgorithmus *heuSort2(IvariVector s)* (vgl. Code 9.14) entspricht im Grundsatz *heuSort(IvariVector s)*, mit dem Unterschied, dass nicht nur Nullintervalle, sondern auch Intervalle mit negativem Vorzeichen gesucht werden (vgl. Code 9.14, Zeile 7). Dadurch soll die untere Dreiecksmatrix möglichst mit Nullintervallen sowie Intervallen mit negativen Vorzeichen besetzt sein.

Code 9.14: Methode heuSort2(IvariVector s)

```
1  private LinkedList<Integer> heuSort(IvariVector s) {
2    //...
3    for (int i = numberRows; i>0; i--) {
4      for (int j = 0; j<i; j++) {
5        numberZeros=0;
6        for (int k = 0; k<i; k++) {
7          if (getValue(j,k).getLowerBound()<=0) {
8            numberZeros++;
9          }
10         //...
11       }
12     }
13   }
14   return swapList;
15 }
```

Bei beiden Methoden ist aufgrund der Spaltenvertauschungen eine Rücksubstitution erforderlich, da diese Auswirkungen auf die Elementenanordung des resultierenden Ergebnisvektors hat. Daher werden die vertauschten Spaltennummern in der *swapList* gespeichert (vgl. Code 9.13, Zeile 21 f.), welche als doppelt verkettete Liste vom Typ einer LinkedList<Integer> (vgl. Code 9.13, Zeile 2) instanziiert wurde. Sie wird von der Methode *heuSort(IvariVector s)* bzw. *heuSort2(IvariVector s)* zurückgegeben (vgl. Code 9.13, Zeile 28 und Code 9.14, Zeile 14).

Methode heuBack(IvariVector vector, LinkedList<Integer> list)

Durch die Methode *heuBack(IvariVector vector, LinkedList<Integer> list)* (vgl. Code 9.15) werden die Intervalle, die als IvariVector zurückgegeben werden, wieder an die korrekte Position zurückgesetzt (vgl. Code 9.15, Zeile 5 f.). Der zu bearbeitende IvariVector sowie die *swapList* werden der Methode als Parameter übergeben.

Code 9.15: Methode heuBack(IvariVector vector, LinkedList<Integer> list)

```
1  private IvariVector heuBack(IvariVector vector, LinkedList<Integer> list) {
2    for (int i=1; i<=list.size()/2; i++) {
3      int z = list.get(list.size() - 2*i);
4      int y = list.get(list.size() + 1 - 2*i);
5      changeColumns(y, z);
6      vector.changeValues(y, z);
7    }
8    return vector;
9  }
```

Rückgabewert der Methode ist der resultierende IvariVector *vector* (vgl. Code 9.15, Zeile 8).

Methode gaussOpt(IvariVector s)

Durch die Methode *gaussOpt(IvariVector s)* (vgl. Code 9.16) wird der größte Untergrenzwert bzw. der kleinste Obergrenzwert der beiden Solver *siga* und *salt* gesucht und zurückgegeben. Dabei wird das Gleichungssystem mit beiden Solvern gelöst und durch die Referenzvariablen *result1* bzw. *result2* bereitgestellt (vgl. Code 9.16, Zeile 7 und Code 9.16, Zeile 19). Etwaige Fehler, die im Zuge der Berechnung auftreten, werden abgefangen und der Boolean *r1exists* für den Algorithmus des Solvers *siga* bzw. *r2exists* für den Algorithmus des Solvers *salt* von *true* auf *false* gesetzt (vgl. Code 9.16, Zeile 6 ff. und Code 9.16, Zeile 18 ff.).

Code 9.16: Methode gaussOpt(IvariVector s)

```
1  private IvariVector gaussOpt(IvariVector s) {
2    IvariMatrix matrix = new IvariMatrix(this);
3    IvariVector result = new IvariVector(numberRows);
4    IvariVector result1 = new IvariVector(numberRows);
5    boolean r1exists=true;
6    try {
7      result1 = gauss(Solver.siga, s);
8    } catch (Exception e) {
9      r1exists=false;
10   }
11   for (int i=0; i<result.getSize(); i++) {
12     for (int j=0; j<result.getSize(); j++) {
13       this.setValue(i,j,matrix.getValue(i,j));
14     }
15   }
16   IvariVector result2 = new IvariVector(numberRows);
17   boolean r2exists=true;
18   try {
19     result2 = gauss(Solver.salt, s);
20   } catch (Exception e) {
21     r2exists=false;
22   }
23   for (int i=0; i<result.getSize(); i++) {
24     IvariScalar sca = new IvariScalar();
25     Double result1low = result1.getValue(i).getLowerBound();
26     Double result1upp = result1.getValue(i).getUpperBound();
27     Double result2low = result2.getValue(i).getLowerBound();
28     Double result2upp = result2.getValue(i).getUpperBound();
29     if ((r1exists==false)^(r2exists==false)) {
30       if (r1exists==false) {
31         result1low=Double.MIN_VALUE;
32         result1upp=Double.MAX_VALUE;
33       }
34       if (r2exists==false) {
35         result2low=Double.MIN_VALUE;
36         result2upp=Double.MAX_VALUE;
37       }
38     }
39     if ((r1exists==true)(r2exists==true)) {
40       if (result1low<result2low) {
41         sca.setLowerBound(result2low);
42       } else {
43         sca.setLowerBound(result1low);
```

```
44        }
45        if (result1upp>result2upp) {
46          sca.setUpperBound(result2upp);
47        } else {
48          sca.setUpperBound(result1upp);
49        }
50        result.setValue(i,sca);
51      } else {
52        result=null;
53      }
54    }
55    return result;
56  }
```

Die Unter- bzw. Obergrenzwerte der Ergebnisse des Solvers *siga* werden zeilenweise den Objekten *result1low* bzw. *result1upp* vom Typ der Wrapper-Klasse Double übergeben, die Ergebnisse des Solvers *salt* gleichsam den Objekten *result2low* und *result2upp* (vgl. Code 9.16, Zeile 23 ff.). Wenn bei einem der beiden Solver eine Exception ausgelöst wird, werden den Untergrenzwerten das kleinste Element der Wrapper-Klasse, die Konstante MIN_VALUE, übergeben und analog den Obergrenzwerten das größte Element, die Konstante MAX_VALUE, zugewiesen (vgl. Code 9.16, Zeile 29 ff.). Wird bei beiden Solvern eine Exception ausgelöst, wird der IvariVector *result* dereferenziert (vgl. Code 9.16, Zeile 52) Anschließend wird über den IvariVector *result* (vgl. Code 9.16, Zeile 55) der höhere Untergrenzwert sowie der niedrigere Obergrenzwert der beiden Solver übergeben (vgl. Code 9.16, Zeile 39 ff.).

9.4.2 Solver auf Basis des Intervall-Adjunkten-Verfahrens

Die Solver *adivaOrigin*, *adiheu* und *adiperm* beruhen auf dem Intervall-Adjunkten-Verfahren. Die Implementierung des Adjunkten-Verfahrens beruht auf den Grundsätzen des Codes von [36], welcher auf die Intervallebene übertragen wurde. Im Folgenden werden die Methoden der Solver *adivaOrigin*, *adiheu* und *adiperm* vorgestellt. Der Solver *adivaOrigin* (vgl. Code 9.17) ermittelt über die adjunkten Matrix die Inverse. Der Solver *adiheu* (vgl. Code 9.18) basiert auf demselben Verfahren wie der Solver *adivaOrigin*, allerdings wird die IvariMatrix vorsortiert. Beim Solver *adiperm* (vgl. Code 9.19) werden dagegen vorab alle möglichen Zeilenvertauschungen der Intervallmatrix durchgeführt und die Intervallweiten der sich jeweils ergebenden Determinante ermittelt. Das Adjunkten-Verfahren wird schließlich auf dasjenige Intervall-Gleichungssystem angewandt, dessen zugehörige IvariMatrix die Determinante mit der geringsten Intervallweite aufweist.

Methode adivaOrigin(IvariVector s)

Der Solver *adivaOrigin* wird durch die Methode *adivaOrigin(IvariVector s)* (vgl. Code 9.17, Zeile 1 ff.) ausgeführt. Dabei wird über die adjunkten Matrix die Lösung des Intervall-Gleichungssystems ermittelt. Hierfür wird die Methode *inverse(IvariMatrix matrix)* (vgl. Code 9.17, Zeile 9 ff.) aufgerufen.

Code 9.17: Methoden adivaOrigin(IvariVector s), inverse(IvariMatrix matrix), determinant(IvariMatrix matrix) und submatrix(IvariMatrix matrix, int row, int column)

```
1 private IvariVector adivaOrigin(IvariVector s) {
2   IvariMatrix inv = inverse(this);
3   IvariVector result = new IvariVector(s.getSize());
4   result = inv.multVector(s);
5   return result;
6 }
```

```
7
8 // Methode inverse(IvariMatrix matrix) zur Berechnung der Inversen
9 public IvariMatrix inverse(IvariMatrix matrix) {
10   IvariMatrix inverse = new IvariMatrix(matrix.numberRows, matrix.numberColumns);
11   IvariMatrix temp = new IvariMatrix(matrix.numberRows, matrix.numberColumns);
12   IvariScalar determ = determinant(matrix).inv();
13   for (int i = 0; i < matrix.numberRows; i++) {
14     for (int j = 0; j < matrix.numberColumns; j++) {
15       IvariMatrix submatrix = submatrix(matrix, i, j);
16       temp.matrix[i][j] = determinant(submatrix).mult(Math.pow(-1, (i + j)));
17     }
18   }
19   for (int i = 0; i < matrix.numberRows; i++) {
20     for (int j = 0; j < matrix.numberColumns; j++) {
21       inverse.matrix[i][j] = temp.matrix[j][i].mult(determ);
22     }
23   }
24   return inverse;
25 }
26
27 // Methode determinant(IvariMatrix matrix) zur Berechnung der Determinanten
28 private IvariScalar determinant(IvariMatrix matrix) {
29   if (matrix.getSize() != matrix.getSize())
30     throw new IllegalStateException("Abweichende␣Dimensionen");
31   if (matrix.getSize() == 1)
32     return matrix.matrix[0][0];
33   if (matrix.getSize() == 2)
34     return (matrix.matrix[0][0].mult(matrix.matrix[1][1])).sub(matrix.matrix[0][1].mult(matrix.
         matrix[1][0]));
35   int k = 0;
36   IvariScalar det = new IvariScalar (.0,.0);
37   for (int i = 0; i < matrix.getSize(); i++) {
38     k  = i;
39     IvariScalar aij = matrix.matrix[0][k].mult(Math.pow(-1, k));
40     IvariMatrix submatrix = submatrix(matrix, 0, k);
41     IvariScalar determinant = determinant(submatrix);
42     det = det.add(aij.mult(determinant));
43   }
44   return det;
45 }
46
47 // submatrix(IvariMatrix matrix, int row, int column) zur Bildung der Submatrix
48 private IvariMatrix submatrix(IvariMatrix matrix, int row, int column) {
49   IvariMatrix sub = new IvariMatrix(matrix.numberRows-1, matrix.numberRows-1);
50   for (int i = 0; i < matrix.numberRows; i++) {
51     for (int j = 0; i != row && j < matrix.numberColumns; j++) {
52       if (j != column)
53         sub.matrix[i < row ? i : i - 1][j < column ? j : j - 1]
54           = matrix.matrix[i][j];
55     }
56   }
57   return sub;
58 }
```

Die Ermittlung der adjunkten Matrix erfolgt durch Aufruf der Methode *determinant(IvariMatrix matrix)* (vgl. Code

9.17, Zeile 28 ff.) und der Methode *submatrix(IvariMatrix matrix, int row, int column)* (vgl. Code 9.17, Zeile 48 ff.). In der Methode *determinant(IvariMatrix matrix)* wird die Determinante für eine IvariMatrix $\in \mathbb{IR}^{1x1}$ und $\in \mathbb{IR}^{2x2}$ direkt berechnet (vgl. Code 9.17, Zeile 31 ff.); bei größeren Dimensionen erfolgt die Kalkulation der Determinanten über den Laplaceschen Entwicklungssatz (vgl. Code 9.17, Zeile 37 ff.). Die hierfür notwendigen Unterdeterminanten werden rekursiv durch Aufruf der Methode *submatrix(IvariMatrix matrix, int row, int column)* berechnet (vgl. Code 9.17, Zeile 41), deren Rückgabewert wiederum der Methode *determinant(IvariMatrix matrix)* übergeben wird (vgl. Code 9.17, Zeile 41).

adiheu(IvariVector s)

Die Methode *adiheu(IvariVector s)* (vgl. Code 9.18) ergänzt die Methode *adivaOrigin(IvariVector s)* (vgl. Code 9.17) um eine vorherige Sortierung der IvariMatrix *matrix*.

Code 9.18: Methode adiheu(IvariVector s)

```
1 private IvariVector adiheu(IvariVector s) {
2   LinkedList<Integer> swapList = heuSort2(s);
3   IvariVector result = new IvariVector(getSize());
4   result = adivaOrigin(s);
5   result = heuBack(result, tauschListe);
6   return result;
7 }
```

Der aufgerufene Sortieralgorithmus *heuSort2(IvariVector s)* (vgl. Code 9.14), Zeile 2) kommt auch beim Solver *salt* zur Anwendung. Er formt eine IvariMatrix derart um, dass links von der Hauptdiagonalen vorzugsweise Intervalle mit negativen Vorzeichen oder Nullintervalle angeordnet sind.

Methoden adiperm(IvariVector s) und permutateMatrix(IvariMatrix matrix, IvariVector s)

Die Methode *adiperm(IvariMatrix matrix, IvariVector s)* (vgl. Code 9.19) ergänzt *adivaOrigin(IvariVector s)* (vgl. Code 9.17) um eine vorherige Permutation der IvariMatrix. Dabei werden alle möglichen Zeilenvertauschungen sowie einhergehend notwendige Umstellungen des zugehörigen IvariVectors durchgeführt und die Determinantenintervalle der jeweiligen Permutationen berechnet. Das Adjunkten-Verfahren wird anschließend mit derjenigen Permuationsmatrix durchgeführt, deren Determinantenintervall die geringste Weite aufweist.

Code 9.19: Methode adiperm(IvariVector s) und permutateMatrix(IvariMatrix matrix, IvariVector s)

```
1 // adiperm(IvariVector s) zur Berechnung des Intervall-Gleichungssystems
2 private IvariVector adiperm(IvariVector s) {
3   HashMap<IvariMatrix, IvariVector> sorted = permutateMatrix(this, s);
4   IvariMatrix inv = inverse(this);
5   IvariVector result = sorted.get(this);
6   result = inv.multVector(s);
7   return result;
8 }
9
10 //  permutateMatrix(IvariMatrix matrix, IvariVector s) zur Permutation der Zeilen der IvariMatrix
11 private HashMap<IvariMatrix, IvariVector> permutateMatrix(IvariMatrix matrix, IvariVector s) {
12   HashMap<IvariMatrix, IvariVector> sorted = new HashMap<IvariMatrix, IvariVector>();
13   IvariScalar detStart = determinant(matrix);
14   double widthOfDetStart = detStart.getUpperBound()-detStart.getLowerBound();
15   IvariMatrix bestMatrix = new IvariMatrix(numberRows);
```

```
16   IvariVector r = new IvariVector(numberRows);
17   int[] p = new int[numberRows];
18   int i = 1, j = 0;
19   bestMatrix = matrix;
20   r = s;
21   sorted.put(bestMatrix, r);
22   while (i < numberRows) {
23     if (p[i] < i) {
24       j = (i % 2) * p[i];
25       changeRows(i, j);
26       r.changeValues(i, j);
27       IvariScalar detCurrent = determinant(matrix);
28       double widthOfDetCurrent = detCurrent.getUpperBound()
29         - detCurrent.getLowerBound();
30       if (widthOfDetCurrent < widthOfDetStart) {
31         bestMatrix = matrix;
32         sorted.replace(bestMatrix, r);
33       }
34       p[i]++;
35       i = 1;
36     } else {
37       p[i] = 0;
38       i++;
39     }
40   }
41   return sorted;
42 }
```

Der Algorithmus zur Permutation ist in der Methode *permutateMatrix(IvariMatrix matrix, IvariVector s)* implementiert (vgl. Code 9.19, Zeile 11). Er beruht auf den Grundsätzen von [42] und wurde auf die Intervallebene übertragen. Zusätzlich wurde er um die Kalkulation der jeweiligen Determinanten (vgl. Code 9.19, Zeile 27), die Berechnung der Intervallweite der Determinante (vgl. Code 9.19, Zeile 29) und um das Speichern der gesuchten IvariMatrix (vgl. Code 9.19, Zeile 30 ff.) ergänzt. Die Methode gibt schließlich die HashMap *sorted* mit derjenigen IvariMatrix und dem zugehörigen IvariVector zurück, dessen Determinante die geringste Intervallweite aufweist.

9.5 Methoden zum Splitten der Matrix

Mit dem Solver *adisplit* wurde ein rekursiver Algorithmus implementiert, mit dem die zugrunde gelegte IvariMatrix in kleinere Matrizen zerlegt wird. Er wird durch die Methode *adisplit(IvariVector s)* ausgeführt. Bei diesem Verfahren wird die IvariMatrix des Intervall-Gleichungssystems nach linearen und baumartigen Submatrizen durchsucht. Die detektierten Submatrizen werden von der IvariMatrix gesplittet, gesondert berechnet und komprimiert als einzelnes Prozessmodul in die IvariMatrix zurückgeführt. Das Ergebnis ist eine IvariMatrix mit reduzierter Dimension. Für das Verfahren wurden diverse Klassen und Methoden implementiert (vgl. Abbildung 9.4).

9.5.1 Methode adisplit(IvariVector s)

Der Solver *adisplit* (vgl. Code 9.20, Zeile 1 ff.) ergänzt die Gleichungslöser um eine vorherige Splittung der IvariMatrix in kleinere Submatrizen. Diese werden von der ursprünglichen Produktmatrix entfernt, gesondert berechnet und schließlich wieder in die Produktmatrix komprimiert als nur eine Gleichung mit einer Unbekannten integriert. Die Methode *adisplit* ruft zur Lösung die Methode *adiva_ rekursiv(this, s)* (vgl. Code 9.20, Zeile 7 ff.) auf.

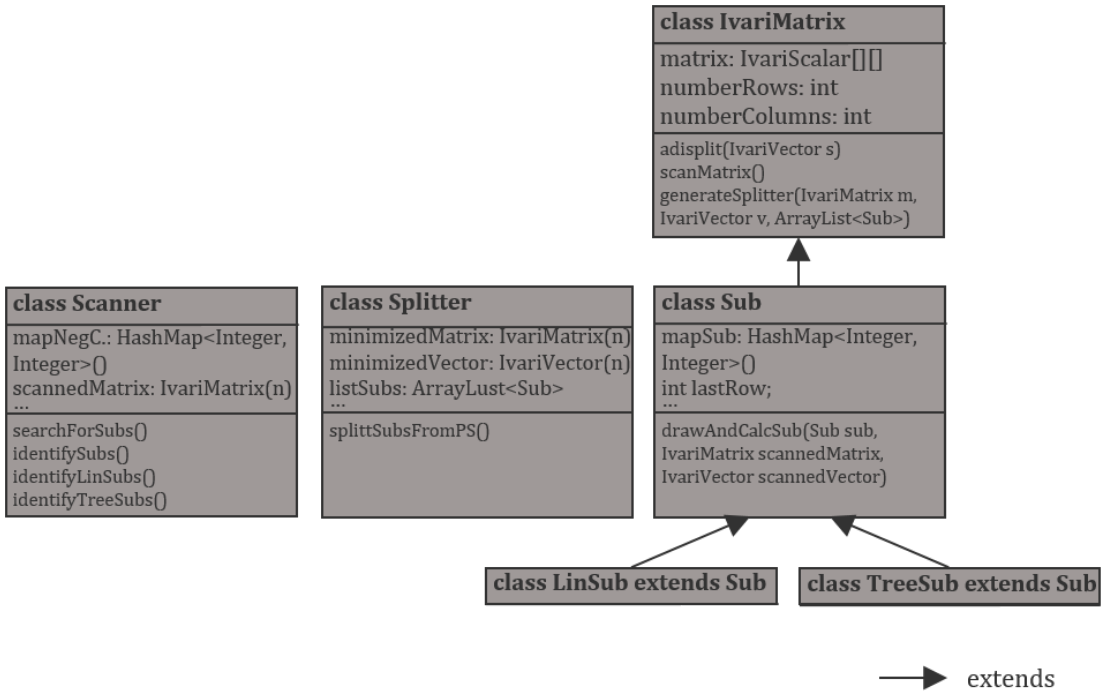


Abbildung 9.4: Die für *adisplit* relevanten Klassen IvariMatrix, Scanner, Splitter, Sub, LinSub und TreeSub.

Das Vorgehen erfolgt rekursiv, d. h. die verkleinerte IvariMatrix wird durch erneutes Aufrufen der Methode *adiva_rekursiv(this, s)* abermals nach auftretenden Subsystemen durchsucht und ggfs. weiter verkleinert. Zunächst werden Subsysteme der zu berechnenden IvariMatrix gescannt (vgl. Code 9.20, Zeile 8) durch die Methode *scanMatrix()* (vgl. Code 9.21) und anschließend die detektierten Subsysteme der Matrix komprimiert (vgl. Code 9.20, Zeile 9) durch die Methode *generateSplitter(IvariMatrix matrix, IvariVector vector, ArrayList<Sub>)* (vgl. Code 9.22). Solange durch erneutes Scannen weitere Subsysteme der minimierten IvariMatrix gefunden werden, wird dieser Vorgang durch rekursiven Aufruf der Methode *adisplit(IvariVector s)* wiederholt (vgl. Code 9.20, Zeile 14 f.). Da die ursprüngliche Produktmatrix dadurch verändert wird, wird sie den Methoden als Parameter übergeben.

Code 9.20: Methode adisplit(IvariVector s) und adiva_rekursiv(IvariMatrix matrix, IvariVector vector)

```
1 private IvariVector adisplit(IvariVector s) {
2   IvariVector finalResult = new IvariVector(numberRows);
3   finalResult = adiva_rekursiv(this, s);
4   return finalResult;
5 }
6
7 private IvariVector adiva_rekursiv(IvariMatrix matrix, IvariVector vector) {
8   ArrayList<Sub> listSubs =  scanMatrix(matrix, vector);
9   Splitter splittedPS = generateSplitter(matrix, vector, listSubs);
10  IvariMatrix minimizedMatrix = splittedPS.getMatrix();
11  IvariVector minimizedVector = splittedPS.getVector();
12  IvariVector finalResult = new IvariVector(vector.getSize());
13  if(!(minimizedMatrix.
14      scanMatrix(minimizedMatrix,  minimizedVector).isEmpty())){
15    adiva_rekursiv(minimizedMatrix, minimizedVector);
16  }
```

```
17   IvariVector resultMinimized = minimizedMatrix.goody(minimizedVector);
18   for (int j = listSubs.size(); j < minimizedVector.getSize(); j++) {
19      finalResult.setValue(splittedPS.getMinimizedPSColumns().get(j), resultMinimized.getValue(j))
            ;
20   }
21   for (int i = 0; i < listSubs.size(); i++) {
22     ArrayList<Integer> listColumns = listSubs.get(i).getListColumns();
23     IvariVector resultSub = listSubs.get(i).getResultSub();
24     IvariVector resultForPS = resultSub.mult(resultMinimized.getValue(i));
25     for (int k = 0; k < resultForPS.getSize(); k++) {
26       finalResult.setValue(listColumns.get(k), finalResult.getValue(listColumns.get(k)).add(
             resultForPS.getValue(k)));
27     }
28   }
29   return finalResult;
30 }
```

Anschließend wird das Intervall-Gleichungssystem der minimierten Matrix (vgl. Code 9.20, Zeile 17) durch Aufruf der Methode *goody(IvariVector s)* berechnet (Anmerkung: die Methode *goody* stellt einen Teil des allgemeinen Verfahrens dar, welches in dieser Arbeit abgeleitet worden ist. Es wird in Kapitel 10.4.2 beschrieben). Dem IvariVector *finalResult* werden zunächst die Ergebnisse der einzelnen Prozessmodule zugewiesen, welche keinem Subsystem zugeordnet worden sind (vgl. Code 9.20, Zeile 18 f.). Anschließend werden die Ergebnisse der Subsysteme hinzugefügt (vgl. Code 9.20, Zeile 23). Die Resultate werden dann mit den entsprechenden Skalierungsfaktoren des minimierten Systems multipliziert und im letzten Schritt der korrekten Stelle im resultierenden IvariVector *finalResult* übergeben (vgl. Code 9.20, Zeile 25 ff.).

Methode scanMatrix(IvariMatrix matrix, IvariVector vector())

Die Methode *scanMatrix(IvariMatrix matrix, IvariVector vector()* (vgl. Code 9.21) ist in der Klasse IvariMatrix implementiert und dient zur Instanziierung eines Objekts vom Typ Scanner (vgl. Code 9.21, Zeile 2). Das erzeugte Objekt wird zur Identifizierung der Subsysteme der ihm übergebenen IvariMatrix *matrix* sowie des zugehörigen IvariVector *vector* verwendet (vgl. Code 9.21, Zeile 3).

Code 9.21: Methode scanMatrix(IvariMatrix matrix, IvariVector vector)

```
1 private ArrayList<Sub> scanMatrix(IvariMatrix matrix, IvariVector vector) {
2   Scanner scannedMatrix = new Scanner(matrix, vector);
3   ArrayList<Sub> listSubs = scannedMatrix.identifySubs();
4   return listSubs;
5 }
```

Methode generateSplitter(IvariMatrix matrix, IvariVector vector(), ArrayList <Sub>)

Die Methode *generateSplitter(IvariMatrix matrix, IvariVector vector(), ArrayList<Sub>)* (vgl. Code 9.22) ist in der Klasse IvariMatrix implementiert und dient zur Instanziierung eines Splitter-Objekts (vgl. Code 9.22, Zeile 2).

Code 9.22: Methode generateSplitter(IvariMatrix matrix, IvariVector vector(), ArrayList <Sub>)

```
1 private Splitter generateSplitter(IvariMatrix matrix, IvariVector vector, ArrayList<Sub> listSubs
      ) {
2   Splitter splitter = new Splitter(matrix, vector, listSubs);
3   splitter.splittSubsFromPS(matrix, vector);
```

```
4   return splitter;
5 }
```

Das erzeugte Objekt wird zur Reduktion der ihm übergebenen IvariMatrix *matrix* sowie des zugehörigen IvariVectors *vector* (vgl. Code 9.22, Zeile 3) durch Komprimierung der detektierten Subsysteme verwendet.

9.5.2 Scanner.java

Die Klasse Scanner (vgl. Code 9.23) stellt Methoden bereit, mit der die ihm übergebene IvariMatrix *scannedMatrix* und der IvariVector *scannedVector* durchsucht werden kann. Die Klasse enthält mehrere Instanzvariablen und einen Konstruktor.

Code 9.23: Instanzvariablen und Konstruktor von Scanner.java

```
1 public class Scanner {
2
3   // Instanzvariablen
4   HashMap<Integer, Integer> mapNegColumns  = new HashMap<Integer, Integer>();
5   HashMap<Integer, Integer> mapPosColumns  = new HashMap<Integer, Integer>();
6   HashMap<Integer, Integer> mapNegRows  = new HashMap<Integer, Integer>();
7   HashMap<Integer, Integer> mapPosRows  = new HashMap<Integer, Integer>();
8   HashMap<Integer, ArrayList<Integer>> mapRoot  = new HashMap<Integer, ArrayList<Integer>>();
9   int numberRows;
10  IvariMatrix scannedMatrix = new IvariMatrix(numberRows);
11  IvariVector scannedVector = new IvariVector(numberRows);
12
13  // Konstruktor
14  public Scanner(IvariMatrix matrix, IvariVector vector) {
15    numberRows = matrix.getSize();
16    scannedMatrix = matrix;
17    scannedVector = vector;
18    searchForSubs();
19  }
20 }
```

Die Instanzvariablen *mapNegColumns, mapPosColumns, mapNegRows* und *mapPosRows* sind Collections vom Typ HashMap<Integer, Integer> deklariert (vgl. Code 9.23, Zeile 4 ff.) und dienen zur Speicherung von Zeilen- und Spaltennummern bestimmter Matrixelemente. In den HashMaps *mapNegColumns* und *mapPosColumns* werden die Zeilennummern als Keys gespeichert und die zugehörigen Spaltennummern als Values; in den HashMaps *mapPosRows* und *mapNegRows* werden die Spaltennummern als Keys und die zugehörigen Zeilennummern als Values gespeichert. Die Instanzvariable *mapRoot* ist vom Typ HashMap<Integer, ArrayList<Integer>> als Collection deklariert (vgl. Code 9.23, Zeile 8). In dieser HashMap werden mehrere Matrixelemente einer Spalte gespeichert; die Spaltennummern werden als Keys eingetragen und die zugehörigen Values als Values in Form einer ArrayList übergeben. Der Konstruktor (vgl. Code 9.23, Zeile 14 ff.) enthält Parameter vom Typ IvariMatrix und vom Typ IvariVector, durch welche die zu durchsuchenden Objekte übergeben werden. Beim Aufrufen eines Scanner-Objekts wird direkt die Methode *searchForSubs()* (vgl. Code 9.23, Zeile 18) aufgerufen.

Methode searchForSubs()

Mit der Methode *searchForSubs()* (vgl. Code 9.24) werden die Parameter direkt beim Aufruf des Konstruktors hinsichtlich der Merkmale eines linearen oder baumartigen Subsystems (vgl. Kapitel 7.5.1) gescannt. In den lokalen

Variablen *posColumns, negColumns, posRows* und *negRows* werden Spalten- und Zeilennummern der Produktmatrix in Form von Listen vom Typ ArrayList<Integer> hinterlegt (vgl. Code 9.24, Zeile 2 ff.). Dies erfolgt für diejenigen Spalten- und Zeilennummern, welche zeilenweise (*posColumns* und *negColumns*) bzw. spaltenweise (*posRows* und *negRows*) positive oder negative Intervalle aufweisen (vgl. Code 9.24, Zeile 8 ff.).

Code 9.24: Methode searchForSubs()

```
1  public void searchForSubs() {
2    ArrayList<Integer> posColumns = new ArrayList<Integer>();
3    ArrayList<Integer> negColumns = new ArrayList<Integer>();
4    ArrayList<Integer> posRows = new ArrayList<Integer>();
5    ArrayList<Integer> negRows = new ArrayList<Integer>();
6    for (int i = 0; i < numberRows; i++) {
7      for (int j = 0; j < numberRows; j++) {
8        if (scannedMatrix.getValue(i, j).getLowerBound() < 0.0){
9          negColumns.add(j);
10       }
11       if (scannedMatrix.getValue(i, j).getLowerBound() > 0.0) {
12         posColumns.add(j);
13       }
14       if (scannedMatrix.getValue(j, i).getLowerBound() < 0.0
15           && scannedVector.getValue(j).getLowerBound()== 0) {
16         negRows.add(j);
17       }
18       if (scannedMatrix.getValue(j, i).getLowerBound() > 0.0) {
19         posRows.add(j);
20       }
21     }
22     if (posColumns.size() == 1
23         && scannedVector.getValue(i).getLowerBound()== 0) {
24       mapPosColumns.put(i, posColumns.get(0));
25       if (negColumns.size() == 1) {
26         mapNegColumns.put(i, negColumns.get(0));
27       }
28     }
29     if (posRows.size() == 1
30         && scannedVector.getValue(posRows.get(0)).getLowerBound()== 0) {
31       mapPosRows.put(i, posRows.get(0));
32       if (negRows.size() == 1 ) {
33         mapNegRows.put(i, negRows.get(0));
34       } else if(negRows.size()> 1) {
35         ArrayList<Integer> rowsRoot = new ArrayList<Integer>();
36         rowsRoot.addAll(negRows);
37         mapRoot.put(i, rowsRoot);
38       }
39     }
40     negColumns.clear(); posColumns.clear();
41     negRows.clear(); posRows.clear();
42   }
43 }
```

Anhand dieser Nummern wird überprüft, ob die Produktmatrix die Kriterien eines linearen oder baumartigen Produktsystems erfüllt. Ist dies der Fall, werden die Nummern den jeweiligen HashMaps übergeben. Positive Elemente der IvariMatrix *matrix* werden in der HashMap<Integer, Integer> mapPosColumns gespeichert, wenn nur

ein positives Intervall aller Spaltenelemente innerhalb einer Zeile (inklusive der zugehörigen Zeile des IvariVectors *vector*) identifiziert wurde (vgl. Code 9.24, Zeile 24). Wird in derselben Zeile ebenfalls genau ein negatives Intervall identifiziert, wird dieses Element analog hierzu in der HashMap<Integer, Integer> *mapNegColumns* gespeichert (vgl. Code 9.24, Zeile 26). Nach dem gleichen Prinzip werden alle Zeilen einer Spalte durchsucht und dementsprechend in der HashMap<Integer, Integer> *mapPosRows* bzw. HashMap<Integer, Integer> *mapNegRows* gespeichert (vgl. Code 9.24, Zeile 31 und (vgl. Code 9.24, Zeile 33). Hier werden zusätzlich in der HashMap *mapRoot* Prozessmodule gespeichert, die als Wurzelknoten eines baumartigen Subsystems fungieren könnten. Die Values der *mapRoot* mit den negativen Intervallelementen einer Spalte werden als ArrayList<Integer> gespeichert, welche über die zugehörige Spaltennummer, den Key der HashMap *mapRoot*, aufgerufen werden können.

Methode identifySubs()

Durch die in der Klasse Scanner implementierte Methode *identifySubs()* (vgl. Code 9.25) werden die Objekte zuerst nach linearen Subsystemen und anschließend nach baumartigen Subsystemen durchsucht. Die Zeilen- und Spaltennummern baumartiger Subsysteme dürfen nicht Teil linearer Subsysteme sein. Die identifizierten Subsysteme in der lokalen Variable ArrayList<Sub> *listSubs* (vgl. Code 9.25, Zeile 4) gespeichert.

Code 9.25: Methode identifySubs()

```
1  public ArrayList<Sub> identifySubs() {
2
3    HashSet<Integer> rowsToRemove = new HashSet<Integer>();
4    ArrayList<Sub> listSubs = new ArrayList<Sub>();
5    ArrayList<Sub> listLinSubs = identifyLinSubs();
6    listSubs.addAll(listLinSubs);
7    ArrayList<Integer> listAllRows = this.collectRowsOfAllSubs(listLinSubs);
8    for (int i = 0; i < listAllRows.size(); i++) {
9      for (Integer key : mapRoot.keySet()) {
10       if (mapRoot.containsKey(listAllRows.get(i)) ||  mapRoot.get(key).contains(listAllRows.get
             (i))) {
11         rowsToRemove.add(key);
12       }
13     }
14   }
15     Iterator<Integer> iterator = rowsToRemove.iterator();
16     while(iterator.hasNext()){
17         mapRoot.remove(iterator.next());
18     }
19   ArrayList<Sub> listTreeSubs = identifyTreeSubs();
20   listSubs.addAll(listTreeSubs);
21   return listSubs;
22 }
```

Die Identifikation nach linear- bzw. baumartigen Subsystemen erfolgt durch die Methoden *identifyLinSubs()* (vgl. Code 9.26) und *identifyTreeSubs()* (vgl. Code 9.27). Um das Kriterium der Unabhängigkeit zu erfüllen, d. h. zu vermeiden, dass ein Teil eines baumartigen Subsystems bereits einem linearen Subsystem zugeordnet wurde, werden in der Collection *rowsToRemove* vom Typ HashSet<Integer> (vgl. Code 9.25, Zeile 3) alle Zeilen gespeichert, die bereits einem linearen Subsystem zugeordnet werden, jedoch auch in der HashMap *mapRoot* als Teil potentieller, baumartiger Subsystem-Kanditaten aufgelistet sind (vgl. Code 9.25, Zeile 8 ff.). Die gespeicherten Zeilennummern werden daher von der HashMap *mapRoot* entfernt (vgl. Code 9.25, Zeile 16 ff.). Im Anschluss werden die verbliebe-

nen möglichen, baumartigen Subsysteme durch Aufruf der Methode *identifyTreeSubs()* (vgl. Code 9.27) überprüft und bei Erfüllung aller Anforderungen der ArrayList<Sub> *listSubs* hinzugefügt.

identifyLinSubs()

Mit der Methode *identifyLinSubs()* (vgl. Code 9.26) werden die HashMaps verwendet, um zu überprüfen, ob die Anforderungen an ein lineares Subsystem erfüllt werden. Zunächst wird ein rein erzeugendes Prozessmodul ohne Vorgänger-Prozessmodule gesucht (vgl. Code 9.26, Zeile 7), in dem kein Koppelprodukt erzeugt wird (vgl. Code 9.26, Zeile 5) und dessen erzeugtes Produkt nur in einem weiteren Prozessmodul verwendet wird (vgl. Code 9.26), Zeile 6). Die Spaltennummer des detektierten Prozessmoduls wird in der erzeugten *linSub* gespeichert (vgl. Code 9.26, Zeile 10). Von diesem „Startmodul“ ausgehend werden nachfolgende Prozessmodule gesucht. Dies wird fortgesetzt, solange kein Koppelprodukt auftritt (vgl. Code 9.26, Zeile 12), das Prozessmodul nur einen Nachfolger aufweist (vgl. Code 9.26, Zeile 13) und das erzeugte Produkt nur einmalig hergestellt wird (vgl. Code 9.26, Zeile 14). Wird dies erfüllt, wird die Zeilennummer des erzeugten Produkts und die Spaltennummer des zugehörigen Prozessmoduls entsprechend in der HashMap *mapLinSub* gespeichert (vgl. Code 9.26, Zeile 15).

Code 9.26: Methode identifyLinSubs()

```
1  public ArrayList<Sub> identifyLinSubs() {
2    ArrayList<Sub> listLinSubs = new ArrayList<Sub>();
3    for (int i = 0; i < numberRows; i++) {
4      int j = i;
5      if (mapPosColumns.containsKey(j)
6          && mapNegColumns.containsKey(j)
7          && !(mapNegColumns.containsValue(mapPosColumns.get(j)))){
8        LinSub linSub = new LinSub();
9        HashMap<Integer, Integer> mapLinSub = new HashMap<Integer, Integer>();
10         linSub.setFirstColumn(mapPosColumns.get(j));
11         while (j < numberRows
12             && mapPosColumns.containsKey(j)
13             && mapNegColumns.containsKey(j)
14             && (mapPosRows.containsKey(mapPosColumns.get(j)))) {
15         mapLinSub.put(j, mapPosColumns.get(j));
16         int column = mapNegColumns.get(j);
17         if (mapNegRows.containsKey(column)
18             && mapPosRows.containsKey(column)) {
19           int row = mapPosRows.get(column);
20           j = row;
21         } else {
22           linSub.setLastRow(j);
23           linSub.setLastColumn(mapPosColumns.get(j));
24           j = numberRows;
25         }
26       }
27       if (mapLinSub.size() > 1) {
28         linSub.setMapSub(mapLinSub);
29         linSub.drawAndCalcSub(linSub, scannedMatrix, scannedVector);
30         listLinSubs.add(linSub);
31       }
32     }
33   }
34   return listLinSubs;
```

```
35  }
```

Sobald die Anforderungen nicht mehr erfüllt sind, wird die Zeilennummer des zuletzt gültigen Prozessmoduls in der linSub gespeichert (vgl. Code 9.26, Zeile 22). Enthält die HashMap *mapLinSub* mehr als ein Wertepaar, wird die entsprechende *linSub* modelliert, berechnet und der Liste *listLinSubs* hinzugefügt (vgl. Code 9.26, Zeile 27 ff.).

Methode identifyTreeSubs()

Mit der Methode *identifyTreeSubs()* (vgl. Code 9.27) wird überprüft, ob die Anforderungen an ein baumartiges Subsystem erfüllt werden. Die Durchsuchung erfolgt ausgehend von den einzelnen, potentiell baumartigen Systemen, die in der HashMap *mapRoot* gespeichert sind (vgl. Code 9.27, Zeile 4 ff.). Zunächst wird die Spaltennummer des Prozessmoduls, welches als potentieller „Wurzelknoten“ fungiert, als *firstColumn* des erzeugten TreeSubs *treeSub* gespeichert (vgl. Code 9.27, Zeile 7).

Code 9.27: Methode identifyTreeSubs()

```
1   public ArrayList<Sub> identifyTreeSubs() {
2     ArrayList<Sub> listTreeSubs = new ArrayList<Sub>();
3     HashMap<Integer, Integer> mapPrevious  = new HashMap<Integer, Integer>();
4     for (Integer key : mapRoot.keySet()){
5       TreeSub treeSub = new TreeSub();
6       HashMap<Integer, Integer> mapTreeSub = new HashMap<Integer, Integer>();
7       treeSub.setFirstColumn(key);
8       for (int i = 0; i < mapRoot.get(key).size(); i++) {
9         int row = mapRoot.get(key).get(i);
10        if (mapPosColumns.containsKey(row)) {
11          mapTreeSub.put(mapPosRows.get(key), key);
12          int potentialLeaf = mapPosColumns.get(row);
13          if (!(mapRoot.containsKey(potentialLeaf))
14              && mapNegColumns.containsKey(row)) {
15            mapTreeSub.put(row, mapPosColumns.get(row));
16            if (mapNegRows.containsKey(mapPosColumns.get(row))) {
17              if (mapPrevious.isEmpty()
18                  || mapPrevious.containsKey(mapNegRows.get(row))) {
19                mapPrevious.put(mapNegRows.
20                get(row), mapPosColumns.get(mapNegRows.get(row)));
21                treeSub.setFirstColumn(mapPosColumns.get(row));
22              } else {
23                i = mapRoot.get(key).size();
24                mapTreeSub.clear();
25              }
26            }
27          } else {
28            i = mapRoot.get(key).size();
29            mapTreeSub.clear();
30          }
31        }
32      }
33      if (mapTreeSub.size() > 1) {
34        treeSub.setLastColumn(key);
35        treeSub.setLastRow(mapPosRows.get(key));
36        treeSub.setMapSub(mapTreeSub);
37        treeSub.drawAndCalcSub(treeSub, scannedMatrix, scannedVector);
38        listTreeSubs.add(treeSub);
```

```
39        }
40      }
41      return listTreeSubs;
42    }
```

Anschließend werden die „Blätter“ des potentiellen Subsystems untersucht. Wenn in den Prozessmodulen, die als „Blätter“ fungieren, keine Koppelprodukte erzeugt werden (vgl. Code 9.27, Zeile 10), deren erzeugte Produkte nur in einem weiteren Prozessmodul verwendet werden (vgl. Code 9.27, Zeile 14) und die potentiellen „Blätter“ keine internen Wurzelknoten sind (vgl. Code 9.27, Zeile 13), werden sie zur HashMap *mapTreeSub* des TreeSubs *treeSub* hinzugefügt (vgl. Code 9.27, Zeile 15). Anderenfalls werden die Einträge der HashMap *mapTreeSub* geleert und das nächste potentielle Subsystem in der HashMap *mapRoot* durchsucht (vgl. Code 9.27, Zeile 27 ff.).

Zusätzlich wird mit der lokalen Variable *mapPrevious* vom Typ HashMap<Integer, Integer> überprüft, dass nur eines der als „Blätter“ fungierenden Prozessmodule einen Vorgänger aufweist (vgl. Code 9.27, Zeile 17 ff.). Wird ein Prozessmodul mit einem Vorgänger gefunden, wird die *firstColumn* mit der Spaltennummer dieses Prozessmoduls überschrieben. Wird im weiteren Verlauf ein weiteres Prozessmodul mit vorangehendem Prozessmodul gefunden, werden die Einträge der HashMap *mapTreeSub* geleert und und das nächste, potentielle und in der HashMap *mapRoot* gespeicherte Subsystem überprüft (vgl. Code 9.27, Zeile 22 ff.). Wenn alle eingetragenen Werte der HashMap *mapRoot* eines Keys untersucht worden sind und die HashMap *mapTreeSub* mehr als ein Wertepaar enthält, wird das entsprechende TreeSub *treeSub* modelliert, berechnet und der ArrayList *listTreeSubs* hinzugefügt (vgl. Code 9.27, Zeile 33 ff.).

9.5.3 Splitter.java

Die Klasse Splitter (vgl. Code 9.28) stellt Methoden bereit, mit der Intervallmatrizen und deren zugehörige Intervallvektoren durch Komprimierung der detektierten Subsysteme verkleinert werden können. Die Klasse enthält mehrere Instanzvariablen und einen Konstruktor. Die Instanzvariablen *minimizedMatrix* vom Typ IvariMatrix und *minimizedVector* vom Typ IvariVector dienen zur Speicherung der verkleinerten Produktmatrix und dem zugehörigen, komprimierten Bedarfsvektor (vgl. Code 9.28, Zeile 4 f.).

Code 9.28: Instanzvariablen und Konstruktoren von Splitter.java

```
1  public class Splitter {
2    // Instanzvariablen
3    int numberColumns;
4    IvariMatrix minimizedMatrix = new IvariMatrix(numberColumns);
5    IvariVector minimizedVector = new IvariVector(numberColumns);
6    ArrayList<Sub> listSubs = new ArrayList<Sub>();
7    ArrayList<Integer> minimizedPSRows = new ArrayList<Integer>();
8    ArrayList<Integer> minimizedPSColumns = new ArrayList<Integer>();
9    ArrayList<Integer> subsLastColumns = new ArrayList<Integer>();
10
11   // Konstruktor
12   public Splitter (IvariMatrix matrix, IvariVector vector, ArrayList<Sub> listSubs) {
13     this.numberColumns = matrix.getSize();
14     this.minimizedMatrix = matrix;
15     this.minimizedVector = vector;
16     this.listSubs = listSubs;
17   }
18   //...
19 }
```

In den Listen *minimizedPSRows* und *minimizedPSColumns* werden die notwendigen Zeilen- bzw. Spaltennummern gespeichert (vgl. Code 9.28, Zeile 7 f.), sie sind vom Typ ArrayList<Integer>. Mit der Instanzvariablen *subsLastColumns* desselben Typs werden die letzten Spalten der detektierten Subsysteme bereitgestellt (vgl. Code 9.28, Zeile 9). Der Konstruktor (vgl. Code 9.28, Zeile 12 ff.) enthält die Parameter IvariMatrix *matrix*, IvariVector *vector* und ArrayList<Integer> *listSubs*; mit *listSubs* werden die durchsuchten Objekte und detektierten Subsysteme übergeben.

Methode splittSubsFromPS()

Durch die Methode *splittSubsFromPS(IvariMatrix matrix, IvariVector s)* (vgl. Code 9.29) werden die minimierte IvariMatrix *minimizedMatrix* und der zugehörige IvariVector *minimizedVector* mit den entsprechenden Prozessmodulen belegt. Dafür werden zunächst die lokalen Variablen *collectRows* und *collectColumns* instanziiert; sie sind vom Typ vom Typ ArrayList<Integer> (vgl. Code 9.29, Zeile 2 f.) und schließlich initialisiert durch Zuweisung aller Zeilen- bzw. Spaltennummern, welche einem Subsystem zugeordnet werden (vgl. Code 9.29, Zeile 10 f.). Die ArrayList<Sub> *listSubs* des Splitters enthält die gespeicherten Subsysteme. Diese Subsysteme werden durchsucht (vgl. Code 9.29, Zeile 4 ff.). Dabei werden den Instanzvariablen *minimizedPSRows, minimizedPSColumns* und *subsLastColumns* auch die relevanten letzten Zeilennummern (vgl. Code 9.29, Zeile 5) und ersten bzw. letzten Spaltennummern (vgl. Code 9.29, Zeile 6 bzw. Code 9.29, Zeile 7) der jeweiligen Subs zugewiesen. Anschließend werden alle Zeilen bzw. Spalten, die nicht Teil der Subsysteme sind, den ArrayLists *minimizedPSRows* und *minimizedPSColumns* hinzugefügt (vgl. Code 9.29, Zeile 15 ff.).

Code 9.29: Methode splittSubsFromPS(IvariMatrix matrix, IvariVector s)

```
1  public void splittSubsFromPS(IvariMatrix matrix, IvariVector s) {
2    ArrayList<Integer> collectRows = new ArrayList<Integer>();
3    ArrayList<Integer> collectColumns = new ArrayList<Integer>();
4    for (Sub sub : listSubs) {
5      minimizedPSRows.add(sub.getLastRow());
6      minimizedPSColumns.add(sub.getFirstColumn());
7      subsLastColumns.add(sub.getLastColumn());
8      for (int i = 0; i < matrix.getSize(); i++) {
9        if (sub.getMapSub().containsKey(i)) {
10         collectRows.add(i);
11         collectColumns.add(sub.getMapSub().get(i));
12       }
13     }
14   }
15   for (int i = 0; i < matrix.getSize(); i++) {
16     if (!(collectRows.contains(i))) {
17       minimizedPSRows.add(i);
18     }
19     if (!(collectColumns.contains(i))) {
20       minimizedPSColumns.add(i);
21       subsLastColumns.add(i);
22     }
23   }
24   int minimizedSize = minimizedPSRows.size();
25   minimizedMatrix = new IvariMatrix(minimizedSize);
26   minimizedVector = new IvariVector(minimizedSize);
27   for (int i = 0; i < minimizedSize; i++) {
28     minimizedVector.setValue(i, s.getValue(minimizedPSRows.get(i)));
```

```
29        for (int j = 0; j < minimizedSize; j++) {
30          if (i < listSubs.size() && j < listSubs.size()) {
31            minimizedMatrix.setValue(i, j, matrix.getValue(minimizedPSRows.get(i), subsLastColumns.
                  get(j)));
32          } else if (i >= listSubs.size() && j < listSubs.size()) {
33            int positionFirstColum = listSubs.get(j).getListColumns().indexOf(listSubs.get(j).
                  getFirstColumn());
34            minimizedMatrix.setValue(i, j, matrix.getValue(minimizedPSRows.get(i),
                  minimizedPSColumns.get(j)).mult(listSubs.get(j).getResultSub().
35            getValue(positionFirstColum)));
36          } else {
37             minimizedMatrix.setValue(i,j, matrix.getValue(minimizedPSRows.get(i),
                   minimizedPSColumns.get(j)));
38          }
39        }
40      }
41    }
```

Die Belegung der minimierten IvariMatrix *minimizedMatrix* sowie des zugehörigen minimierten IvariVectors *minimizedVector* wird wie folgt vorgenommen:

- Zunächst werden die erzeugten Produkte der einzelnen Subelemente und deren Beziehungen untereinander eingetragen (vgl. Code 9.29, Zeile 30 f.), welche in den *lastRows* und *lastColumns* der jeweiligen Subs zu finden ist.

- Anschließend (vgl. Code 9.29, Zeile 32 f.) werden die Beziehungen zu anderen Prozessmodulen, die keinem Subsystem zugeordnet wurden, in weiteren Zeilen der *minimizedMatrix* eingetragen. Dies erfolgt in den Spalten, die zu den jeweiligen Subs gehören. Hierfür werden die Elemente der gespeicherten *firstColumns* und *lastRows* der Subs verwendet. Sie werden mit dem bereits berechneten Skalierungsfaktor des Subsystems multipliziert.

- Zuletzt werden die Elemente der übrigen Prozessmodule in weiteren Spalten sowie Zeilen der *minimizedMatrix* und auch die Beziehungen der Subs zu diesen Prozessmodulen eingetragen (vgl. Code 9.29, Zeile 36 f.). Dies erfolgt über den Aufruf der in den *minimizedPSrows* gespeicherten *lastRows* der Subs bzw. den Zeilennummern der übrigen Prozessmodule sowie durch Aufruf der *firstColumns* der Subs bzw. den Spaltennummern von denjenigen Prozessmodulen, die keinem Sub zugeordnet sind.

9.5.4 Sub.java

Die Klasse Sub erbt von der Klasse IvariMatrix. Sie repräsentiert Subsysteme und enthält mehrere Instanzvariablen sowie zwei Konstruktoren. Die Instanzvariable HashMap<Integer, Integer> *mapSub* (vgl. Code 9.30, Zeile 2) dient zur Speicherung der relevanten Zeilen- und zugehörigen Spaltennummern des Subsystems vom Typ Sub. Die Instanzvariable *lastRow* vom primitiven Datentyp int dient zur Speicherung von Zeilennummern der durchsuchten IvariMatrix. Es wird diejenige Zeilennummer des Subsystems gespeichert, welche dessen erzeugtes Produkt beschreibt (vgl. Code 9.30, Zeile 3). Die Instanzvariable *lastColumn* (vgl. Code 9.30, Zeile 5) vom primitiven Datentyp int dient analog zur Speicherung einer Spaltennummer. Der Variable *lastColumn* wird die Spaltennummer von dem Prozessmodul zugewiesen, welches das Produkt des untersuchten Subsystems erzeugt. Die Instanzvariable *firstColumn* (vgl. Code 9.30, Zeile 4) vom primitiven Datentyp int wird ebenfalls zur Speicherung von Spaltennnummern benötigt. Dabei wird die Spaltenummer desjenigen Prozessmoduls hinterlegt, welches über

ein Vorgänger-Prozessmodul verfügt. Es ist nur ein Vorgänger-Prozessmodul je Subsystem möglich. Es wird somit - im Falle eines linearen Subsystems - diejenige Spaltennummer gespeichert, welche in der Prozessfolge das erste Prozessmodul abbildet oder - im Falle eines baumartigen Subsystems - das „Blatt“, welches einen Vorgänger zu verzeichnen hat. Gibt es kein vorangehendes Prozessmodul und demzufolge keine in das Subsystem einfließenden Produktflüsse, wird - im Falle eines baumartigen Subsystems die Spaltennummer des als „Wurzel“ fungierenden Prozessmoduls gespeichert.

Code 9.30: Instanzvariablen und Konstruktoren von Sub.java

```
1 public class Sub extends IvariMatrix{
2   private HashMap<Integer, Integer> mapSub = new HashMap<Integer, Integer>();
3   private int lastRow;
4   private int firstColumn;
5   private int lastColumn;
6   private ArrayList<Integer> listColumns = new ArrayList<Integer>();
7   private ArrayList<Integer> listRows = new ArrayList<Integer>();
8   private IvariMatrix subMatrix = new IvariMatrix(mapSub.size());
9   private IvariVector subVector = new IvariVector(mapSub.size());
10  private IvariVector resultSub = new IvariVector(mapSub.size());
11
12  public Sub(int n) {
13    super(n);
14  }
15
16  public Sub() {
17  }
18  //...
19 }
```

Die ArrayLists *listColumns* (vgl. Code 9.30, Zeile 6) und *listRows* (vgl. Code 9.30, Zeile 7) dienen zur Speicherung der Spalten- bzw. Zeilennummern des durchsuchten Subs. Sie werden bei der Berechnung des gesamten Produktsystems benötigt, um die Ergebnisse der Subsysteme korrekt im IvariVector *finalResult* zu positionieren (vgl. Kapitel 9.5.1: Code 9.20, Zeile 25 ff.). Die IvariMatrix *subMatrix* (vgl. Code 9.30, Zeile 8) dient zur Speicherung des Subsystems als IvariMatrix und analog dazu dient die Instanziierung vom IvariVector *subVector* (vgl. Code 9.30, Zeile 9) zur Speicherung des zugehörigen IvariVectors. Im IvariVector *resultSub* wird die Lösung des Subsystems gespeichert (vgl. Code 9.30, Zeile 10). Ein Objekt vom Typ Sub kann ohne Parameter (vgl. Code 9.30, Zeile 16) erzeugt werden oder durch Aufruf des Konstruktors der Superklasse mit einem Parameter vom primitiven Datentyp int (vgl. Code 9.30, Zeile 12). Dieser legt dann die Dimension der IvariMatrix fest.

Methode drawAndCalcSub(Sub sub, IvariMatrix scannedMatrix, IvariVector scannedVector)

Mit der Methode *drawAndCalcSub(Sub sub, IvariMatrix scannedMatrix, IvariVector scannedVector)* (vgl. Code 9.31) kann die detektierte Submatrix *subMatrix* vom Typ IvariMatrix sowie der zugehörige Vektor *subVector* vom Typ IvariVector erzeugt werden. Dafür werden das Sub *sub* sowie die durchsuchte IvariMatrix *scannedMatrix* und der zugehörige IvariVector *scannedVector* übergeben.

Code 9.31: Methode drawAndCalcSub(Sub sub, IvariMatrix scannedMatrix, IvariVector scannedVector)

```
1 public void drawAndCalcSub(Sub sub, IvariMatrix scannedMatrix, IvariVector scannedVector) {
2     ArrayList<Integer> subColumns = new ArrayList<Integer>();
3     ArrayList<Integer> subRows = new ArrayList<Integer>();
4     for (int i = 0; i < scannedMatrix.getSize(); i++) {
```

```
5       if (sub.getMapSub().containsKey(i)) {
6         subRows.add(i);
7         subColumns.add(sub.getMapSub().get(i));
8       }
9     }
10    sub.setListColumns(subColumns);
11    sub.setListRows(subRows);
12    int subSize = sub.getMapSub().size();
13    IvariMatrix subMatrix = new IvariMatrix(subSize);
14    IvariVector subVector = new IvariVector(subSize);
15    subVector.setValue(subRows.indexOf(sub.getLastRow()), scannedMatrix.getValue(sub.getLastRow()
          , sub.getMapSub().get(sub.getLastRow())));
16    for (int i = 0; i < subSize; i++) {
17      for (int j = 0; j < subSize; j++) {
18        subMatrix.setValue(i, j, scannedMatrix.getValue(subRows.get(i), subColumns.get(j)));
19      }
20    }
21    sub.setSubMatrix(subMatrix);
22    sub.setSubVector(subVector);
23    sub.setResultSub(subMatrix.gauss(subVector));
24  }
```

Zur Erzeugung der *subMatrix* und des zugehörigen *subVectors* werden zunächst die detektierten Zeilen und Spalten erfasst (vgl. Code 9.31, Zeile 5 ff.). Daraufhin wird dem *subVector* das IvariScalar des erzeugten Produkts des Subs in die *lastRow* übergeben (vgl. Code 9.31, Zeile 15). Anschließend werden die Elemente des Subsystems der *subMatrix* übergeben (vgl. Code 9.31, Zeile 16 ff.). Zuletzt wird das Subsystem durch Aufruf der Methode *gauss(IvariVector vector)* berechnet (vgl. Code 9.31, Zeile 23). Das Ergebnis wird der Instanzvariable *resultSub* übergeben (vgl. Code 9.31, Zeile 23).

9.5.5 LinSub.java und TreeSub.java

Die Klassen LinSub (vgl. Code 9.32, Zeile 3 ff.) und TreeSub (vgl. Code 9.32, Zeile 15 ff.) erben von der Klasse Sub. Sie dienen zur Repräsentation linearer und baumartiger Subsysteme.

Code 9.32: Kindklassen LinSub.java und TreeSub.java

```
1
2  // Kindklasse LinSub
3  public class LinSub extends Sub{
4
5    // Konstruktoren
6    public LinSub(int n) {
7      super(n);
8    }
9
10   public LinSub() {
11   }
12 }
13
14 // Kindklasse TreeSub
15 public class TreeSub extends Sub {
16
17   // Konstruktoren
```

```
18   public TreeSub(int n) {
19     super(n);
20   }
21
22   public TreeSub() {
23   }
24 }
```

Beide Klassen enthalten einen parameterlosen Konstruktor sowie einen von der Superklasse geerbten Konstruktor, dessen Parameter zur Festlegung der Dimension der Intervallmatrizen dient.

Kapitel 10

Verifikation der Methoden und Ableitung eines allgemeinen Verfahrens

Die Algorithmen werden an Testreihen erprobt, analysiert und verifiziert. In Kapitel 10.1 wird die Verifikation der implementierten Solver vorgestellt. Kapitel 10.2 dient zur Erläuterung der Ergebnisse diverser Solver hinsichtlich unterschiedlicher Aspekte der zu berechnenden Produktsysteme wie die damit einhergehenden Matrixtypen, die Position der Prozessmodule innerhalb des Produktsystems sowie die Systemstruktur der Produktsysteme. In Kapitel 10.3 wird die Komplexität der Solver thematisiert. Aus den Erkenntnissen der Verifikation, der Analyse und des Rechenzeitaufwands werden in Kapitel 10.4 die Methoden modifiziert und auf Basis dieser Methoden dann ein allgemeines Verfahren bestimmt. Dieses Verfahren kann zur Berechnung der intervallbasierten Ökobilanzierung verwendet werden.

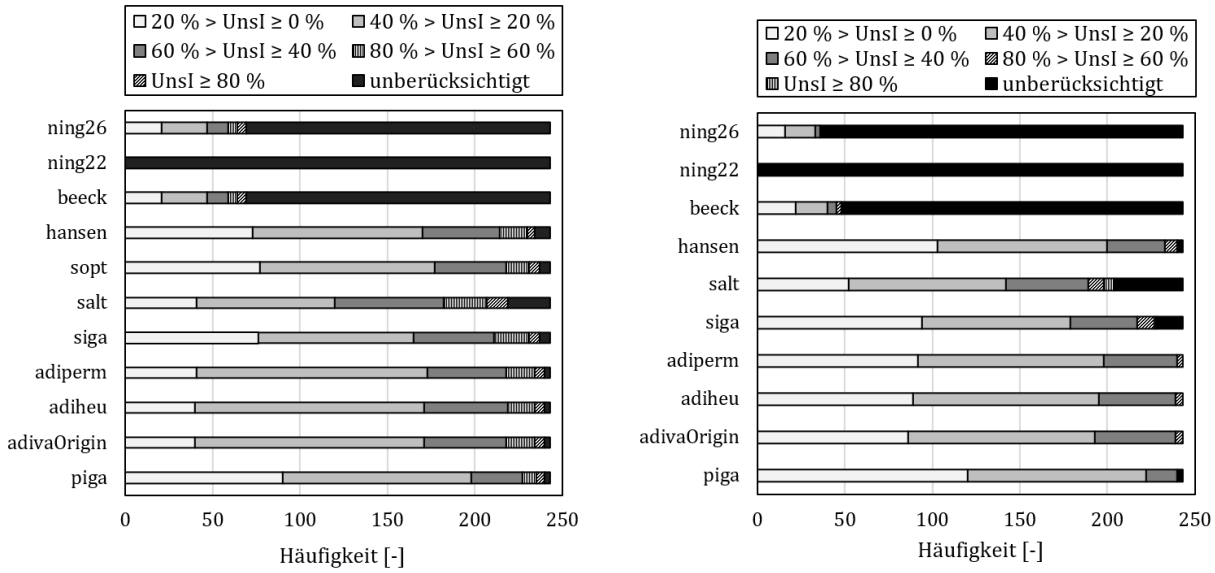
10.1 Verifikation der Gleichungslöser

Die Parameterstudien dienen zur Erprobung der Funktionalität der Gleichungslöser sowie zum Vergleich der Intervallweiten der Ergebnisse (Erläuterung zum Konzept der Parameterstudien vgl. Kapitel 8.6). Im Folgenden wird anhand der Testreihen *3x3-linear* und *3x3-baum* die Anwendbarkeit der implementierten Methoden für das entwickelte intervallbasierte Verfahren zur ökologischen Bilanzierung analysiert. Je Testreihe ergeben sich 81 unterschiedliche Produktsysteme, die durchnummeriert sind und als „testGP1“ bis „testGP81“ bezeichnet werden. Detaillierte Informationen zu den jeweiligen Produktsystemen befinden sich im Anhang B.

In Kapitel 10.1.1 wird die Eignung der Solver am Beispiel der Testreihen *3x3-linear* und *3x3-baum* thematisiert. Auf das Verfahren von Nirmala wird dabei in Kapitel 10.1.2 gesondert eingegangen. Die Eignung der Solver *beeck* und *ning26* wird in Kapitel 10.1.3 behandelt. Für eines der Produktsysteme der Testreihe *3x3-linear* wird von keinem Solver ein Ergebnis zurückgegeben aufgrund einer Ausnahme, die im Zuge einer Kehrwertberechnung auftritt. Dies wird in Kapitel 10.1.4 vorgestellt. Bei einigen Analysen werden aus Gründen der Vergleichbarkeit manche Produktsysteme ausgenommen, da einzelne Solver für diese Produktsysteme kein Ergebnis zurückgegeben haben, sondern eine Exception ausgelöst wurde; dies gilt insbesondere für den Solver *salt*. Eine Auswahl ausgenommener Produktsysteme wird gesondert in Kapitel 10.1.5 ausgewertet. Abschließend werden in Kapitel 10.1.6 die Ergebnisse der Testreihen *3x3-linear* und *3x3-baum* zusammengefasst.

10.1.1 Eignung der implementierten Solver

Die Auswertung der Testreihen *3x3-linear* und *3x3-baum* zeigt, dass die zurückgegebenen Ergebnisse aller Methoden die festgelegten Gültigkeitskriterien („Gültigkeitskriterien“ vgl. Kapitel 8.4.2) im Rahmen ihrer jeweiligen Gültigkeitsgrenzen einhalten (spezifische Gültigkeitsgrenzen der jeweiligen Solver vgl. Tabelle 8.1 in Kapitel 8.5).



(a) 243 *UnsI*-Werte der Testreihe *3x3-linear*.

(b) 243 *UnsI*-Werte der Testreihe *3x3-baum*.

Abbildung 10.1: Häufigkeit der *UnsI*-Werte nach Kategorien der Testreihen *3x3-linear* und *3x3-baum*.

In Abbildung 10.1a sind die Häufigkeit der *UnsI*-Werte aller Skalierungsfaktoren der Testreihe *3x3-linear* und in Abbildung 10.1b die Häufigkeit der *UnsI*-Werte aller Skalierungsfaktoren der Testreihe *3x3-baum* in Kategorien dargestellt. Die *UnsI*-Kategorien entsprechen den in Kapitel 8.5.4 festgelegten Kategorien („*UnsI*-Kategorien“ vgl. auch 8.5.4, Tabelle 8.2). Auf die zusätzliche, hier eingeführte sechste Kategorie „unberücksichtigt“ entfallen die *UnsI*-Werte von denjenigen Skalierungsfaktoren, für die eine Exception ausgelöst wurde oder welche für die jeweiligen Produktmatrizen nicht anwendbar sind. Die überwiegende Mehrheit der Solver gibt für die Produktsysteme Ergebnisse zurück. Ungültige Ergebnisintervalle werden zuverlässig abgefangen. Dies trifft insbesondere auf die Solver *ning22*, *ning26* und *beeck* zu. Beim Solver *ning22* werden Exceptions für alle Skalierungsfaktoren S_1 und S_2 der beiden Testreihen ausgelöst. Dies ist darin begründet, dass der Solver *ning22* oftmals Oberhüllen mit größeren Intervallweiten ermittelt, wodurch die Untergrenzen der Ergebnisintervalle auch negativ werden; dies ist im entwickelten Verfahren nicht zulässig und wird daher unterbunden. Insgesamt ist das Verfahren *ning22* zur Lösung des in dieser Arbeit adressierten Problems daher als weniger geeignet einzuordnen. Der Solver *beeck* liefert für M-Matrizen die Hülle der Lösungsmenge, der Solver *ning26* für invers-positive Matrizen. Beide Solver wurden auch an weiteren Matrixtypen getestet, erweisen sich allerdings für andere Matrixtypen als dem M-Matrixtyp als nicht geeignet. So wurden für andere Matrixtypen häufig schmalere Ergebnisintervalle zurückgegeben als vom referenziellen Solver *piga*.

Die *UnsI*-Werte der Skalierungsfaktoren, welche für die Testreihe *3x3-linear* von den Solvern *piga*, *siga*, *sopt* und *hansen* berechnet werden, lassen sich mit Werten von <20 % vergleichsweise häufig der Kategorie „vergleichsweise sehr sicher“ zuordnen. Die *UnsI*-Werte der Testreihe *3x3-linear*, die von den Solvern *adivaOrigin*, *adiheu* und *adiperm* kalkuliert werden, entsprechen vergleichsweise häufig der *UnsI*-Kategorie „vergleichsweise sicher“

mit Werten im Bereich von ≥20 % bis <40 %. Die durch den Solver *salt* ermittelten *UnsI*-Werte der Testreihe 3x3-linear gehören vergleichsweise häufig den *UnsI*-Kategorien ab einem Wertebereich von ≥40 % und größer an.

10.1.2 Analyse der Methode Nirmala

Eine Verifikation der Nirmala-Methode mit den in [63] aufgeführten Beispielen zeigte, dass die in dieser Arbeit festgelegten Gültigkeitskriterien nicht erfüllt werden (Gültigkeitskriterien vgl. Kapitel 8.4.2). Die Intervallgrenzen sind schmaler als die Grenzen, die mit der vollständigen Szenarioanalyse durch den Solver *piga* ermittelt werden. Damit ist die Nirmala-Methode für das in dieser Arbeit entwickelte Berechnungsverfahren zur ökologischen Bewertung nicht geeignet und wird im weiteren Verlauf nicht weiter in Betracht gezogen.

Die Hüllen der Lösungsmengen der Beispiele „Example 3.4“ und „Example 3.5“ (vgl. [63], S. 619 ff.) wurden mit dem Solver *piga* berechnet. Für das „Example 3.4“ werden die Hüllen ebenfalls mit den Verfahren von Ning (basierend auf Theorem 2.2 und 2.5) und dem Verfahren von Hansen ermittelt (vgl. [56], S. 1301 f., Anmerkung des Autors: Die Beispiele heißen hier „Example 3.2“ und „Example 3.3“). Die Hülle der Lösungsmenge von „Example 3.5“ wird ebenfalls mit den Solvern *beeck* bzw. dem Verfahren von Ning nach Theorem 2.5 und *siga* ermittelt (vgl. ebenfalls [56], S. 1301 f., Anmerkung des Autors: Das Beispiel heißt hier „Example 3.3“).

10.1.3 Analyse der Solver beeck und ning26

Der Solver *beeck* gibt bei den 16 Produktsystemen der Testreihe *3x3-linear*, deren Produktmatrizen vom Typ einer M-Matrix sind, die Hülle der Lösungsmenge zurück. Zusätzlich wird durch den Solver *beeck* bei weiteren sieben Produktsystemen (*testGP55, testGP56, testGP58, testGP59, testGP64, testGP65* und *testGP67* der Testreihe *3x3-linear*, vgl. Anhang B) die Hülle der Lösungsmenge berechnet. Die Produktmatrizen dieser sieben Produktsysteme haben mit der Produktmatrix eines weiteren Produktsystems (*testGP68* der Testreihe *3x3-linear*, vgl. auch Anhang B) gemein, dass sie invers-positiv sind, jedoch nicht den M-Matrizen angehören. Der Unterschied zwischen den sieben Produktmatrizen und dem Produktsystem *testGP68* ist, dass einzig das Produktsystem *testGP68* nicht den H-Matrizen zugeordnet werden kann. Das Produktsystem *testGP68* wird vom Solver *ning26* exakt gelöst, welches auf dem Verfahren von Ning und dem Theorem 2.6 basiert. Wenn die Produktmatrizen vor der Berechnung auf Positivität ihrer Inversen überprüft werden, bietet sich daher die Verwendung des Solvers *ning26* an.

10.1.4 Exceptions bei der Kehrwertberechnung

Die Berechnung des Produktsystems *testGP41* der Testreihe *3x3-linear* wird unterbrochen, da die Grenzwerte eines bei der Berechnung notwendigen Kehrwert-Intervalls unterschiedliche Vorzeichen aufweisen. Dies ist beim Berechnungsverfahren unzulässig. Grund hierfür ist, dass die Funktion $f(x) = \frac{1}{x}$ über keinen beidseitigen Grenzwert verfügt, wenn x gegen Null strebt. So strebt der Limes linksseitig gegen $-\infty$, rechtsseitig gegen $+\infty$:

$$\lim_{x \to 0^-} \frac{1}{det^{\mathrm{I}}} = -\infty \lim_{x \to 0^+} \frac{1}{det^{\mathrm{I}}} = +\infty$$

Der Kehrwert $\frac{1}{det^{\mathrm{I}}}$ einer möglichen Determinante det^{I} der Produktmatrix beträgt z. B. gerundet $\frac{1}{det^{\mathrm{I}}}$ = [-15,3; 1,9]. Die berechneten Grenzwerte umhüllen jedoch nicht alle möglichen Ergebnisse. So zeigt eine Kehrwertberechnung von Zwischenwerten von $\frac{1}{det^{\mathrm{I}}}$, dass diese außerhalb des durch Intervallrechnung ermittelten Intervalls liegen (vgl. Abbildung 10.2).

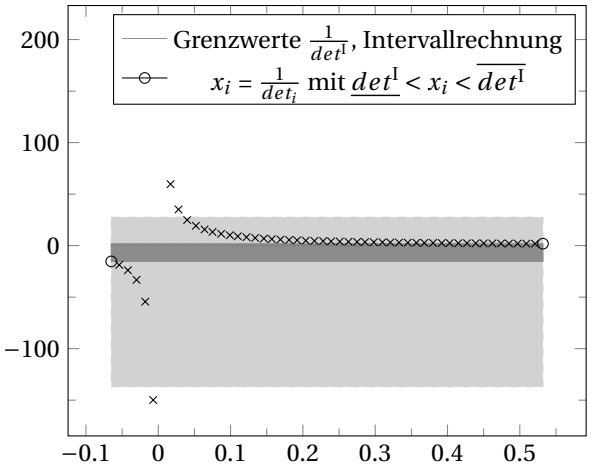


Abbildung 10.2: Problematik von Intervallen, welche die Null enthalten, am Beispiel des Kehrwerts $\frac{1}{det^I}$ der theoretisch zu erwartenden Determinante det^I des Produktsystems *testGP41*.

10.1.5 Produktsysteme mit fehlenden Rückgabewerten

Bei den vergleichenden Analysen im folgenden Kapitel 10.2 werden überwiegend acht Produktsysteme (*testGP14, testGP23, testGP47, testGP49, testGP50, testGP52, testGP53* und *testGP68*, vgl. Anhang B) der Testreihe *3x3-linear* ausgeschlossen, da bei diesen Produktsystemen die Berechnung einzelner Solver abgefangen wird. Dieser Fall tritt z. B. ein, wenn ein Intervall die Null enthält.

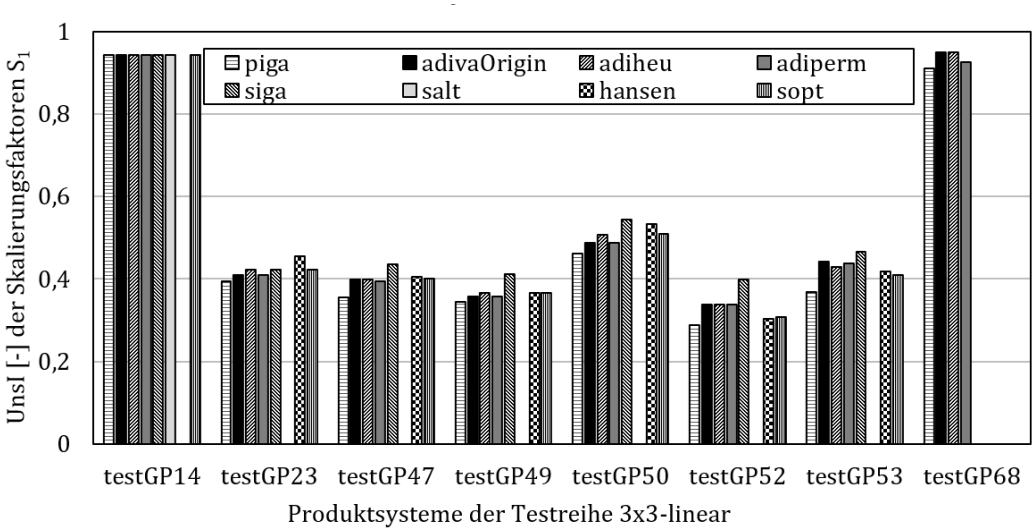


Abbildung 10.3: *UnsI*-Werte [-] der Skalierungsfaktoren S_1 ausgewählter Produktsysteme der Testreihe *3x3-linear*.

In Abbildung 10.3 sind die *UnsI*-Werte dieser Produktsysteme exemplarisch anhand der Skalierungsfaktoren S_1 dargestellt, fehlende Säulen beschreiben hier ein Fehlen des Rückgabewerts. Beim Solver *salt* wurde bei sieben der acht Produktsysteme eine Exception ausgelöst. Die Produktsysteme *testGP14* und *testGP68* weisen sehr große *UnsI*-Werte auf, womit diese gemäß der *UnsI*-Kategorien daher als „vergleichsweise sehr unsicher“ eingestuft werden. Die Ergebnisse dieser beiden fiktive Produktsysteme ist demzufolge so vage, dass hier bei einem realen System für verwendbare Aussagen eine vertiefte Recherche erforderlich wäre. Die übrigen Produktsysteme weisen

mit Werten zwischen ca. 0,30 und ca. 0,54 geringe bis moderate *UnsI*-Werte auf, welche den *UnsI*-Kategorien „vergleichsweise sicher“ bis „vergleichsweise mäßig unsicher“ zuzuordnen sind. In Abbildung 10.4 sind die Intervalle der Skalierungsfaktoren S_1 der übrigen Produktsysteme dargestellt. Die Ergebnisintervalle der Solver weichen nur geringfügig voneinander ab.

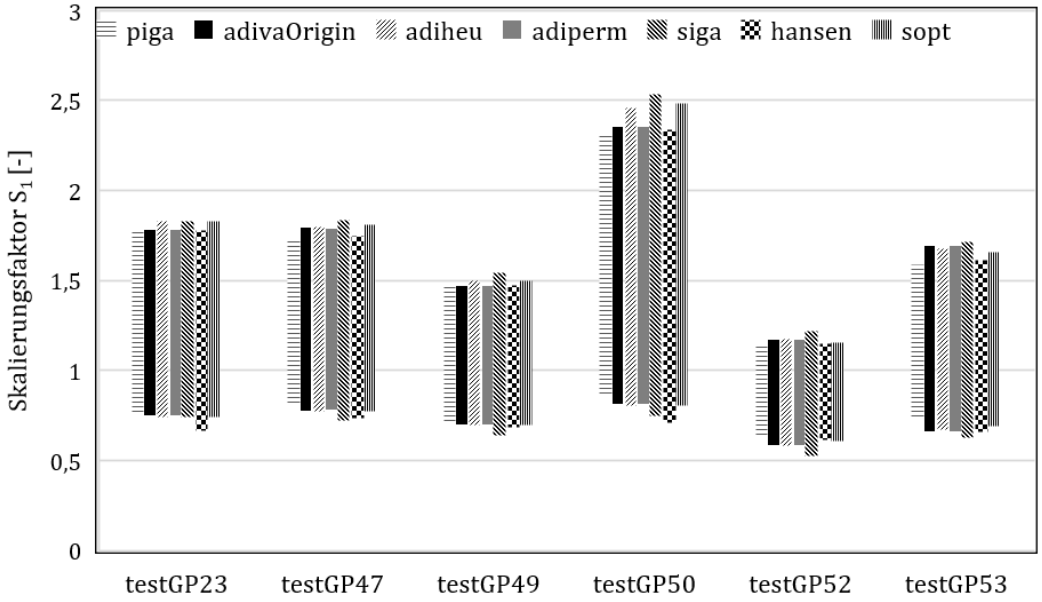


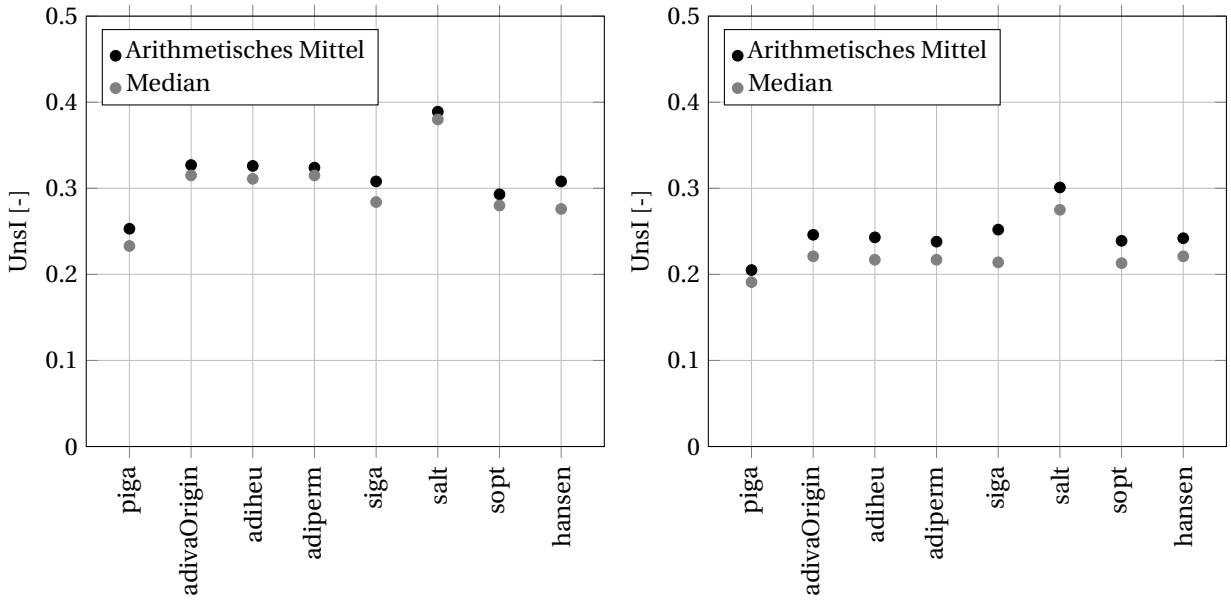
Abbildung 10.4: Ergebnisintervalle der Skalierungsfaktoren S_1 von Produktsystemen der Testreihe *3x3-linear*.

10.1.6 Übersicht der Ergebnisse

Die Ergebnisse der Gleichungslöser, die für die Testreihen *3x3-linear* sowie *3x3-baum* berechnet werden, wurden zunächst ungeachtet des Matrixtyps, der Anordnung des Prozessmoduls im Produktsystem sowie der Produktsystemstruktur vergleichend gegenübergestellt. Die Solver *beeck* und *ning26* sind nicht auf alle Matrixtypen anwendbar, die im Zuge der fiktiven Produktsysteme der Testreihen vertreten sind. Deshalb werden die beiden Solver aus Gründen der Vergleichbarkeit in diesem Abschnitt nicht berücksichtigt.

In Abbildung 10.5 sind die Ergebnisse der Testreihen *3x3-linear* und *3x3-baum* dargestellt. Die Analyse der Testreihe *3x3-linear* zeigt, dass der Median der *UnsI*-Werte des Solvers *piga* bei ca. 0,23 liegt. Die Mediane der *UnsI*-Werte der Solver *sopt*, *siga* und *hansen* sind mit ca. 0,28 ebenfalls vergleichsweise gering. Die Mediane der *UnsI*-Werte der Intervall-Adjunkten-Verfahren (Solver *adivaOrigin*, *adiheu* und *adiperm*) liegen etwas erhöht bei ca. 0,31. Der Median der *UnsI*-Werte des Solvers *salt* ist im Vergleich am größten mit einem Median von ca. 0,38. Der Median der *UnsI*-Werte der Testreihe *3x3-baum* beträgt beim Solver *piga* ca. 0,19, wobei die Solver *siga* und *sopt* mit einem Median von ca. 0,21 und die Solver *adivaOrigin*, *adiheu*, *adiperm* und *hansen* mit einem Median der *UnsI*-Werte von ca. 0,22 im Vergleich nur geringfügig größer sind.

Vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* weichen die berücksichtigten Solver bei der Testreihe *3x3-linear* zwischen minimal ca. 15,8 % (Solver *sopt*) und maximal ca. 53,8 % (Solver *salt*) ab (vgl. Abbildung 10.6a). Die arithmetischen Mittel der *UnsI*-Werte der Solver *adivaOrigin*, *adiperm* und *adiheu* weisen Abweichungen auf von ca. 28,1 % (Solver *adiperm*) bis 29,2 % (Solver *adivaOrigin*). Die Abweichungen des arithmetischen Mittels der *UnsI*-Ergebnisse vom Solver *hansen* liegen bei ca. 21,7 %. Bei der Testreihe *3x3-baum* weicht das arith-



(a) 219 *UnsI*-Werte der Testreihe *3x3-linear*.

(b) 186 *UnsI*-Werte der Testreihe *3x3-baum*.

Abbildung 10.5: Arithmetische Mittel [-] und Mediane [-] von *UnsI*-Werten der Testreihen *3x3-linear* und *3x3-baum*.

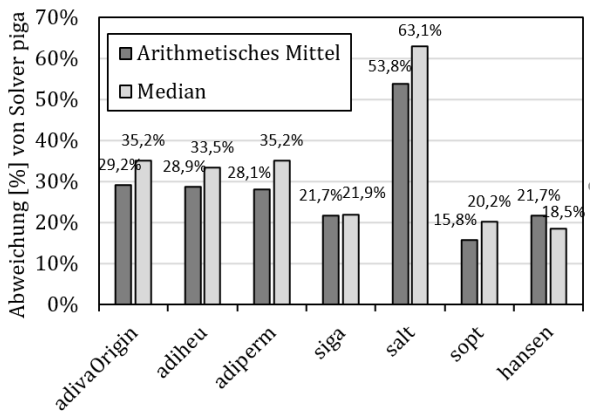
metische Mittel der *UnsI*-Werte des Solvers *adiperm* am wenigsten von dem des Solvers *piga* ab mit ca. 16,1 %, während das arithmetische Mittel der *UnsI*-Werte vom Solver *salt* die größten Abweichungen von ca. 46,8 % aufweist (vgl. Abbildung 10.6b). Tendenziell weichen die *UnsI*-Werte des Solvers *salt* im Mittel am stärksten von den jeweiligen Hüllen der Lösungsmengen ab.

10.2 Detaillierte Analyse der Solver

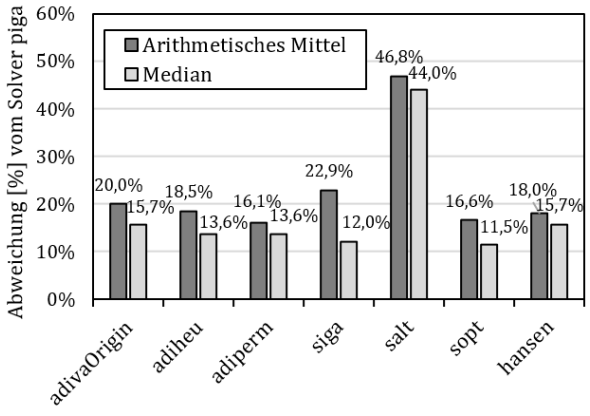
Im Folgenden werden in Kapitel 10.2.1 die Ergebnisse der Skalierungsfaktoren der erprobten Produktsysteme unter Berücksichtigung der zugehörigen Position ihrer Prozessmodule analysiert. Im Anschluss erfolgt in Kapitel 10.2.2 die Analyse der Ergebnisse der unterschiedlichen Gleichungslöser unter Berücksichtigung der spezifischen Produktsystemstrukturen. Daran anschließend werden in Kapitel 10.2.3 die Ergebnisse der Testreihen unter Berücksichtigung der Matrixtypen untersucht.

10.2.1 Auswirkungen durch die Position der Prozessmodule

Die Auswirkungen durch die Position der Prozessmodule wird anhand der Testreihe *3x3-linear* untersucht. Die Testreihe *3x3-baum* wird nicht berücksichtigt, da bei dieser Testreihe nur zwei Prozessmodule aufeinanderfolgen. Die Mediane der einzelnen *UnsI*-Skalierungsfaktoren der untersuchten Produktsysteme sind in Abbildung 10.7 illustriert. Der Median der *UnsI*-Werte beträgt beim Skalierungsfaktor S_3 des Prozessmoduls M_3 0,111 (Solver *piga*). Die Mediane der *UnsI*-Werte der übrigen Solver weichen hiervon ab mit einem Wert von mindestens 0,116 um ca. 4,5 % (Solver *sopt*) und maximal bis zu ca. 126 % (Solver *adiheu*) mit einem Wert von ca. 0,25. Beim Skalierungsfaktor S_3 ist zu erkennen, dass insbesondere die *UnsI*-Mediane der Solver *siga* und *sopt* vergleichsweise gering sind, während die Ergebnisse der Solver auf Basis der Intervall-Adjunkten-Verfahren erhöhte *UnsI*-Mediane aufweisen.



(a) :219 *UnsI*-Werte der Testreihe *3x3-linear*.



(b) :186 *UnsI*-Werte der Testreihe *3x3-baum*.

Abbildung 10.6: Abweichung [%] der arithmetischen Mittel sowie der Mediane von den Testreihen *3x3-linear* und *3x3-baum* diverser Gleichungslöser im Vergleich zu den Ergebnissen des Solvers *piga*.

Der Median der *UnsI*-Werte vom Skalierungsfaktor S_2 des Solvers *piga* ist mit einem Wert von 0,245 größer als der entsprechende Wert für S_3. Die übrigen Solver weichen bei S_2 mit *UnsI*-Medianen von 0,287 um mindestens ca. 17 % (Solver *adiperm*) und maximal bis zu 70 % (Solver *salt*) ab. Der Median der *UnsI*-Werte beträgt für den Skalierungsfaktor S_1 beim Solver *piga* 0,337, die übrigen Solver weichen mit einem Median der *UnsI*-Werte von 0,378 um mindestens ca. 12 % (Solver *hansen*) und maximal bis zu ca. 31 % (Solver *siga*) mit einem Wert von ca. 0,44 ab.

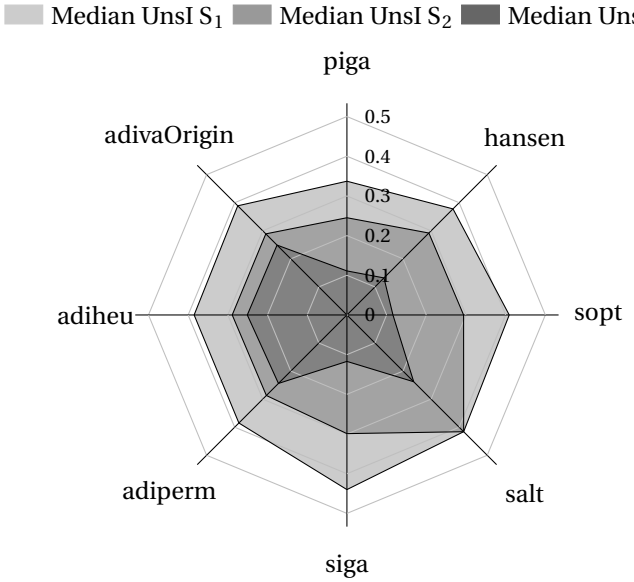


Abbildung 10.7: Mediane [-] von 219 *UnsI*-Werten der Testreihe *3x3-linear*.

Dies zeigt, dass sich die *UnsI*-Werte der Skalierungsfaktoren der Testreihe *3x3-linear* durchschnittlich vom zuletzt angeordneten Prozessmodul zu den vorangehenden Prozessmodulen erhöhen. Dies ist darin begründet, dass die Skalierungsfaktoren *vorangehender* Prozessmodule in Abhängigkeit von den Skalierungsfaktoren *nachfolgender* Prozessmodule bestimmt werden, wobei auch deren Unsicherheiten übernommen werden (vgl. hierzu auch

den „Unsicherheitsfluss“ in Kapitel 6.2.8). Dies ist auch in Abbildung 10.7 zu erkennen. So nimmt der Betrag der *UnsI*-Mediane von S_3 zu S_1 - dargestellt als Fläche - zu. Ebenso ist zu erkennen, dass sich die Verbindungslinien zwischen den *UnsI*-Medianen der Solver von den Skalierungsfaktoren S_3 zu den Skalierungsfaktoren S_1 der Form eines Kreises annähern - die Diskrepanzen zwischen den Ergebnissen der einzelnen Solver nehmen folglich im Mittel von den Skalierungsfaktoren S_3 zu den Skalierungsfaktoren S_1 ab.

10.2.2 Berücksichtigung der Produktsystemstruktur

Im Folgenden werden die Produktsysteme der Testreihen *3x3-linear* und *3x3-baum* hinsichtlich der Systemstruktur der Produktsysteme unterschieden und Produktsysteme mit gleichartiger Systemstruktur gesondert untersucht. Dadurch soll analysiert werden, ob für spezielle Produktsystemstrukturen einzelne Gleichungslöser besonders geeignet sind.

Produktsysteme mit schleifenartiger Struktur durch Verzweigung

In den Testreihen *3x3-linear* und *3x3-baum* weisen jeweils sieben Produktsysteme ausschließlich einfache oder mehrfache Schleifen durch Verzweigung auf. Eines der Produktsysteme der Testreihe *3x3-linear* (*testGP14*) wird aus dem weiteren Vergleich ausgeschlossen. Grund hierfür ist, dass beim Solver *hansen* eine Exception ausgelöst wurde. In Abbildung 10.8 sind die prozentualen Abweichungen der arithmetischen Mittel der *UnsI*-Werte unterschiedlicher Gleichungslöser von dem arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* dargestellt.

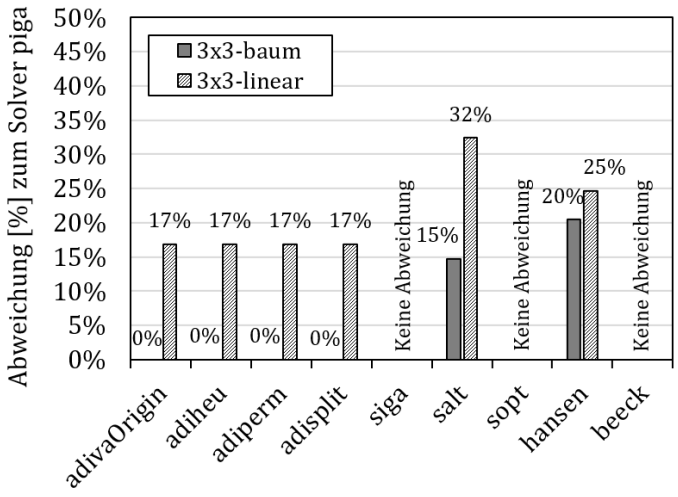


Abbildung 10.8: Abweichung der arithmetischen Mittel der *UnsI*-Werte von Produktsystemen mit Verzweigung von dem des Solvers *piga* (18 *UnsI*-Werte der Testreihe *3x3-linear* sowie 21 der *UnsI*-Werte der Testreihe *3x3-baum*).

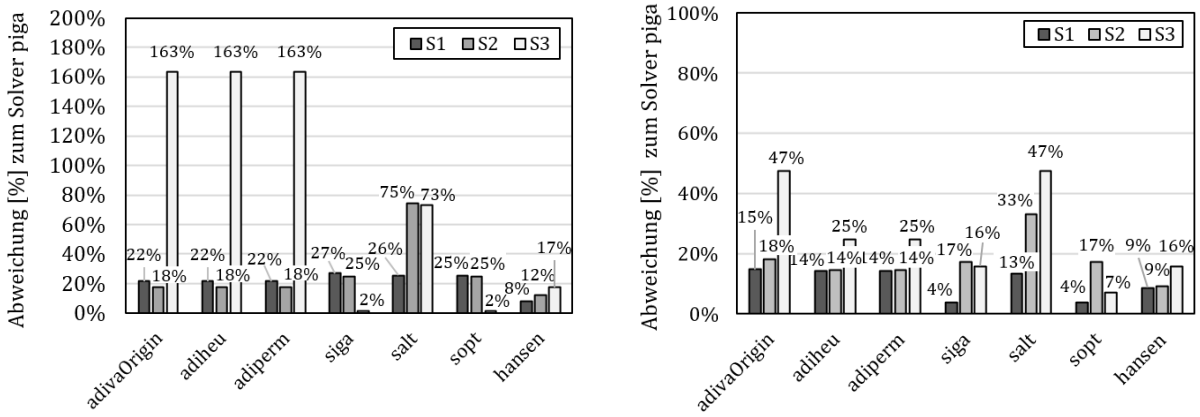
Bei der Testreihe *3x3-baum* weichen einzig die Ergebnisse der Solver *salt* und *hansen* um ca. 15 % (Solver *salt*) und um ca. 20 % (Solver *hansen*) von der Hülle der Lösungsmenge ab. Bei der Testreihe *3x3-linear* weisen die Ergebnisse dieser beiden Solver prozentuale Abweichungen von ca. 32 % (Solver *salt*) bzw. ca. 25 % (Solver *hansen*) auf; daneben weichen auch die Ergebnisse der Intervall-Adjunkten-Verfahren um ca. 17 % von den Ergebnissen des Solvers *piga* ab.

Produktsysteme mit schleifenartiger Struktur durch Verzweigung können den Z-Matrizen zugeordnet werden (Einordnung der Produktmatrizen zu Matrixtypen vgl. Kapitel 7.6). In diesem Fall entsprechen alle berücksich-

tigten Produktsysteme auch einer M-Matrix. Demzufolge wird durch die Solver *siga* und *beeck* die Hülle der Lösungsmenge ermittelt, womit diese Ergebnisse keine Abweichungen zu den Resultaten des Solvers *piga* aufweisen. Dieses Ergebnis deckt sich mit der Empfehlung von Hansen (Eignung diverser Methoden je Matrixtyp nach Hansen vgl. Kapitel 8.5.5), der für M-Matrizen das Intervall-Gauß-Verfahren empfiehlt. Die Produktmatrizen von schleifenartigen Produktsystemen durch Verzweigung sind jedoch nicht ausschließlich vom Typ einer M-Matrix, weswegen für diese Produktsystemstrukturen nicht allgemein das Intervall-Gauß-Verfahren oder die Methode von Beeck als das am besten geeignete Verfahren festgelegt werden kann.

Produktsysteme mit Pseudoschleifen

In der Testreihe *3x3-linear* weisen sieben Produktsysteme Pseudoschleifen auf, bei denen ausschließlich einfache oder mehrfache Pseudoschleifen vorkommen. In der Testreihe *3x3-baum* weisen 15 Produktsysteme eine solche Struktur auf, wobei ein Produktsystem aufgrund einer Exception bei einem der Solver aus der weiteren Analyse ausgeschlossen wird. Die Produktmatrizen von Produktsystemen mit Pseudoschleifen können nicht den Z-Matrizen zugeordnet werden und entsprechen daher auch nicht den M-Matrizen (Einordnung der Matrixtypen diverser Produktsystemstrukturen vgl. Kapitel 7.6). In diesem Fall entsprechen alle genannten Produktmatrizen einer H-Matrix.



(a) :21 *UnsI*-Werte der Testreihe *3x3-linear*.

(b) :42 *UnsI*-Werte der Testreihe *3x3-baum*.

Abbildung 10.9: Abweichung [%] der arithmetischen Mittel der *UnsI*-Werte von Produktsystemen mit Pseudoschleifen von dem des Solvers *piga* (21 *UnsI*-Werte der Testreihe *3x3-linear* und 42 *UnsI*-Werte der Testreihe *3x3-baum*.

In Abbildung 10.9a sind die Abweichungen der arithmetischen Mittel der *UnsI*-Werte der Skalierungsfaktoren S_1, S_2 und S_3 von den jeweiligen arithmetischen Mitteln der Skalierungsfaktoren der *UnsI*-Werte des Solvers *piga* der Testreihe *3x3-linear* dargestellt, in Abbildung 10.9b sind analog die Abweichungen der Testreihe *3x3-baum* abgebildet.

Mit dem Solver *hansen* werden für beide Testreihen vergleichsweise schmale Ergebnisintervalle kalkuliert mit geringen Abweichungen zum Solver *piga*. So betragen die Abweichungen bei den Produktsystemen der Testreihe *3x3-linear* im Mittel zwischen 8 % (*UnsI-Werte* der Skalierungsfaktoren S_1 der Pozessmodule M_1) und 17 % (*UnsI*-Werte der Skalierungsfaktoren S_3 der Prozessmodule M_3). Die Abweichungen der *UnsI*-Werte der Solver *siga* und *sopt* betragen allerdings für die Skalierungsfaktoren S_3 im Mittel nur 2 %. Die Intervall-Adjunkten-Verfahren weisen hingegen vergleichsweise große Abweichungen auf, wobei hier insbesondere die *UnsI*-Werte der Skalierungsfaktoren S_3 mit Abweichungen von ca. 163 % zu nennen sind.

Bei den Produktsystemen der Testreihe *3x3-baum* weicht das arithmetische Mittel der *UnsI*-Werte des Solvers *sopt* von den Skalierungsfaktoren S_1 und S_3 um ca. 4 % (arithmetisches Mittel von S_1) und ca. 7 % (arithmetisches Mittel von S_3) vergleichsweise am geringsten von dem des Solvers *piga* ab. Dagegen weist das arithmetische Mittel der *UnsI*-Werte der Skalierungsfaktoren S_2 des Solvers *hansen* die geringsten Abweichungen mit ca. 9 % zum arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* auf. Eine Betrachtung der Abweichungen des Solvers *sopt* zeigt, dass eine Kombination unterschiedlicher Solver zu einer Reduktion der Abweichungen führen kann. So weist bspw. der Solver *sopt* für die *UnsI*-Werte der Skalierungsfaktoren S_3 (Testreihe *3x3-baum*) im Mittel Abweichungen zum Solver *piga* von ca. 7 % auf, während die Solver *siga* und *salt* Abweichungen von 16 % (Abweichungen der Ergebnisse des Solvers *siga*) und 47 % (Abweichungen der Ergebnisse des Solvers *salt*) aufweisen.

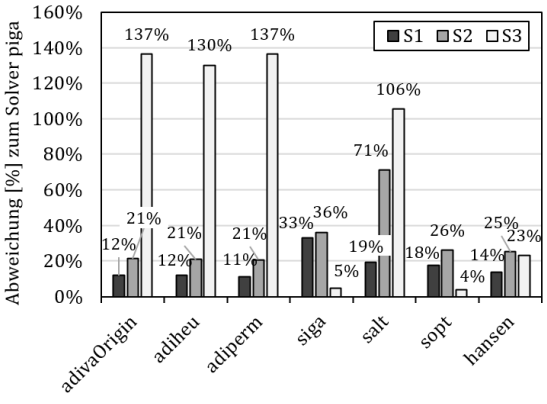
Insgesamt lässt sich anhand der Ergebnisse der beiden Testreihen kein einzelner Gleichungslöser identifizieren, der für Produktsysteme mit Pseudoschleifen als besonders geeignet gelten kann. Vielmehr sollte für Produktsysteme mit Pseudoschleifen eine Kombination mehrerer Solver in Betracht gezogen werden.

Produktsysteme mit Mischstrukturen

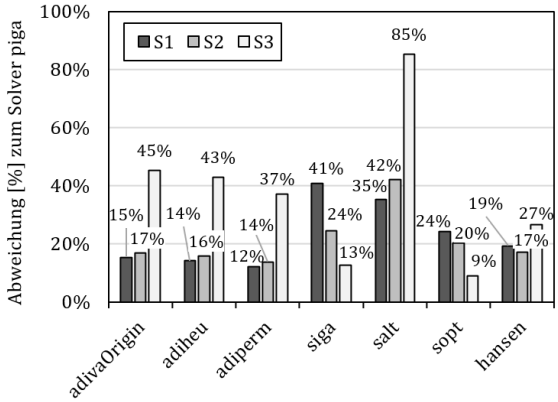
In der Testreihe *3x3-linear* verfügen 49 der Produktsysteme über Produktsysteme mit Mischstrukturen. Dies bedeutet, dass die Produktsysteme zwei oder mehr unterschiedliche Arten von Systemstrukturen in Kombination enthalten. Im folgenden Vergleich werden 44 Produktsysteme von 49 Produktsystemen mit gemischten Systemstrukturen der Testreihe *3x3-linear* berücksichtigt sowie 36 Produktsysteme von 50 Produktsystemen mit Mischstrukturen der Testreihe *3x3-baum*, da hier vereinzelt Solver keinen Rückgabewert liefern.

In Abbildung 10.10a sind die Abweichungen der arithmetischen Mittel der *UnsI*-Werte der Skalierungsfaktoren S_1, S_2 und S_3 von den jeweiligen arithmetischen Mitteln der *UnsI*-Werte des Solvers *piga* der Testreihe *3x3-linear* dargestellt, in Abbildung 10.10b sind analog hierzu die entsprechenden Abweichungen der Testreihe *3x3-baum* abgebildet. Durch den Solver *hansen* werden bei beiden Testreihen vergleichsweise schmale Ergebnisintervalle mit geringen Abweichungen zum Solver *piga* berechnet. So betragen die Abweichungen bei den Produktsystemen der Testreihe *3x3-linear* im Mittel zwischen 14 % (*UnsI*-Werte der Skalierungsfaktoren S_1 der Pozessmodule M_1) und 25 % (*UnsI*-Werte der Skalierungsfaktoren S_2 der Prozessmodule M_2). Die Abweichungen der *UnsI*-Werte der Solver *siga* und *sopt* betragen allerdings für die Skalierungsfaktoren S_3 im Mittel nur 4 % (Solver *sopt*) bzw. 5 % (Solver *siga*). Die Intervall-Adjunkten-Verfahren weisen hingegen vergleichsweise große Abweichungen auf, wobei hier insbesondere die *UnsI*-Werte der Skalierungsfaktoren S_3 mit Abweichungen von bis zu ca. 137 % (Solver *adivaOrigin* und *adiperm*) zu nennen sind. Die Analyse der Testreihe *3x3-baum* zeigt, dass die arithmetischen Mittel der *UnsI*-Werte des Solvers *hansen* zwischen 17 % (mittlere Abweichung der *UnsI*-Werte der Skalierungsfaktoren S_2) und 27 % (mittlere Abweichung der *UnsI*-Werte der Skalierungsfaktoren S_3) von den Ergebnissen des Solvers *piga* abweichen. Der Solver *salt* hingegen weist mittlere Abweichungen von bis zu 85 % (mittlere Abweichung der *UnsI*-Werte der Skalierungsfaktoren S_3) auf. Die Ergebnisse der Solver *adivaOrigin*, *adiheu* und *adiperm* weichen zwischen 12 % (mittlere Abweichung der *UnsI*-Werte der Skalierungsfaktoren S_1 des Solvers *adiperm*) und im Mittel ca. 45 % (mittlere Abweichung der *UnsI*-Werte der Skalierungsfaktoren S_3 des Solvers *adivaOrigin*) ab.

Die *UnsI*-Werte der Skalierungsfaktoren des Solvers *sopt* weisen bei beiden Testreihen im Vergleich zu den Ergebnissen der Solver *salt* und *siga* reduzierte Abweichungen auf. Zusätzlich ist zu erkennen, dass insbesondere die Solver *siga* und *sopt* im Mittel vergleichsweise geringe Abweichungen der *UnsI*-Werte der Skalierungsfaktoren S_3 mit minimal ca. 9 % (Solver *sopt*) erreichen. Dagegen weichen die *UnsI*-Werte der Skalierungsfaktoren S_1 auf Basis der Intervall-Adjunkten-Verfahren mit minimal ca. 12 % (Solver *adiperm*) vergleichsweise gering ab. Dies deutet darauf hin, dass bei Produktsystemen mit Mischstrukturen eine Kombination mehrerer Solver geeignet sein kann, um möglichst schmale Ergebnisintervalle zu erhalten.



(a) :132 *UnsI*-Werte der Testreihe *3x3-linear*.



(b) :108 *UnsI*-Werte der Testreihe *3x3-baum*.

Abbildung 10.10: Abweichung [%] der arithmetischen Mittel der *UnsI*-Werte einzelner Skalierungsfaktoren im Vergleich zu den Ergebnissen des Solvers *piga* von Produktsytemen mit gemischten Systemstrukturen der Testreihen *3x3-linear* und *3x3-baum*.

10.2.3 Berücksichtigung des Matrixtyps

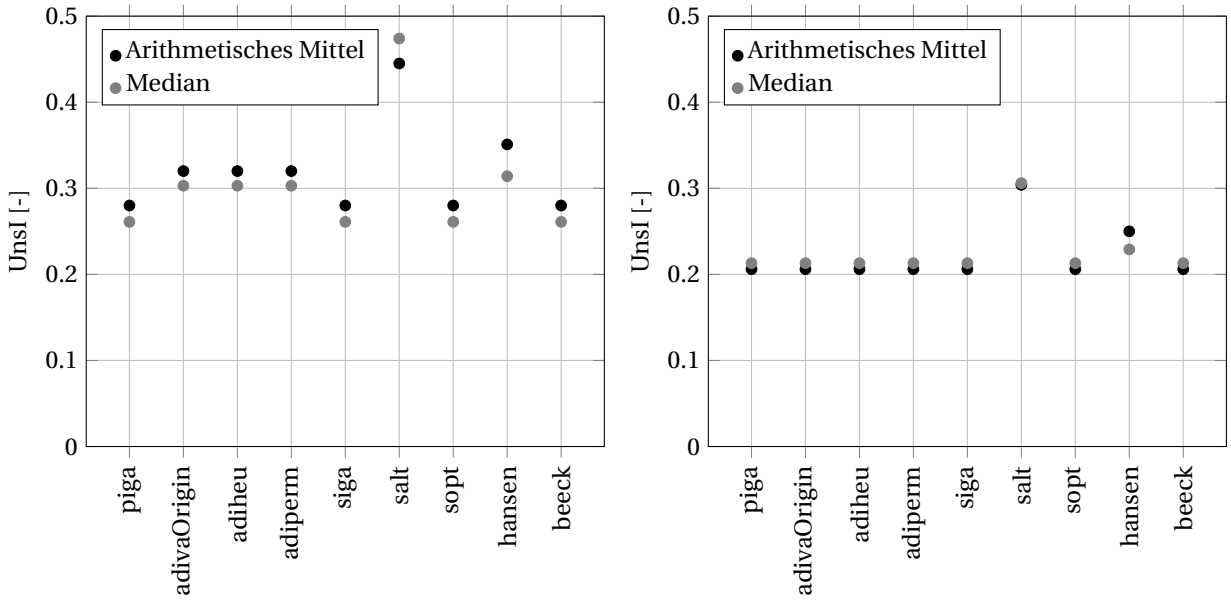
Im Folgenden werden die Produktsysteme der Testreihen *3x3-linear* und *3x3-baum* hinsichtlich des Matrixtyps der Produktmatrizen unterschieden und Produktsysteme mit Produktmatrizen gleichartigen Matrixtyps gesondert miteinander verglichen. Dadurch soll analysiert werden, ob für spezielle Matrixtypen einzelne Gleichungslöser besonders geeignet sind.

Produktmatrizen vom Typs einer M-Matrix

Jeweils 16 Produktsysteme der beiden Testreihen *3x3-linear* und *3x3-baum* lassen sich einer M-Matrix zuordnen. Im folgenden Vergleich werden jeweils zwei Produktsysteme von beiden Testreihen ausgeschlossen, da hier vereinzelt Solver keinen Rückgabewert liefern. In diesem Abschnitt wird auch der Solver *beeck* berücksichtigt, da die Produktmatrizen dem Matrixtyp einer M-Matrix entsprechen und die Produktsysteme damit im Gültigkeitsbereich des Solvers *beeck* liegen.

In Abbildung 10.11 sind die arithmetischen Mittel der *UnsI*-Werte aller Skalierungsfaktoren der Produktsysteme mit Produktmatrizen vom Typ einer M-Matrix der Testreihen *3x3-linear* (a) und *3x3-baum* (b) abgebildet. Die Solver *piga*, *beeck* und die Solver *siga* und *sopt* des Intervall-Gauß-Verfahrens liefern die Hülle der Lösungsmenge mit einem mittleren *UnsI*-Wert aller Skalierungsfaktoren von 0,28 (Testreihe *3x3-linear*) bzw. ca. 0,21 (Testreihe *3x3-baum*). Diese Ergebnisse decken sich mit dem Vorschlag der Methodenwahl nach Hansen (Einschätzung von Hansen vgl. Kapitel 8.5.5), dass das Intervall-Gauß-Verfahren bei M-Matrizen geeignet ist, sowie dem Theorem nach Beeck (Verfahren von Beeck vgl. Kapitel 3.7.3), was besagt, dass mit dem Verfahren für M-Matrizen die Hülle der Lösungsmenge ermittelt wird. Das arithmetische Mittel der *UnsI*-Werte aller Skalierungsfaktoren der Testreihe *3x3-linear* des Solvers *piga* sowie der Solver *siga*, *sopt* und *beeck* beträgt 0,28. Die Ergebnisse der Adiva-Verfahren (Solver *adivaOrigin*, *adiheu* und *adiperm*) weisen im Mittel einen *UnsI*-Wert von 0,32 auf und weichen daher um ca. 14,3 % von der Hülle der Lösungsmenge ab. Das arithmetische Mittel der *UnsI*-Werte des Solvers *hansen* beträgt ca. 0,35; die Ergebnisse dieses Solvers weichen um ca. 25,4 % vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* ab.

Für Produktsysteme, bei denen bekannt ist, dass die zugehörige Produktmatrix einer M-Matrix entspricht, bietet sich daher die Anwendung der Solver *beeck* und *siga* an. Diese Art von Matrixtyp liegt in der Regel bei linearen



(a) 42 *UnsI*-Werte der Testreihe *3x3-linear*.

(b) 42 *UnsI*-Werte der Testreihe *3x3-baum*.

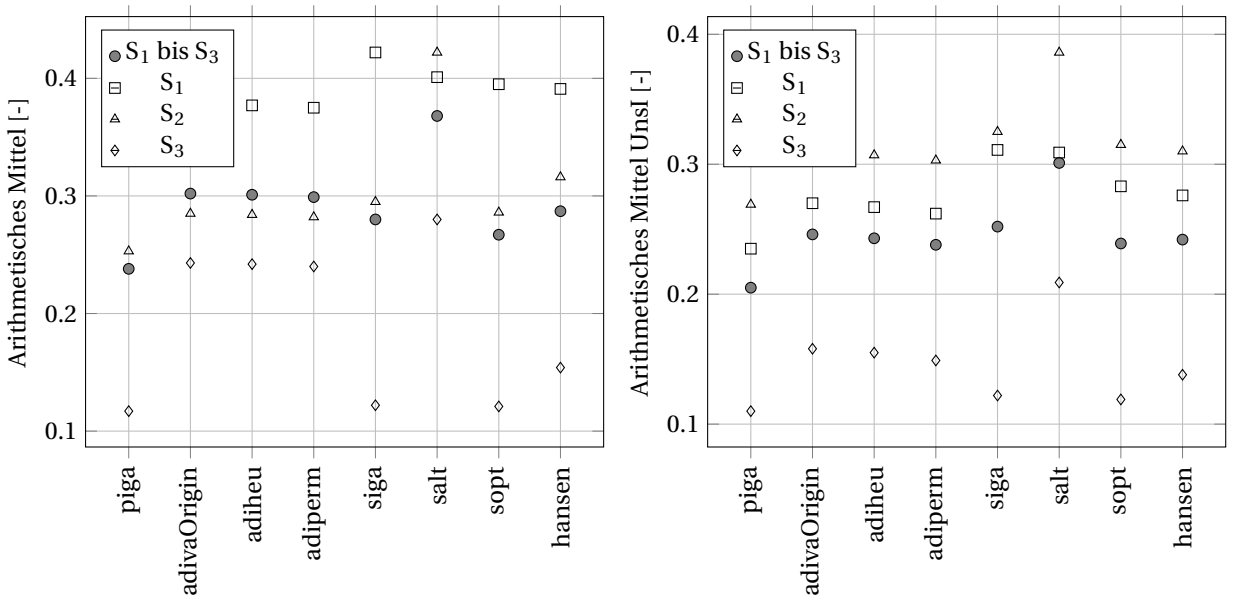
Abbildung 10.11: Arithmetische Mittel [-] und Mediane [-] von *UnsI*-Werten aller Skalierungsfaktoren, Produktsysteme mit Produktmatrizen vom M-Matrixtyp der Testreihen *3x3-linear* (a) und *3x3-baum* (b).

und baumartigen Produktsystemen vor (Klassifikation der Produktmatrizen linearer und baumartiger Produktsysteme vgl. auch Kapitel 7.6.1).

Produktmatrizen vom Typ einer H-Matrix

64 Systeme der Testreihe *3x3-linear* entsprechen einer H-Matrix. Bei der Testreihe *3x3-baum* können alle 81 Produktsysteme den H-Matrizen zugeordnet werden. Aus Gründen der Vergleichbarkeit werden vier Produktsysteme der Testreihe *3x3-linear* und 19 Produktsysteme der Testreihe *3x3-baum* nicht berücksichtigt, da für diese vereinzelt keine Ergebnisse zurückgegeben werden. In Abbildung 10.12 sind die arithmetischen Mittel der *UnsI*-Werte aller Skalierungsfaktoren der Produktsysteme mit Produktmatrizen vom Typ einer H-Matrix der Testreihen *3x3-linear* (a) und *3x3-baum* (b) abgebildet.

Die arithmetischen Mittel der *UnsI*-Werte aller Skalierungsfaktoren der Testreihe *3x3-linear* weichen mit einem Wert von 0,267 (Solver *sopt*) zwischen mindestens ca. 12,2 % vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* ab bis zu ca. 54,6 % mit einem arithmetischen Mittel der *UnsI*-Werte von 0,368 (Solver *salt*). Die *UnsI*-Werte der Skalierungsfaktoren S_1 der Intervall-Adjunkten-Verfahren weichen im Mittel mit Werten von ca. 0,377 (Solver *adivaOrigin* und *adiheu*) um ca. 9 % vergleichsweise am geringsten vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* mit einem Wert von 0,346 ab. Etwas höher sind durchschnittlich die *UnsI*-Werte der Intervall-Gauß-Verfahren; hier liegen die Abweichungen zum Solver *piga* zwischen ca. 14,2 % mit einem Wert von 0,395 (Solver *sopt*) und ca. 22 % mit Werten von 0,422 (Solver *siga*). Die Analyse der Ergebnisse der *UnsI*-Werte der S_2-Skalierungsfaktoren zeigt, dass die arithmetischen Mittel zwischen minimal ca. 10,3 % mit einem Wert von ca. 0,30 (Solver *adiperm*) und maximal um ca. 67 % mit einem Wert von ca. 0,42 (Solver *salt*) abweichen vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* mit einem Wert von 0,269. Bei den *UnsI*-Werten der Skalierungsfaktoren S_3 hingegen erzielen die Intervall-Gauss-Verfahren mit einem arithmetischen Mittel von ca. 0,12



(a) 180 *UnsI*-Werte der Testreihe *3x3-linear*.

(b) 186 *UnsI*-Werte der Testreihe *3x3-baum*.

Abbildung 10.12: Arithmetische Mittel [-] von *UnsI*-Werten der Skalierungsfaktoren S_1, S_2 und S_3, Produktsysteme mit Produktmatrizen vom Typ einer H-Matrix der Testreihen *3x3-linear* (a) und *3x3-baum* (b) berücksichtigt.

(Solver *siga* und *sopt*) im Mittel deutlich geringere *UnsI*-Werte als die Intervall-Adjunkten-Verfahren. Diese weisen mit einem arithmetischen Mittel von ca. 0,24 (Solver *adivaOrigin*, *adiheu* und *adiperm*) deutlich höhere Abweichungen von ca. 105 % auf im Vergleich zum arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* mit einem Wert von 0,117.

Die Analyse der Ergebnisse deutet darauf hin, dass Produktmatrizen, die den H-Matrizen zugeordnet werden können, keiner der untersuchten Gleichungslöser vorrangig vor einem anderen Solver verwendet werden sollte. Diese Art von Matrixtyp kann in der Regel bei Produktmatrizen von schleifenartigen Produktsystemen durch Zusammenfluss vorliegen (Klassifikation der Produktmatrizen schleifenartiger Produktsysteme vgl. Kapitel 7.6.2).

Produktmatrizen anderen Matrixtyps

Insgesamt 16 Systeme der Testreihe *3x3-linear* lassen sich weder einer M-, H- noch einer Z-Matrix zuordnen. In diese Kategorie fällt kein Produktsystem der Testreihe *3x3-baum*. Vier Produktsysteme der Testreihe *3x3-linear* werden aus der weiteren Betrachtung aus Gründen der Vergleichbarkeit ausgeschlossen, da bei diesen vereinzelt von Solvern kein Ergebnis zurückgegeben wurde. In Abbildung 10.13 sind die arithmetischen Mittel der *UnsI*-Werte aller Skalierungsfaktoren der Produktsysteme mit Produktmatrizen vom H-Matrixtyp der Testreihe *3x3-linear* abgebildet.

Die arithmetischen Mittel der *UnsI*-Werte aller Skalierungsfaktoren weichen mit einem Wert von ca. 0,41 (Solver *hansen* und *sopt*) mindestens um ca. 26,5 % vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* mit einem Wert von 0,325 ab. Die maximalen Abweichungen von ca. 52 % weisen im Mittel die *UnsI*-Werte des Solvers *salt* mit einem Wert von 0,495 auf. Die Analyse der Ergebnisintervalle der *UnsI*-Werte der S_1-Skalierungsfaktoren zeigt, dass die *UnsI*-Werte im Mittel um minimal ca. 15,6 % mit Werten von ca. 0,52 (Solver *hansen*, *adivaOrigin*,*adiheu* und *adiperm*) und maximal um ca. 50 % mit einem Wert von 0,67 (Solver *siga*) vom arithmetischen

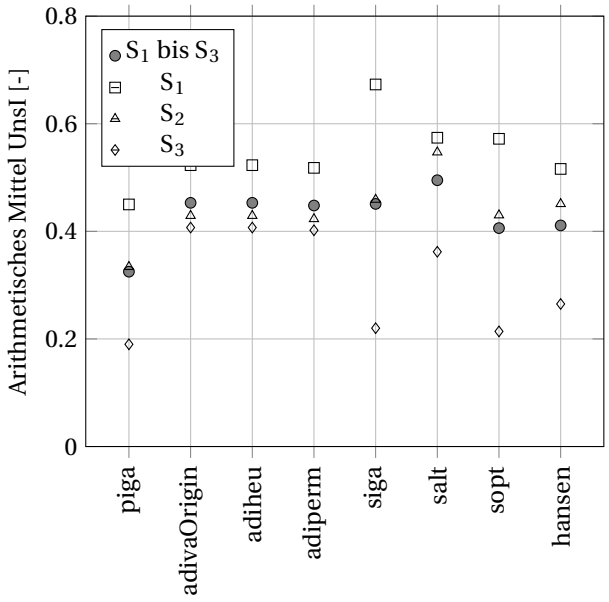


Abbildung 10.13: Arithmetisches Mittel von 36 *UnsI*-Werten aller Skalierungsfaktoren S_1, S_2 und S_3; Produktsysteme der Testreihe *3x3-linear*, die keinem M-, H- oder Z-Matrixtyp entsprechen, berücksichtigt.

Mittel der *UnsI*-Werte des Solvers *piga* mit einem Wert von 0,45 abweichen. Auch bei den Skalierungsfaktoren S_2 sind die *UnsI*-Werte der Intervall-Adjunkten-Verfahren sowie der Solver *sopt* mit arithmetischen Mitteln der *UnsI*-Werte von ca. 0,42 (Solver *adiperm*) bzw. 0,43 (Solver *adivaOrigin*, *adiheu* und *sopt*) vergleichsweise gering. Sie weichen vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* (mit einem Wert von 0,334) um ca. 27 % (Solver *adiperm*) bis 29 % (Solver *adivaOrigin*, *adiheu* und *sopt*) ab. Die *UnsI*-Werte des Solvers *salt* weichen im Mittel um etwa 64 % ab mit einem Wert von 0,547.

Die *UnsI*-Werte der Skalierungsfaktoren S_3 sind beim Solver *siga* vergleichsweise gering; sie weichen mit einem arithmetischen Mittel von 0,22 nur um ca. 15,8 % vom arithmetischen Mittel der *UnsI*-Werte des Solvers *piga* mit einem Wert von 0,19 ab. Die Intervall-Adjunkten-Verfahren weisen dagegen im Mittel erhöhte Abweichungen von ca. 114 % auf mit einem Wert von ca. 0,407 (Solver *adiheu* und *adivaOrigin*). Die Abweichungen der Ergebnisse des Solvers *hansen* sind dagegen mit ca. 39,5 % vergleichsweise moderat mit einem arithmetischen Mittel von 0,265.

Die Ergebnisse lassen den Schluss zu, dass bei Mischstrukturen und den daraus resultierenden komplizierten Matrixtypen keiner der Gleichungslöser am besten geeignet zu sein scheint. Produktmatrizen, welche sich keiner M-, H- oder Z-Matrix zuordnen lassen, können mitunter bei Pseudo-Schleifen oder schleifenartigen Produktsystemen durch Verzweigung vorkommen (Klassifizierung der Produktmatrizen vgl. auch Kapitel 7.6.2).

10.3 Rechenzeitaufwand der Methoden

Die Komplexität der implementierten Methoden beruht auf Angaben der Literatur oder sie wurde anhand der Zuweisungen der implementierten Algorithmen abgeschätzt. Der Aufwand der Methoden wurde mit weiteren Tests erprobt. In Kapitel 10.3.1 wird der Rechenzeitaufwand der implementierten Solver vorgestellt. Kapitel 10.3.2 dient zur Gegenüberstellung der Rechenzeit des Solvers *adivaOrigin* mit und ohne Anwendung der Methode *adisplit*.

10.3.1 Gleichungslöser

Es wurde der Aufwand der Gleichungslöser *piga*, *adivaOrigin*, *adiperm*, *hansen*, *siga*, *ning26*, der modifizierten Solver *picky*, *quickpick* und *quickily* sowie der Aufwand des Solvers *goodSolution* erprobt (Anmerkung: die modifizierten Solver *picky*, *quickpick* und *quickily* werden im folgenden Kapitel 10.4.1 vorgestellt; der Solver *goodSolution* stellt das abgeleitete Verfahren dar und wird in Kapitel 10.4.2 erläutert).

Zunächst werden die Gleichungslöser *piga*, *adivaOrigin* und *adiperm* getestet und die Rechenzeit erfasst. Hierfür wurden vollbesetzte, lineare Produktmatrizen geringer Dimension erstellt. Die gemessenen Rechenzeiten wurden mit kalkulierten Rechenzeiten verglichen. Die kalkulierten Rechenzeiten beruhen auf Basis der angenommenen Komplexität und einer mittleren Rechnerleistung von zwei GHz. Die gemessenen Rechenzeiten sind auf Basis einer Windows 10 Workstation mit Intel CPU E5-2630 (8 Kerne, 16 Threads, 2,4 GHz) sowie 32 GB RAM ermittelt worden. In Abbildung 10.14 sind die gemessenen Rechenzeiten der Solver *piga*, *picky*, *adivaOrigin* und *adiperm* dargestellt für die Matrixdimensionen *1* bis maximal *11* sowie die kalkulierten Rechenzeiten anhand einer geschätzten Komplexität von O(n^n) und O(2^{n^2}). Demnach ist der Solver *piga* bei einer vollbesetzten Matrix bis zu einer Matrixdimension von *6* mit *25* Intervallen geeignet (mit ca. 60 Sekunden Rechenzeit), während mit den Solvern *picky* und *adiperm* bis zu einer Matrixdimension von *7* in kurzer Zeit (ca. 36 Sekunden Rechenzeit) die Lösung berechnet werden kann. Die Rechenzeit des Solvers *adivaOrigin* beträgt bei einer Matrixdimension von *10* ca. 18 Sekunden.

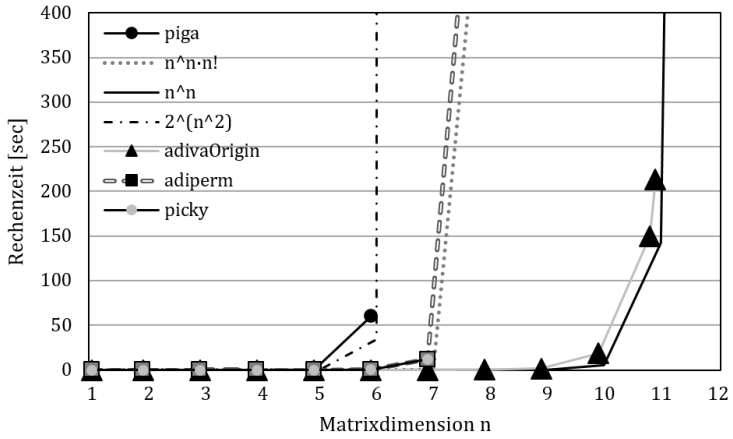


Abbildung 10.14: Rechenzeit [sec] der Solver *piga*, *picky*, *adivaOrigin* und *adiperm* in Abhängigkeit der Matrixdimension.

Abbildung 10.15 stellt die gemessenen Rechenzeiten der Solver *hansen*, *rohn* und *quickpick* dar sowie eine kalkulierte Rechenzeit anhand einer geschätzten Komplexität von O(n^4). Der Solver *hansen* braucht für eine Matrixdimension von *100* ca. 13 Sekunden, der Solver *rohn* benötigt etwa neun Sekunden Zeit. Der Solver *quickpick*, der den Solver *hansen* aufruft, weist eine vergleichbare Rechenzeit von ca. 14 Sekunden auf.

Die Solver *siga*, *ning26*, *quickily* und *goodSolution* sind Gleichungslöser mit der vergleichsweise geringsten Komplexität. Die Rechenzeit des Solvers *siga* beträgt für Matrixdimensionen von *500* ca. 0,5 Sekunden. Die Lösung wird bei Anwendung des Solvers *ning26* in ca. 18 Sekunden ermittelt, während der Solver *quickily* eine vergleichbare Rechenzeit von ca. 19 Sekunden aufweist. Der Gleichungslöser *goodSolution*, welcher die Lösung auf Basis des allgemeinen Verfahrens ermittelt, benötigt für eine Matrixdimension von *500* ca. 19 Sekunden.

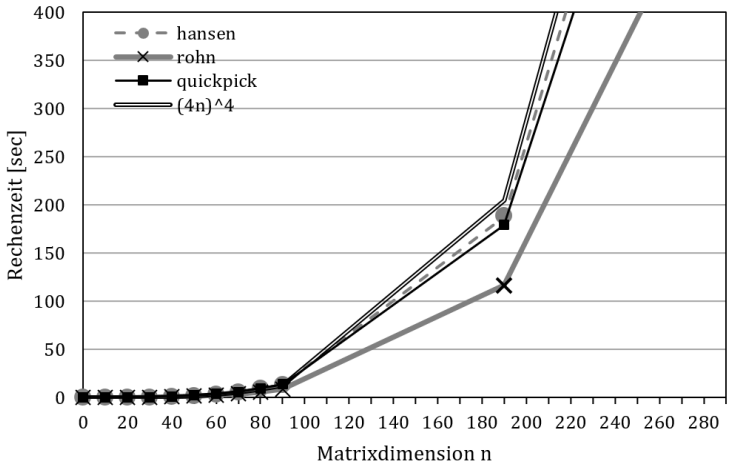


Abbildung 10.15: Rechenzeit [sec] der Solver *hansen, rohn* und *quickpick* in Abhängigkeit der Matrixdimension.

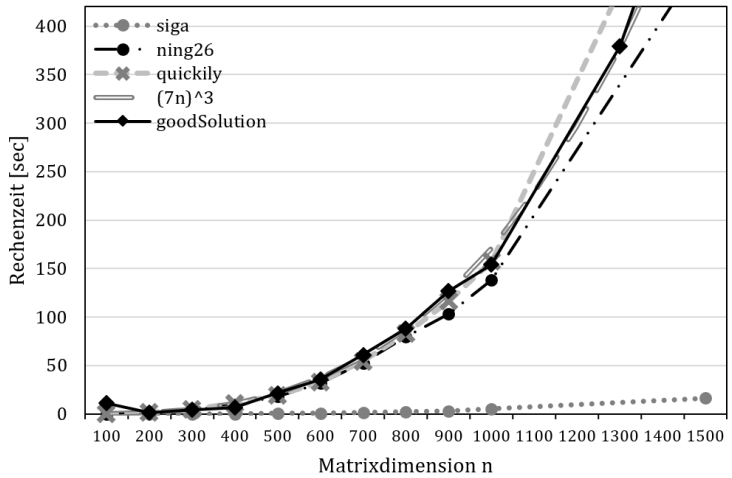


Abbildung 10.16: Rechenzeit [sec] der Solver *siga, ning26, quickily* und *goodSolution* in Abhängigkeit der Matrixdimension.

10.3.2 Methode adisplit

Die Methode *adisplit* wurde an einzelnen JUnit-Tests erprobt. Die getesteten Produktsysteme weisen zehn Prozessmodule auf und enthalten Subsysteme mit linearer und baumartiger Systemstruktur. Die Testdurchläufe der Methode *adisplit* zum Splitten von baumartigen und linearen Submatrizen sowie zum Einfügen der komprimierten Submatrizen als einzelne Prozessmodule verliefen bei diesen Tests erfolgreich. Der Solver *adivaOrigin* benötigte zum Lösen der Tests ohne vorherige Splittung der Produktmatrix ca. 13 Sekunden. Wenn die Methode *adisplit* verwendet wird, erforderte die Berechnung mit dem Solver *adivaOrigin* dagegen nur ein Bruchteil von Sekunden.

10.4 Allgemeines Verfahren zur intervallbasierten Ökobilanzierung

Auf Basis der erworbenen Erkenntnisse wurden die Solver verbessert und als alternative Gleichungslöser ebenfalls implementiert. Mit ihnen sollen möglichst schmale Ergebnisintervalle berechnet werden unter Berücksichtigung der Rechenzeit. Die modifizierten Solver werden in Kapitel 10.4.1 erläutert. In Kapitel 10.4.2 wird ein allgemeines Verfahren vorgestellt, dass zur intervallbasierten Berechnung ökologischer Bilanzen verwendet werden kann und welches die modifizierten Solver verwendet.

10.4.1 Entwicklung modifizierter Solver

Für kompliziertere Produktsystemstrukturen ließ sich mit den Analysen der *3x3-linear* und *3x3-baum* kein einzelnes Verfahren identifizieren, welches grundsätzlich als am besten geeignet gelten kann. Stattdessen stellt die Verwendung mehrerer Solver eine gute Lösung dar. Mit dem Solver *ning22* werden häufig Oberhüllen berechnet, die für das in dieser Arbeit erwünschte Verfahren als ungeeignet einzustufen sind und daher nicht berücksichtigt werden (Eignung der implementierten Solver vgl. Kapitel 10.1.1). Da sich durch die Parameterstudien bestätigte, dass der Solver *ning26* bei den invers-positiven Matrizen vergleichsweise die schmalsten Ergebnisintervalle kalkuliert (Analyse des Solvers *ning26* vgl. Kapitel 10.1.3), wurde dieser Solver bei der Entwicklung der Methoden *picky*, *quickpick* und *quickily* berücksichtigt. Er wird aufgerufen, wenn die Inverse der Produktmatrix positiv ist.

Auf Basis dieser Erkenntnisse wurde die Methode *picky* implementiert, welche die Solver *hansen*, *siga*, *salt*, *adivaOrigin*, *adiperm* und *ning26* aufruft und das jeweils schmalste Ergebnisintervall zurückgibt (Code des Solvers *picky* vgl. Anhang A). Da diese Methode somit auch Verfahren verwendet, die bei größeren Matrizen aufgrund der Rechenzeit als nicht prakikabel einzustufen sind, wurde eine weitere Methode *quickpick* implementiert, welche sich auf die Solver *siga*, *salt*, *hansen* und *ning26* beschränkt und aus diesen Gleichungslösern das jeweils schmalste Ergebnisintervall ermittelt (Code der Methode *quickpick* vgl. auch Anhang A). Diese Verfahren sind auch bei größeren Matrizen noch vergleichsweise schnell. Für sehr große Matrizen wurde zusätzlich eine weitere Methode *quickily* implementiert, welche weitgehend der Methode *quickpick* entspricht, jedoch auf den hinsichtlich der Rechenzeit aufwändigsten Solver *hansen* verzichtet (Code des Solvers *quickily* vgl. Anhang A).

In Abbildung 10.17 ist dargestellt, wie oft die Solver *picky*, *quickpick* und *quickily* sowie die bereits analysierten Gleichungslöser am Beispiel der Testreihe *3x3-linear* die schmalsten - oder zusammen mit weiteren Solvern die mit am schmalsten und damit besten - Ergebnisintervalle zurückgeben. Nicht berücksichtigt werden aus Gründen der Vergleichbarkeit diejenigen Solver, die nur gültig sind für M-Matrizen (Solver *beeck*), H-Matrizen (Solver *ning22*), invers-positive Matrizen (Solver *ning26*) sowie der auf einer vollständigen Szenarioanalyse beruhende Solver *piga*. Die abgebildeten Säulen stellen dar, wie oft mit einzelnen Solver die schmalsten - oder zusammen mit weiteren Solvern die schmalsten - Ergebnisintervalle kalkuliert werden können.

In Abbildung 10.18 sind zusätzlich die durchschnittlichen und maximalen Abweichungen zu den Ergebnissen

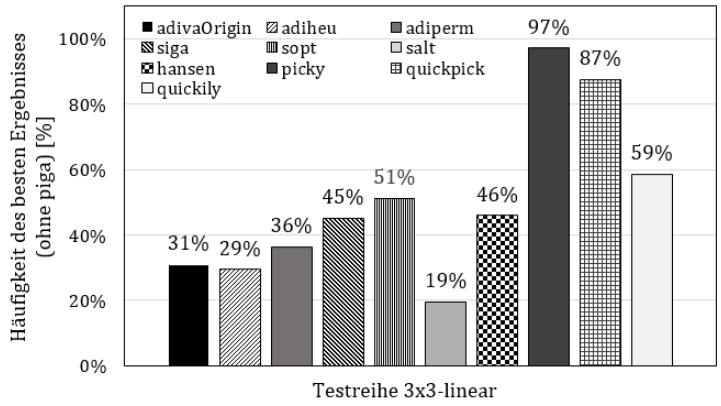


Abbildung 10.17: Prozentuale Häufigkeit der schmalsten - und damit besten - Ergebnisintervalle diverser Solver (ohne *piga*) der Testreihe *3x3-linear*.

der schmalsten Ergebnisintervalle dargestellt. Abweichungen treten dann auf, wenn ein Solver nicht das schmalste Ergebnisintervall zurückgibt. Es werden sowohl die maximalen Abweichungen zum schmalsten Ergebnisintervall ohne den Solver *piga* sowie die maximalen Abweichungen zu den Ergebnissen des Solvers *piga*, d. h. der Hülle der Lösungsmenge, abgebildet. Daneben werden auch die durchschnittlichen Abweichungen zu den Ergebnissen des Solvers *piga* dargestellt.

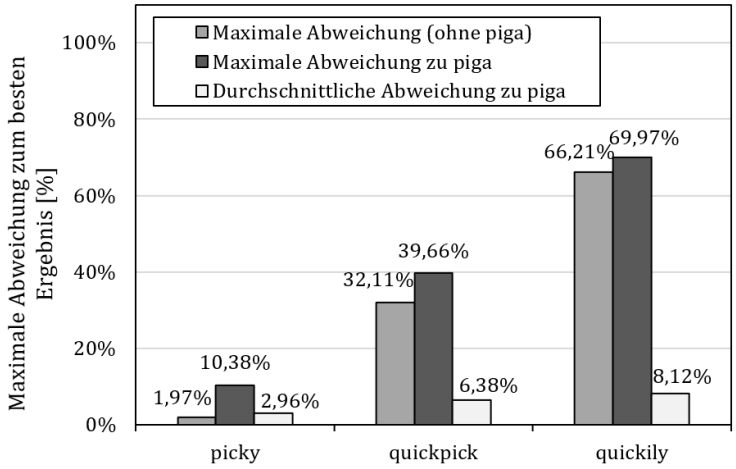


Abbildung 10.18: Maximale Abweichung zum schmalsten Ergebnisintervall (ohne *piga*-Ergebnisse), sowie die durchschnittliche und maximale Abweichung zum Ergebnis von *piga* der Solver *picky*, *quickpick* und *quickily*, wenn diese nicht das schmalste Ergebnisintervall liefern.

Für 97 % der Produktsysteme der Testreihe *3x3-linear* und 99 % der Produktsysteme der Testreihe *3x3-baum* liefert der Solver *picky* die schmalsten Ergebnisintervalle. Bei den 3 % der Produktsysteme der Testreihe *3x3-linear* bzw. den 1 % der Produktsysteme der Testreihe *3x3-baum*, bei denen andere Solver als der Solver *picky* die schmaleren Ergebnisintervalle zurückgeben, liegen die Abweichungen der Ergebnisse des Solvers *picky* zum jeweils schmalsten Ergebnisintervall (ohne Berücksichtigung der Ergebnisse von *piga*) zwischen 0,2 % und ca. 2,0 % (Testreihe *3x3-linear*) bzw. 0,1 % und 1,9 % (Testreihe *3x3-baum*). Der Solver *quickpick* ermittelt in 87 % der Tests der Testreihe *3x3-linear* das beste Ergebnis mit den schmalsten Intervallweiten. Bei den 13 % der Produktsysteme, bei denen andere Solver als der Solver *quickpick* die schmaleren Ergebnisintervalle zurückgeben, liegen die Ab-

weichungen zwischen 0,2 % und ca. 32 %. Bei der Testreihe *3x3-baum* wird durch den Solver *quickpick* in 80 % der Tests das beste Ergebnis mit den schmalsten Intervallweiten berechnet; hier weichen die Ergebnisse der 20 % der Tests, bei denen andere Solver als der Solver *quickpick* das bessere Ergebnis liefern, zwischen 0 % und ca. 30,6 % ab. Der Solver *quickily*, der sich ausschließlich auf die Ergebnisse der Intervall-Gauß-Verfahren sowie des Solvers *ning26* beschränkt, weist in 59 % der Tests der Testreihe *3x3-linear* das beste Ergebnis auf. In den übrigen 41 % der Tests weichen die zurückgegebenen Ergebnisse zwischen 0,2 % und ca. 66 % von den jeweils besten Ergebnissen ab. Bei der Testreihe *3x3-baum* liefert der Solver *quickily* in 55 % der Tests das beste Ergebnis; hier weichen die Ergebnisse von den 45 % der Tests, bei denen andere Solver als der Solver *quickpick* das bessere Ergebnis liefern, zwischen 0,2 % und ca. 79 % ab.

10.4.2 Ableitung eines allgemeinen Verfahrens

Auf Basis der implementierten Solver wurde ein allgemeines Verfahren entworfen, das zur Durchführung praktikabler, intervallbasierter Ökobilanzen verwendet werden kann. Es wurde in der Methode *goodSolution* implementiert. Die Vorgehensweise des Verfahrens ist in Abbildung 10.19 dargestellt. Um die Rechenzeit zu berücksichtigen, wird die Anzahl der Intervalle und/oder die Matrixdimension der intervallbasierten Produktmatrix abgefragt und je nach Komplexität geeignete Methoden angewandt.

Bei komplizierteren Matrizen ist hinsichtlich der Intervallweiten grundsätzlich der Solver *piga*, welcher eine vollständige Szenarioanalyse durchführt, hinsichtlich der Intervallweiten das am besten geeignete Verfahren. Der Solver ist jedoch nur bis zu einer begrenzten Anzahl von Intervallen aufgrund des Zeitaufwands verwendbar. Liegen größere Intervallmatrizen mit mehr Intervallen vor, bietet es sich an, die Produktmatrix in kleinere Submatrizen zu splitten und diese gesondert zu berechnen. In Abbildung 10.19 erfolgt dies bis zu einem bestimmten Grenzwert m. Mit der Effizienz der aktuell implementierten Methoden wird dies mit einer vergleichbaren Rechnerleistung wie der in dieser Arbeit verwendeten (verwendete Rechnerleistung vgl. Kapitel 10.3.1) bis zu einer Matrixdimension von ca. *1300* empfohlen. In der Praxis ist davon auszugehen, dass die zu berechnenden Produktsysteme nicht mehr als 1300 Prozessmodule beinhalten. Wird die Dimension von m = *1300* erreicht, wird der Solver *siga* aufgerufen. Für eine Matrixdimension von *3000* benötigt der Solver *siga* zur Berechnung der Lösung mit einer vergleichbaren Rechnerleistung ca. drei Minuten.

Liegen weniger als 1300 Prozessmodule vor, wird die Methode *adisplit* aufgerufen, durch welche die Matrixdimension der vorliegenden, intervallbasierten Produktmatrix reduziert wird. Aus den vorherigen Analysen ergibt sich, dass linear- und baumartige Produktsysteme M-Matrizen aufweisen, bei denen die Hülle der Lösungsmenge zuverlässig mit dem schnellen Intervall-Gauß-Verfahren gelöst werden kann (Analyse von Produktmatrizen vom Typ einer M-Matrix der Testreihen *3x3-linear* und *3x3-baum* vgl. Kapitel 10.2.3). Der komprimierten Produktmatrix liegen kompliziertere Produktsystemstrukturen zugrunde. Das Verfahren ruft den Solver *piga* aus, sofern die Anzahl der Intervalle unter einem bestimmten Grenzwert i liegt. Auf Basis der gemessenen und berechneten Rechenzeit wird dies mit einer vergleichbaren Rechnerleistung für maximal 20 Intervalle empfohlen, die Rechenzeit beträgt hierfür ca. $1,7$ Sekunden.

Die Submatrizen werden gesondert mit dem Solver *siga* berechnet, da dieser die baumartigen und linearen Intervallmatrizen zuverlässig lösen kann und hierfür das schnellste Verfahren darstellt. Weist die komprimierte Produktmatrix nicht mehr als i Intervalle auf, wird für diese der Solver *piga* aufgerufen. Enthält die Intervallmatrix hingegen mehr als i Intervalle und hat dennoch eine Matrixdimension $< j$, wird der Solver *picky* aufgerufen. Auf Basis der gemessenen und berechneten Rechenzeit wird dies bei einer vergleichbaren Rechnerleistung für eine maximale Matrixdimension von 7 empfohlen, die Rechenzeit beträgt hierfür ca. $11,8$ Sekunden. Größere Produktmatrizen, welche eine Matrixdimension von bis zu n aufweisen, werden stattdessen mit dem Solver *quickpick*

gelöst. Auf Basis der gemessenen und berechneten Rechenzeit wird dies mit einer vergleichbaren Rechnerleistung für eine maximale Matrixdimension von *80* empfohlen, die Rechenzeit beträgt hierfür ca. acht Sekunden. Wird auch die Matrixdimension von n überstiegen, wird hingegen der Solver *quickily* zur Berechnung verwendet. Bei einer Matrixdimension von *1300* ist bei Anwendung des Solvers *quickily* ein Rechenzeitaufwand von ca. 6,3 Minuten zu erwarten.

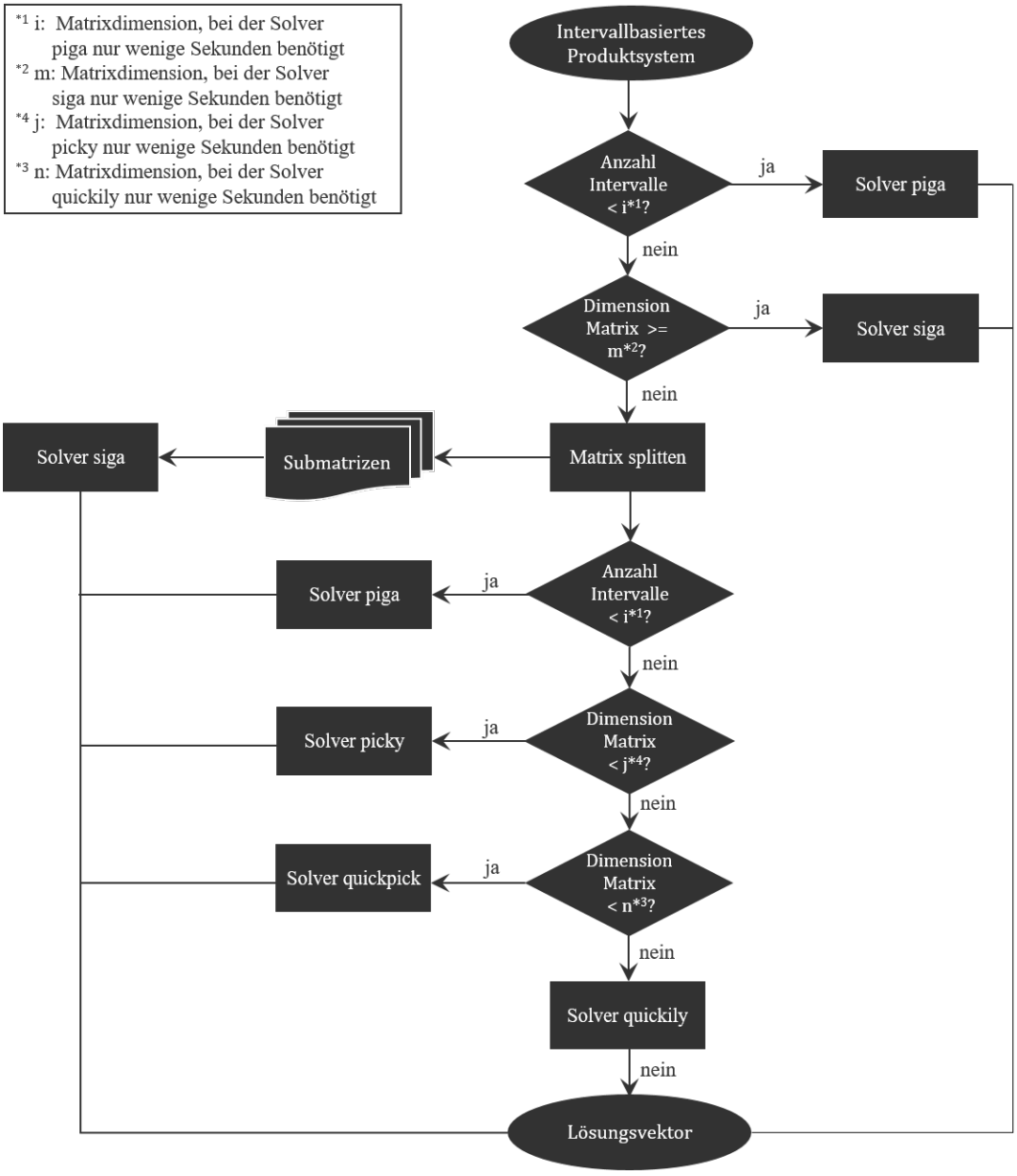


Abbildung 10.19: Allgemeines Verfahren zur Berechnung intervallbasierter Produktsysteme, in der Methode *goodSolution* implementiert.

Kapitel 11

Zusammenfassung und Ausblick

Im Folgenden werden die wesentlichen Aspekte dieser Arbeit zusammengefasst und die Erkenntnisse diskutiert. Zusätzlich wird ein Ausblick zur Weiterentwicklung und Modifikation des in dieser Arbeit entwickelten Verfahrens gegeben.

11.1 Zusammenfassung

In dieser Arbeit wurde die Hypothese aufgestellt, dass die Erweiterung der Ökobilanzierung um einen intervallarithmetischen Ansatz ein praktikables Verfahren zur ingenieursgerechten Berücksichtigung von Unsicherheiten darstellt. Im Zuge einer Ökobilanz sind eine Vielzahl von Informationen aus verschiedenen Bereichen der Wissenschaft notwendig, die oftmals auf unsicheren Daten basieren. Eine intervallbasierte Ökobilanzierung ermöglicht es, diese umfassend zu berücksichtigen. Dieser Ansatz wurde bereits vor vielen Jahren empfohlen, jedoch hat er sich bisher nicht in der Praxis durchgesetzt. Der Schwerpunkt dieser Arbeit bildete daher die Entwicklung und Implementierung eines intervallbasierten Berechnungsverfahrens zur ökologischen Bilanzierung. Dafür wurden Methoden aus den Bereichen der Intervallarithmetik, der objektorientierten Programmierung und der Graphentheorie angewandt.

11.1.1 Notwendigkeit einer intervallbasierten Ökobilanzierung zur Berücksichtigung von Unsicherheiten

Der intervallbasierte Ansatz zur Berücksichtigung von Unsicherheit reiht sich ein in den wissenschaftlichen Konsens von Experten, erkannte und nicht eliminierbare Unsicherheiten in der Modellierung zuzulassen. Unsicherheiten sollen in der Berechnung berücksichtigt und in Form vager Ergebnisse transparent dargestellt werden, anstatt diese im Nachgang der Kalkulation zu analysieren.

Ein intervallbasierter Ansatz in der Ökobilanzierung ist aus zwei wesentlichen Gründen dringend angeraten: Als erster Punkt ist die Heterogenität der Unsicherheit zu nennen. Sie wird in dieser Arbeit in Unpräzision, Ungenauigkeit und Ungewissheit differenziert, die sich aus den Dimensionen der Unsicherheit „Natur“, „Ort“ und „Größe der Unsicherheit“ ableiten lassen. Für die jeweils unterschiedlichen Arten von Unsicherheit existieren verschiedene Methoden zur Analyse der Unsicherheit. Zwar ist es mitunter möglich, diverse Verfahren zusammenzuführen; dies ist aber oftmals schwierig, insbesondere auch im Hinblick auf die Interpretation der daraus gewonnenen Ergebnisse. Die Erfassung des Möglichen und Relevanten als Intervall mit lediglich zwei Grenzwerten

kommt der geringen Informationsbasis entgegen; es ist der (kleinste) gemeinsame Nenner, der für alle Arten von Unsicherheit in Frage kommt.

Der zweite Aspekt ist, dass es durch den relativen Ansatz der Ökobilanzierung keine sichere Seite gibt. So wird der Einfluss von Elementarflüssen, welche die Technosphäre verlassen, stets *relativ* zu anderen Elementarflüssen bestimmt. Produktsysteme werden *relativ* zu möglichen Alternativen bewertet. Die Ergebnisse der Wirkungsabschätzung sind *Äquivalente* und keine absoluten Werte. Daher können unsichere Daten nicht mit Sicherheitsfaktoren belegt werden, um auf der sicheren Seite liegend die Unsicherheit zu berücksichtigen: eine Veränderung *eines* Kennwerts (z. B. der Charakterisierungsfaktor einer Emission) verändert nicht nur diesen, sondern auch den relativen Bezug und Beurteilung *aller anderen* Kennwerte, die mit diesem im Verhältnis stehen (z. B. durch Zuordnung zum gleichen Wirkungsindikator). Letztere weisen dann im Vergleich veränderte Einflüsse auf die Umwelt auf, ohne dass sich der Einfluss dieser *tatsächlich* verändert hat. Dies birgt die Gefahr der zu optimistischen oder zu pessimistischen Einschätzung von all denjenigen Systemen, welche diese Kennwerte im Zuge einer Ökobilanzierung anwenden. Das entwickelte Verfahren vermeidet Über- oder Unterschätzungen von Umweltwirkungen, indem es bei unsicheren Daten die gesamte Bandbreite des Möglichen und Relevanten einbezieht. Durch die Entwicklung des Unsicherheitsindexes *UnsI* können die Ergebnisintervalle hinsichtlich ihrer Unsicherheit einzeln, aber auch im Vergleich mit anderen Intervallen eingeordnet werden.

11.1.2 Graphentheoretische Betrachtungen

Die Produktsysteme stellen zusammenhängende, gerichtete Graphen dar. Sie sind Teilgraphen des übergeordneten Graphen zur Beschreibung der Ökobilanzierung, welcher weitere Knoten in Form von Wirkungsindikatoren enthält. Es konnten typische Systemstrukturen von Produktsystemen identifiziert werden, die in einfache und komplizierte Systemstrukturen differenziert werden können. Lineare und baumartige Produktsysteme stellen einfache Systemstrukturen dar. Die Analyse ihrer Produktmatrizen zeigte, dass sie M-Matrizen entsprechen. Gleichungssysteme mit Matrizen von diesem Typ können durch das Intervall-Gauß-Verfahren zuverlässig gelöst werden.

Komplizierte Systemstrukturen mit Hyperkanten liegen vor, wenn interne Koppelprodukte auftreten oder wenn ein Zwischenprodukt in diversen Prozessmodulen benötigt oder verarbeitet wird. Insbesondere durch die politischen Bestrebungen, die Ressourcen zu schonen und die industriellen Prozesse in eine Kreislaufwirtschaft zu überführen, nehmen Produktsysteme mit komplizierter Systemstruktur zu. Für Produktmatrizen solcher Produktsysteme sind beim sequentiellen Berechnungsansatz vermehrt iterative Berechnungsschritte notwendig. Diese können das zu große Aufweiten von Intervallen fördern. Das entwickelte Berechnungsverfahren beruht daher auf einem matrixbasierten Ansatz.

11.1.3 Entwicklung des intervallbasierten Berechnungsverfahrens

Die Lösung intervallarithmetischer Probleme birgt Herausforderungen. Hierbei ist insbesondere die Komplexität des Problems zu nennen - sie wird als hoch eingestuft. Durch die Verwendung von Näherungsverfahren tritt zudem das Problem abhängiger Intervalle auf. So werden Intervalle im Zuge der Berechnung durch mehrfaches Einsetzens zu sehr aufgeweitet, was die Aussagekraft der Resultate reduziert. Die vollständige Szenarioanalyse kann das intervallarithmetische Problem zwar approximationsfrei lösen, kommt jedoch nur bei sehr wenigen Intervallen aufgrund des erhöhten Rechenzeitbedarfs in Frage. Für spezielle Matrixtypen gibt es effizientere Methoden, mit denen die Hülle der Lösungsmenge berechnet werden kann. Näherungsverfahren, welche diese für allgemeine Matrizen ermitteln können, sind bisher jedoch nicht bekannt. Daher war es Ziel dieser Arbeit, ein Berechnungs-

verfahren zu entwickeln, mit dem unter Berücksichtigung der Praktikabilität möglichst enge Ergebnisintervalle kalkuliert werden. Das Problem abhängiger Intervalle tritt auf, wenn der Skalierungsvektor eines Produktsystems im Zuge einer Sachbilanz berechnet wird. Diesem Berechnungsschritt kam daher in dieser Arbeit eine zentrale Rolle zu.

Die Gleichungslöser wurden mit Testreihen an fiktiven Produktsystemen erprobt und mittels vollständiger Szenarioanalyse verifiziert. Die Testreihen basieren auf Produktsystemen linearer oder baumartiger Grundstruktur, wurden jedoch umfassend variiert, sodass eine Vielzahl von Hyper-Produktsystemen mit komplizierten Systemstrukturen entsteht. Bei der Analyse der Ergebnisse wurden die identifizierten Systemstrukturen von Produktsystemen sowie die Matrixtypen berücksichtigt. Es ließ sich bei Produktmatrizen von Produktsystemen mit komplizierten Systemstrukturen einzig die vollständige Szenarioanalyse identifizieren, mit der ausnahmslos die schmalsten Ergebnisintervalle berechnet werden können. Die vollständige Szenarioanalyse weist allerdings eine hohe Komplexität der Größenordnung $O(2^n)$ auf, wobei n die Anzahl der Intervalle mit Intervallweiten ungleich null darstellt. Daher ist dieses Verfahren nur für eine begrenzte Anzahl von Intervallen geeignet. Unter den übrigen Gleichungslösern konnte kein Solver festgestellt werden, mit dem sämtlich besonders schmale Intervalle kalkuliert werden können. Stattdessen stellt die Kombination mehrerer Gleichungslöser eine gute Alternative dar.

Der Unsicherheitsindex *UnsI* dient zur Einordnung der Größe von Unsicherheiten in intervallbasierten Werten. Er berechnet sich aus den Intervallgrenzen der zu bewertenden Größe und kann zwischen 0 % und 100 % betragen. Es wurden fünf Kategorien definiert: als „vergleichsweise sehr sicher“ werden *UnsI*-Werte bezeichnet, die geringer als 20 % sind. *UnsI*-Werte bis zu 40 % werden als „vergleichsweise sicher“ betrachtet. *UnsI*-Werte zwischen 40 % und 60 % sind dagegen „vergleichsweise mäßig unsicher“. Mit *UnsI*-Werten von 80 % sind die Ergebnisse als „vergleichsweise unsicher“ einzuordnen. Ab *UnsI*-Werten von 80 % sind die unsicheren Größen als „vergleichsweise sehr unsicher“ einzustufen, wobei eine vertiefte Recherche zu präziseren Daten dringend anzuraten ist. Ergebnisse mit UnsI-Werten von 0 % stellen sichere Ergebnisse dar, während Ergebnisse mit UnsI-Werten von 100 % völlige Ungewissheit repräsentieren.

11.1.4 Java-basierte Implementierung zur praktischen Anwendung

Die intervallarithmetischen Operationen und Matrix-Konstrukte wurden im Java-Paket *Ivari* programmiert. Es wurden sowohl Methoden implementiert, welche für spezielle Matrixtypen die Hülle der Lösungsmenge ermitteln können, als auch Varianten auf Basis des Intervall-Gauß-Verfahrens sowie auf Basis des Adjunktenverfahrens entworfen. Da das Adjunkten-Verfahren zur Determinantenberechnung auf den Laplaceschen Entwicklungssatz zurückgreift, gilt es mit seiner sehr hohen Komplexität als nicht praktikabel. Daher wurde ein Algorithmus entwickelt, mit dem der Rechenaufwand durch Splitten großer Matrizen in kleinere Submatrizen reduziert werden kann. Die Produktmatrizen werden dabei nach linearen und baumartigen Subsystemen durchsucht, deren Produktmatrizen mit dem Intervall-Gauß-Verfahren effizient berechnet werden können. Das Java-Paket *Ivari* kann in das Ökobilanzierungsprogramm *MultiValCA* integriert werden.

Auf Basis der Erkenntnisse wurden die Methoden *picky*, *quickpick* und *quickily* implementiert, die auf diverse Gleichungslöser zugreifen und das jeweils die schmalsten Ergebnisintervalle zurückgeben. Sie unterscheiden sich hinsichtlich ihres Rechenzeitbedarfs und der Qualität der Ergebnisse. Während die Methode *picky* sehr komplexe Solver aufruft und nur bis zu einer Matrixdimension von *7* geeignet ist, konnte die Methode *quickpick* Matrizen mit einer Dimension von *200* in ca. drei Minuten bei einer in dieser Arbeit verwendeten Rechnerleistung (Windows 10 Workstation mit Intel CPU E5-2630 (8 Kerne, 16 Threads, 2,4 GHz) sowie 32 GB RAM) lösen. Die Methode *quickily* ruft hingegen nur Solver mit einer geringeren Komplexität der Größenordnung $O(n^3)$ auf. Sie benötigt in etwa die gleiche Rechenzeit für eine Matrixdimension von *1000*. Neben dem Rechenzeitaufwand sinkt allerdings

auch die Qualität der Ergebnisse: während der Solver *pickily* bei den Testreihen maximal um ca. 10 % von der Hülle der Lösungsmenge abweicht, ist die maximale Abweichung hierfür beim Solver *quickpick* vergleichsweise vierfach und beim Solver *quickily* siebenfach erhöht.

11.1.5 Ableitung eines allgemeinen Verfahrens

Die Methoden *picky*, *quickpick* und *quickily* weisen unterschiedliche Schwerpunkte im Wechselspiel von Ergebnisqualität und Praktikabilität auf. Während die Methode *picky* die im Vergleich zeitaufwändigste Methode mit den vergleichsweise schmalsten Ergebnisintervallen darstellt, ist die Methode *quickily* die Methode mit der kürzesten Rechenzeit. Sie weist jedoch die Resultate mit den vergleichsweise größten *UnsI*-Werten auf. Die Methode *quickpick* stellt einen Mittelweg zwischen den beiden Methoden *picky* und *quickily* dar. Deswegen wurde ein allgemeingültiges Verfahren abgeleitet, wie die verschiedenen Methoden und Gleichungslöser im Zuge der intervallbasierten Berechnung zur Anwendung kommen können. Mit dem Verfahren werden möglichst schmale Ergebnisintervalle berechnet, wobei die dafür erforderliche Rechenzeit berücksichtigt wird. Zusätzlich ist es auf Gleichungssysteme mit beliebigen Produktmatrizen anwendbar.

Es wurde in der Methode *goodSolution* implementiert. Der Algorithmus verwendet als Auswahlkriterium die Dimension der Produktmatrix als auch die Anzahl der vorliegenden Intervalle mit Intervallweiten ungleich Null. Festgelegte Grenzen bestimmen, ab welcher Matrixdimenion bzw. ab welcher Intervallanzahl bestimmte Gleichungslöser ausgeschlossen werden. Zusätzlich sieht das Verfahren vor, Produktmatrizen linearer und baumartiger Subsysteme von der bestehenden Produktmatrix zu splitten und die Teilmatrizen jeweils gesondert zu berechnen. Dadurch wird die Matrixdimension der Produktmatrix reduziert. Die komprimierte Produktmatrix repräsentiert ein Produktsystem mit komplizierter Systemstruktur. Für Produktmatrizen solcher Produktsysteme ist die Berechnung der Hülle der Lösungsmenge häufig nur mit der vollständigen Szenarioanalyse möglich - diese kommt zur Anwendung, sofern die Intervallanzahl es zulässt. Ansonsten wird der Solver *picky* aufgerufen. Überschreitet die Matrixdimension hierfür den festgelegten Grenzwert, wird der Solver *quickpick* eingesetzt. Ist die Dimension der komprimierten Produktmatrix auch hierfür zu groß, wird stattdessen der Solver *quickily* angewandt.

Mit dem oben beschriebenen Verfahren ist der Grundstein geschaffen worden, um eine neue Ära der Ökobilanzierung anzustoßen, mit der die Vielzahl vorliegender Unsicherheiten ingenieursgerecht und transparent berücksichtigt werden kann.

11.2 Ausblick

Die oben dargestellte Erweiterung der Methodik erfordert die Integration intervallbasierter Daten in der Erfassung, Bearbeitung, Verwaltung und Auswertung von Ökobilanzen. Dies erfordert sowohl eine Anpassung der Denkweise und Arbeitsabläufe der Ökobilanzierenden als auch eine Überarbeitung der verfügbaren Datenbanken und Programme. Das Berechnungsverfahren wurde als Java-Paket *Ivari* implementiert und kann in das Java-Programm *MultiVaLCA* eingebunden werden, welches intervallbasierte Ökobilanzen unterstützt. Im Folgenden werden weitere Entwicklungsmöglichkeiten für diesen Bereich aufgezeigt.

11.2.1 Effizienz der implementierten Solver

Die implementierten Verfahren wurden hinsichtlich ihrer Funktionalität getestet und verifiziert. Die Funktionalität der Methode *adisplit* zum Splitten der Matrix als auch die implementierten Gleichungslöser wurden anhand

diverser Tests erfolgreich überprüft. Sie sollten dennoch durch zusätzliche Testreihen vertieft erprobt werden. Zusätzlich ist davon auszugehen, dass die Codequalität verbessert und die Effizienz der Algorithmen erhöht werden können. Eine Möglichkeit ist beispielsweise, dass das entwickelte Verfahren zum Splitten der Produktmatrix durch iterative Programmierprinzipien in seiner Effizienz verbessert werden kann. Mit effizienteren Algorithmen geht unter Umständen auch eine modifizierte Funktionalität des Verfahrens *goodSolution* einher: können bei gleicher Zeit weitere Solver aufgerufen werden, kann dies zu schmaleren Ergebnisintervallen führen.

11.2.2 Praktische Erprobung des Verfahrens

Das vorgestellte intervallbasierte Berechnungsverfahren erweitert die etablierte Methodik zur Ökobilanzierung. Damit das neue Verfahren in der Praxis angewandt werden kann, müssen die Datensätze unter Berücksichtigung existenter Unsicherheiten neu erfasst werden. Die Verwendung von bestehenden Sachbilanzen und Wirkungsabschätzungen wird nicht empfohlen, da diese auf vermeintlich exakt erfassten Daten und Methoden beruhen, deren Unsicherheiten nicht umfassend abgebildet werden. Mit dem intervallbasierten Ansatz geht auch einher, dass die verfügbaren Datenbanken und Programme derart angepasst werden, dass intervallbasierte Daten verarbeitet werden können. Auf Basis *intervallbasierter* Datensätze ist es möglich, die Kategorien des entwickelten Unsicherheitsindexes *UnsI* zu kalibrieren. Bisher werden *UnsI*-Werte fünf gleichmäßig verteilten Gruppen zugeordnet. Durch einen umfangreiche Basis an *UnsI*-Werten realer Datensätze kann diese Einteilung analysiert und gegebenenfalls angepasst werden.

Für das Verfahren *goodSolution* sind Grenzwerte erforderlich, die über die Wahl des Gleichungslösers entscheiden. Es werden in dieser Arbeit zwar Vorschläge zu konkreten Grenzwerten unterbreitet, allerdings hängen diese von der Rechnerleistung und der Geduld des Ökobilanzierenden ab. Anders als bei aufwändigen Simulationen ist es im Zuge von Ökobilanzen ungewöhnlich, die Resultate nach dem Berechnungsstart nicht innerhalb kurzer Zeit zu erhalten. Die Vorschläge zu den Grenzwerten basieren daher auf Rechenzeiten von wenigen Minuten unter Annahme einer durchschnittlichen Rechnerleistung, sollten aber auf Akzeptanz und Praktikabilität erprobt werden.

11.2.3 Bewertung über Rangordnung

Auf Basis von Ökobilanzen können Produktsysteme mit derselben funktionellen Einheit verglichen werden, wobei die zugrunde gelegten Prozessketten und die abgebildeten Produkte sich unterscheiden können. Gleichsam ist es möglich, Produktsysteme zu vergleichen, welche ähnliche Prozessverfahren abbilden, wobei diese jedoch z. B. hinsichtlich ihrer Maschineneffizienz oder ihrer Lieferanten voneinander abweichen. Dabei ist es auch üblich, jeweils äquivalente Lebenswegabschnitte oder einzelne Prozessmodule gesondert miteinander in Relation zu setzen und daraus eine Rangordnung abzuleiten.

Im Zuge einer intervallbasierten Ökobilanzierung ist es ebenfalls möglich, die jeweiligen Lebenswegabschnitte diverser Produktsysteme bzw. verschiedene Varianten von (aggregierten) Prozessmodulen eines Produktsystems gesondert miteinander zu vergleichen und diese mithilfe eines Rankings zu sortieren. Lässt sich ein Produktsystem identifizieren, bei dem alle Intervalle der betrachteten Lebenswegabschnitte bzw. Prozessmodule kleiner oder gleich sind im Vergleich zu den Intervallen der äquivalenten Lebenswegabschnitte bzw. Prozessmodule der übrigen Produktsysteme, kann dieses Produktsystem in Relation zu den anderen Produktsystemen als dominant und damit im Vergleich als „besser geeignet“ eingestuft werden.

In Abbildung 11.1 sind exemplarisch im linken Diagramm die fiktiven Ergebnisintervalle dreier Produktsysteme für die Informationsmodule *A*, *B*, *C* und *D* abgebildet. Da die Ergebnisintervalle des Produktsystems *1* allesamt

kleiner oder gleich sind in Relation zu den entsprechenden Ergebnisintervallen der Produktsysteme *2* und *3*, wird dem Produktsystem *1* der 1. Rang zugewiesen; dieses Produktsystem weist somit die „vergleichsweise geringsten“ Umweltwirkungen auf. Im rechten Diagramm der Abbildung 11.1 sind dagegen die Ergebnisintervalle diverser Prozessmodule von einem Produktsystem abgebildet, wobei die Prozesse des abgebildeten Verfahrens modifiziert werden.

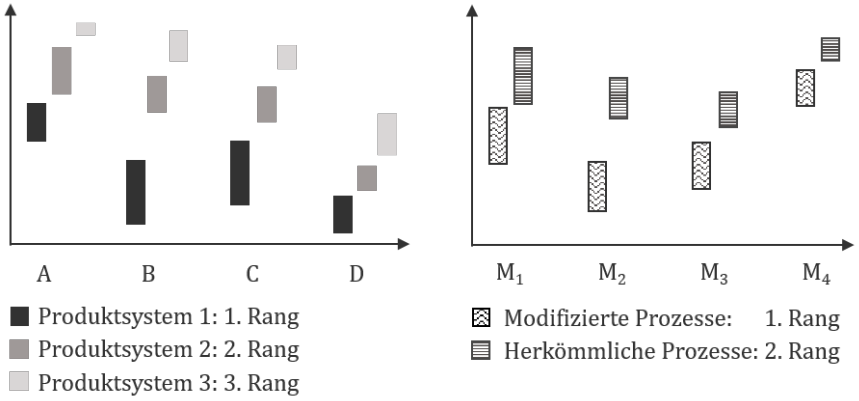


Abbildung 11.1: Bewertung vergleichbarer Produktsysteme oder Prozessmodule über Rangordnung.

Da die Ergebnisintervalle der modifizierten Prozesse allesamt unterhalb der Ergebnisintervalle der herkömmlichen Prozesse liegen, können die modifizierten Prozesse als „vergleichsweise besser geeignet“ beurteilt werden. Ein Verfahren, welches derlei Rangordnungen und Vergleiche automatisiert durchführt, kann die Bewertung intervallbasierter Ökobilanzen wesentlich erleichtern und sollte daher ebenfalls implementiert werden.

Literaturverzeichnis

[1] ARENS, Tilo ; HETTLICH, Frank ; KOCKELKORN, Ulrich ; LICHTENEGGER, Klaus ; STACHEL, Hellmuth ; KARPFINGER, Christian: *Mathematik*. 4. Auflage. Berlin, Heidelberg : Springer Berlin Heidelberg, 2018. http://dx.doi.org/10.1007/978-3-662-56741-8.

[2] HOEVER, Georg: *Höhere Mathematik kompakt: Mit Erklärvideos und interaktiven Visualisierungen*. 3. Auflage. Berlin : Springer Spektrum, 2020 https://doi.org/10.1007/978-3-662-62080-9.

[3] KARPFINGER, Christian ; STACHEL, Hellmuth: *Lineare Algebra*. 1. Auflage. Berlin : Springer Spektrum, 2020 https://doi.org/10.1007/978-3-662-61340-5.

[4] STRAMPP, Walter ; JANSSEN, Dörthe: *Höhere Mathematik 1: lineare Algebra*. 4. Auflage. Berlin : Springer Vieweg, 2020 https://doi.org/10.1007/978-3-662-61023-7.

[5] BOSCH, Siegfried: *Lineare Algebra*. 5. Auflage. Berlin, Heidelberg : Springer Spektrum, 2014 https://doi.org/10.1007/978-3-642-55260-1.

[6] BRU, R. ; CORRAL, C. ; GIMENEZ, I. ; MAS, J.: Classes of general H-matrices. In: *Linear Algebra and its Applications* 429 (2008), Nr. 10, S. 2358–2366. http://dx.doi.org/10.1016/j.laa.2007.10.030.

[7] FIEDLER, Miroslav ; PTÁK, Vlastimil: On matrices with non-positive off-diagonal elements and positive principal minors. In: *Czechoslovak Mathematical Journal* 12 (1962), Nr. 3, S. 382–400. http://dx.doi.org/10.21136/cmj.1962.100526.

[8] FIEDLER, Miroslav ; PTÁK, Vlastimil: Some generalizations of positive definiteness and monotonicity. In: *Numerische Mathematik* 9 (1966), Nr. 2, S. 163–172. http://dx.doi.org/10.1007/bf02166034.

[9] PLEMMONS, R. J.: M-matrix characterizations.I—nonsingular M-matrices. In: *Linear Algebra and its Applications* 18 (1977), Nr. 2, S. 175–188. https://doi.org/10.1016/0024-3795(77)90073-8.

[10] TÖRNIG, W.: *Numerische Mathematik für Ingenieure und Physiker*. Bd. Band 1: *Numerische Mathematik für Ingenieure und Physiker: Numerische Methoden der Algebra*. Berlin Heidelberg New York : Springer, 1979 https://doi.org/10.1007/978-3-642-96508-1.

[11] POLMAN, Ben: Incomplete blockwise factorizations of (block) H-matrices. In: *Linear Algebra and its Applications* 90 (1987), S. 119–132. http://dx.doi.org/10.1016/0024-3795(87)90310-7.

[12] KLUWE, Matthias: *Ein Beweis des Multiplikativen Ergodensatzes*. Bremen, Universität Bremen, Diplomarbeit, 2002. https://math.stugen.de/downloads/diplome/mes.pdf.

[13] BURBACH, H.: *Algorithmen zur parallelen Determinantenberechnung.* Dortmund, Universität Dortmund, Diplomarbeit, 1992. https://burbach.eu/diplom.pdf.

[14] ENGELN-MÜLLGES, Gisela ; NIEDERDRENK, Klaus ; WODICKA, Reinhard: *Numerik-Algorithmen: Verfahren, Beispiele, Anwendungen.* 9. Auflage. Berlin : Springer, 2005 http://lib.myilibrary.com/detail.asp?id=62766.

[15] BÄRWOLFF, Günter: *Numerik für Ingenieure, Physiker und Informatiker.* 3. Auflage. Berlin : Springer Spektrum, 2020 https://doi.org/10.1007/978-3-662-61734-2.

[16] KANZOW, Christian: *Numerik linearer Gleichungssysteme: Direkte und iterative Verfahren.* Berlin Heidelberg New York : Springer, 2005. http://dx.doi.org/10.1007/b138019.

[17] MEISTER, Andreas: *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren. Mit MATLAB®-Implementierungen von C. Vömel.* 5. Auflage. Wiesbaden : Springer Spektrum, 2015 https://doi.org/10.1007/978-3-658-07200-1.

[18] WITT, Kurt-Ulrich ; MÜLLER, Martin E.: *Algorithmische Informationstheorie: Berechenbarkeit und Komplexität verstehen.* 1. Auflage. Berlin Heidelberg : Springer Spektrum, 2020 https://doi.org/10.1007/978-3-662-61694-9.

[19] ERNST, HARTMUT, SCHMIDT, JOCHEN ; BENEKEN, Gerd: *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis - eine umfassende, praxisorientierte Einführung.* 7. Auflage. Wiesbaden : Springer Vieweg, 2020. http://dx.doi.org/10.1007/978-3-658-30331-0.

[20] SILBERBAUER, Christian: *Einstieg in Java und OOP: Grundelemente, Objektorientierung, Design-Patterns und Aspektorientierung.* 2. Auflage. Berlin : Springer Vieweg, 2020 https://doi.org/10.1007/978-3-662-61309-2.

[21] GÜTING, Ralf H. ; DIEKER, Stefan: *Datenstrukturen und Algorithmen.* 4. Auflage. Wiesbaden : Springer Vieweg, 2018. http://dx.doi.org/10.1007/978-3-658-04676-7.

[22] SAAKE, Gunter ; SATTLER, Kai-Uwe: *Algorithmen und Datenstrukturen: Eine Einführung mit Java.* 6. Auflage. Heidelberg : dpunkt.verlag, 2021 https://dpunkt.de/produkt/algorithmen-und-datenstrukturen.

[23] KNEBL, Helmut: *Algorithmen und Datenstrukturen: Grundlagen und probabilistische Methoden für den Entwurf und die Analyse.* Wiesbaden : Springer Vieweg, 2019 https://doi.org/10.1007/978-3-658-26512-0.

[24] BARTELS, Sören: *Numerik 3x9: Drei Themengebiete in jeweils neun kurzen Kapiteln.* Berlin Heidelberg : Springer Spektrum, 2016 http://dx.doi.org/10.1007/978-3-662-48203-2.

[25] ERK, Katrin; PRIESE, Lutz: *Theoretische Informatik: Eine umfassende Einführung.* 3. Auflage. Berlin Heidelberg : Springer, 2008 https://doi.org/10.1007/978-3-540-76320-8.

[26] LOGOFĂTU, Dorina: *Grundlegende Algorithmen mit Java: Lern- und Arbeitsbuch für Informatiker und Mathematiker.* 2. Auflage. Wiesbaden : Springer Vieweg, 2014 https://doi.org/10.1007/978-3-8348-2355-7.

[27] ULLENBOOM, Christian: *Java ist auch eine Insel: Einführung, Ausbildung, Praxis*. https://openbook.rheinwerk-verlag.de/javainsel/. Version: 15. Auflage, 2020.

[28] HICKLIN, J. ; MOLER, C. ; WEBB, P. ; BOISVERT, R. F. ; MILLER, B. ; POZO, R. ; REMINGTON, K.: *JAMA: A Java Matrix Package*. Java package. https://math.nist.gov/javanumerics/jama/. Version: 2012.

[29] HICKEY, Timothy J.: *IAMath.java*. Java package. www.cs.brandeis.edu/~tim/Applets/ia/ia/ia_math/IAMath.java. Version: 1997.

[30] NEUBERT, Stefan: *Grundkurs Theoretische Informatik*. 1. Auflage. Bonn : Rheinwerk Verlag, 2021.

[31] ZIMMER, M.: *Software zur hocheffizienten Loesung von Intervallgleichungssystemen mit C-XSC*. Wuppertal, Bergische Universität Wuppertal, Dissertation, 2013. http://elpub.bib.uni-wuppertal.de/servlets/DerivateServlet/Derivate-3490/dc1302.pdf.

[32] GERDTS, Matthias: *Einführung in die Numerik: Universität der Bundeswehr München Wintertrimester 2014*. Neubiberg, Universität der Bundeswehr München, Skript, 2014.

[33] DEUFLHARD, Peter ; HOHMANN, Andreas: *Eine algorithmisch orientierte Einführung: Teil des mehrbändigen Werks Numerische Mathematik*. Bd. Band 1. 4. Auflage. Berlin New York : Walter de Gruyter, 2008. http://dx.doi.org/10.1515/9783110203554.

[34] SEDGEWICK, Robert ; WAYNE, Kevin: *GaussianElimination.java*. https://introcs.cs.princeton.edu/java/95linear/GaussianElimination.java.html. Version: 20.10.2017.

[35] SCHREYER, Jens: *Vorlesung Mathematik für Informatiker*. Ulmenau, Technische Universität Ulmenau, Skript, 2019. https://wikiin.de/_media/fach/mathematik-fur-informatiker-1/ws1819/mfi_ws1819.pdf.

[36] CHEN, Richard: *Github: algorithms/Matrix.java*. https://github.com/rchen8/Algorithms/blob/master/Matrix.java. Version: 28.10.2020.

[37] POTTMEYER, Lukas: *Diskrete Mathematik: Ein kompakter Einstieg*. Berlin : Springer Spektrum, 2019 https://doi.org/10.1007/978-3-662-59663-0.

[38] AIGNER, Martin: *Diskrete Mathematik: Mit 600 Übungsaufgaben*. 6. Auflage. Wiesbaden : Vieweg, 2006. http://dx.doi.org/10.1007/978-3-8348-9039-9.

[39] DIETMAIER, Christopher: *Mathematik für angewandte Wissenschaften*. Berlin, Heidelberg : Springer, 2014. http://dx.doi.org/10.1007/978-3-8274-2421-1.

[40] CRAMER, Erhard ; KAMPS, Udo: *Grundlagen der Wahrscheinlichkeitsrechnung und Statistik: Eine Einführung für Studierende der Informatik, der Ingenieur- und Wirtschaftswissenschaften*. 5. Auflage. Berlin, Heidelberg : Springer, 2020 https://doi.org/10.1007/978-3-662-60552-3.

[41] MÜLLER, Christine ; DENECKE, Liesa: *Stochastik in den Ingenieurwissenschaften: Eine Einführung mit R*. Berlin, Heidelberg : Springer Vieweg, 2013 (Statistik und ihre Anwendungen). https://doi.org/10.1007/978-3-642-38960-3.

[42] ADMIN: Iterative approach to find permutations of a string in C++, Java and Python. In: *Techie Delight* (8.12.2016). https://www.techiedelight.com/find-permutations-string-cpp-java-iterative/.

[43] NESTMANN, Uwe: *Theoretische Grundlagen der Informatik I (TheGI1): Grundlagen und Algebraische Stukturen*. Berlin, Technische Universität Berlin, Formelsammlung WiSe 2012/13:v11, 2013.

[44] BEECK, H.: *Über intervallarithmetische Methoden bei linearen Gleichungssystemen mit Inervallkoeffizienten und Zusammenhänge mit der Fehleranalysis*. München, Universität München, Dissertation, 1972.

[45] ALEFELD, Götz ; HERZBERGER, Jürgen: *Reihe Informatik*. Bd. 12: *Einführung in die Intervallrechnung*. Mannheim : Bibliografisches Institut Wissenschaftsverlag, 1974.

[46] HANSEN, Eldon R. ; WALSTER, G. W.: *Monographs and textbooks in pure and applied mathematics*. Bd. 264: *Global optimization using interval analysis*. 2. Edition. New York : Dekker, 2004.

[47] KULISCH, Ulrich: *De Gruyter Studies in Mathematics*. Bd. 33: *Computer arithmetic and validity: Theory, implementation, and applications*. 2. Edition. Berlin and Boston : De Gruyter, 2013.

[48] NEUMAIER, A.: *Encyclopedia of mathematics and its applications*. Bd. volume 37: *Interval methods for systems of equations*. Cambridge : Cambridge University Press, 1990. http://dx.doi.org/10.1017/CBO9780511526473.

[49] MOORE, Ramon E.: *Intervallanalyse: Mit 7 Abbildungen*. Autorisierte Übersetzung der englischsprachigen Originalausgabe. München and Wien : R. Oldenbourg Verlag, 1969.

[50] SCHAEFER, Uwe: *Das lineare Komplementaritätsproblem mit Intervalleinträgen*. Karlsruhe, Universität Karlsruhe, Dissertation, 1999. http://dx.doi.org/10.5445/IR/87699.

[51] APOSTOLATOS, N. ; KULISCH, U.: Grundlagen einer Maschinenintervallarithmetik. In: *Computing* 2 (1967), Nr. 2, S. 89–104. http://dx.doi.org/10.1007/BF02239180.

[52] HANSEN, E. R.: Bounding the Solution of Interval Linear Equations. In: *SIAM Journal on Numerical Analysis* 29 (1992), Nr. 5, S. 1493–1503. http://www.jstor.org/stable/2158054.

[53] HLADÍK, Milan: Stability of the linear complementarity problem properties under interval uncertainty. In: *Central European Journal of Operations Research* 29 (2021), Nr. 3, S. 875–889. http://dx.doi.org/10.1007/s10100-021-00745-6.

[54] HLADÍK, Milan: On Relation Between P-Matrices and Regularity of Interval Matrices. Version: 2017. https://doi.org/10.1007/978-3-319-49984-0_2. In: BEBIANO, Natália (Hrsg.): *Applied and Computational Matrix Analysis: MAT-TRIAD, Coimbra, Portugal, September 2015 Selected, Revised Contributions* Bd. 192. Cham : Springer, 2017, S. 27–35.

[55] BEECK, H.: Zur Problematik der Hüllenbestimmung von Intervallgleichungssystemen. Version: 1975. http://dx.doi.org/10.1007/3-540-07170-9_12. In: *Interval Mathematics*. Berlin : Springer Verlag, 1975 (Lecture notes in Computer Science), S. 150–159.

[56] NING, S. ; KEARFOTT, R. B.: A Comparison of some Methods for Solving Linear Interval Equations. In: *SIAM Journal on Numerical Analysis* 34 (1997), Nr. 4, S. 1289–1305. http://dx.doi.org/10.1137/S0036142994270995.

[57] ROHN, J.: Inverse Interval Matrix. In: *SIAM Journal on Numerical Analysis* 30 (1993), Nr. 3, S. 864–870. http://dx.doi.org/10.1137/0730044.

[58] *Lexikon der Mathematik - Intervall-Gleichungssystem.* https://www.spektrum.de/lexikon/mathematik/intervall-gleichungssystem/4935. Version: 14.11.2019

[59] HEINDL, Gerhard ; KREINOVICH, Vladik ; LAKEYEV, Anatoly V.: Solving Linear Interval Systems Is NP-Hard Even If We Exclude Overflow and Underflow. In: *Reliable Computing* (1998), Nr. 4, S. 383–388. http://www.nsc.ru/interval/lakeyev/publications/98rc.pdf

[60] KREINOVICH, V. ; LAKEEV, A. V. ; NOSKOV, S. I.: Optimal solution of interval linear systems is intractable (NP-hard). In: *Interval Computations* (1993), S. 6–14. https://www.cs.utep.edu/vladik/1993/tr93-16a.pdf.

[61] BEECK, H.: Über Struktur und Abschätzungen der Lösungsmenge von linearen Gleichungssystemen mit Intervallkoeffizienten. In: *Computing* 10 (1972), Nr. 3, S. 231–244. http://dx.doi.org/10.1007/bf02316910.

[62] GARLOFF, Jürgen: Interval Gaussian Elimination with Pivot Tightening. In: *SIAM Journal on Matrix Analysis and Applications* 30 (2009), Nr. 4, S. 1761–1772. http://dx.doi.org/10.1137/080729621.

[63] NIRMALA, T. ; DATTA, D. ; KUSHWAHA, H. S. ; GANESAN, K.: Inverse Interval Matrix: A New Approach. In: *Applied Mathematical Sciences* 5 (2011), Nr. 13, S. 607–624. http://www.m-hikari.com/ams/ams-2011/ams-13-16-2011/ganesanAMS13-16-2011.pdf.

[64] HANSEN, Eldon R.: The Hull of Preconditioned Interval Linear Equations. In: *Reliable Computing* 6 (2000), Nr. 2, S. 95–103. http://dx.doi.org/10.1023/A:1009962903365.

[65] SCHICHL, Hermann ; DOMES, Ferenc ; MONTANHER, Tiago ; KOFLER, Kevin: Interval unions. In: *BIT Numerical Mathematics* 57 (2017), Nr. 2, S. 531–556. http://dx.doi.org/10.1007/s10543-016-0632-y.

[66] FROMMER, A. ; MAYER, G.: A New Criterion to Guarantee the Feasibility of the Interval Gaussian Algorithm. In: *SIAM Journal on Matrix Analysis and Applications* 14 (1993), Nr. 2, S. 408–419. http://dx.doi.org/10.1137/0614029.

[67] SPEKTRUM.DE: *Lexikon der Mathematik: Inneneinschließung.* https://www.spektrum.de/lexikon/mathematik/inneneinschliessung/4191. Version: 14.11.2019.

[68] ROHN, Jiri: Cheap and Tight Bounds: The Recent Result by E. Hansen Can Be Made More Efficient. In: *Interval Computations* 4 (1993), S. 13–21 http://uivtx.cs.cas.cz/~rohn/publist/69.pdf.

[69] BOSSEL, Hartmut: *Systeme, Dynamik, Simulation: Modellbildung, Analyse und Simulation komplexer Systeme.* Norderstedt : Books on Demand, 2004.

[70] GÜNTHER, Marco ; VELTEN, Kai: *Mathematische Modellbildung und Simulation: Eine Einführung für Wissenschaftler, Ingenieure und Ökonomen.* John Wiley & Sons, 2014.

[71] BUNGARTZ, Hans-Joachim ; ZIMMER, Stefan ; BUCHHOLZ, Martin ; PFLÜGER, Dirk: *Modellbildung und Simulation: Eine anwendungsorientierte Einführung.* 2. Auflage. Berlin and Heidelberg : Springer Spektrum, 2013 (eXamen.press). http://dx.doi.org/10.1007/978-3-642-37656-6.

[72] BRANDSTÄTTER, Markus F.: *Kompatibilitätsmodellierung im Systems-Engineering Umfeld*. München, Universität München, Dissertation, 2009

[73] METZLER, Wolfgang: *Dynamische Systeme in der Ökologie: Mathematische Modelle und Simulation*. Wiesbaden : Vieweg+Teubner Verlag, 1987 (Teubner Studienbücher Mathematik). http://dx.doi.org/10.1007/978-3-322-93109-2.

[74] FISCHER-STABEL, Peter (Hrsg.): *Umweltinformationssysteme*. Heidelberg : Wichmann, 2005.

[75] KALTSCHMITT, Martin ; SCHEBEK, Liselotte: *Umweltbewertung für Ingenieure: Methoden und Verfahren*. Berlin Heidelberg : Springer Vieweg, 2015. http://dx.doi.org/10.1007/978-3-642-36989-6.

[76] WAGNER, Reinhard: *Vermittlung systemwissenschaftlicher Grundkonzepte*. Graz, Universität Graz, Diplomarbeit, 2002.

[77] SCHAEFER, Matthias: *Wörterbuch der Ökologie*. 5. Auflage. Heidelberg : Spektrum Akademischer Verlag, 2012. http://dx.doi.org/10.1007/978-3-8274-2562-1.

[78] HADAC, Emil: Mensch und Landschaft. In: *N. F. Hercynia* Bd. 13 (1976), Nr. 2, S. 214–215. https://www.zobodat.at/pdf/Hercynia_13_0214-0215.pdf.

[79] SIMON, Karl-Heinz: Systemtheoretische Modelle in der sozial-ökologischen Forschung. Version: 2013. https://d-nb.info/1197913742/34#page=27. In: *Region als System - Theorien und Ansätze für die Regionalentwicklung* Bd. 1. 2013, S. 13–23.

[80] BRENNICKE, Axel ; SCHOPFER, Peter: Ökologische Kreisläufe der Stoffe und der Strom der Energie. Version: 2010. http://dx.doi.org/10.1007/978-3-8274-2352-8_15. In: SCHOPFER, Peter (Hrsg.) ; BRENNICKE, Axel (Hrsg.) ; MOHR, Hans (Hrsg.): *Pflanzenphysiologie*. Heidelberg : Spektrum Akad. Verl., 2010, S. 347–353.

[81] BUSELMAIER, Werner: Grundzüge der Ökologie. Version: 2012. https://doi.org/10.1007/978-3-642-27175-5_22. In: *Biologie für Mediziner*. Berlin Heidelberg : Springer, 2012, S. 297–308.

[82] FRITSCHE, Olaf: *Biologie für Einsteiger*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2015. http://dx.doi.org/10.1007/978-3-662-46278-2.

[83] BAUER, Joa: *Industrielle Ökologie : theoretische Annäherung an ein Konzept nachhaltiger Produktionsweisen*. Stuttgart, Universität Stuttgart, Dissertation, 2008. http://dx.doi.org/10.18419/opus-5498.

[84] HARARI, Yuval N.: *Homo Deus: Eine Geschichte von Morgen*. 13. Auflage. München : C.H. Beck, 2018.

[85] BIESER, Jan ; HINTEMANN, Ralph ; BEUCKER, Severin ; SCHRAMM, Stefanie ; HILTY, Lorenz ; ET AL ; KÜHN, Melissa: *Klimaschutz durch digitale Technologien–Chancen und Risiken*. https://www.zora.uzh.ch/id/eprint/190091/1/2020-05_bitkom_klimastudie_digitalisierung%282%29.pdf.

[86] TITTMANN, Peter: *Graphentheorie: Eine anwendungsorientierte Einführung*. 3. Auflage. München, 2019 https://www.hanser-elibrary.com/doi/book/10.3139/9783446465039.

[87] KADERALI, Firoz ; POGUNTKE, Werner: *Graphen, Algorithmen und Netze: Kurseinheit 1: Grundbegriffe*. Hagen, Fernuniversität Hagen, Skript, 2005.

[88] FÁRY, István: On straight-line representation of planar graphs. In: *Acta Sci. Math. Szeged* 11 (1948), S. 229–233. http://acta.bibl.u-szeged.hu/13574/1/math_011_fasc_004_229-233.pdf.

[89] KRUMKE, Sven O.: *Graphentheoretische Konzepte und Algorithmen*. 3. Aufl. 2012. Wiesbaden : Vieweg+Teubner Verlag, 2012 (SpringerLink Bücher). http://dx.doi.org/10.1007/978-3-8348-2264-2.

[90] GALLO, Giorgio ; LONGO, Giustino ; PALLOTTINO, Stefano ; NGUYEN, Sang: Directed hypergraphs and applications. In: *Discrete Applied Mathematics* 42 (1993), Nr. 2-3, S. 177–201. http://dx.doi.org/10.1016/0166-218X(93)90045-P.

[91] NAGAMOCHI, Hiroshi ; IBARAKI, Toshihide: Computing Edge-Connectivity in Multigraphs and Capacitated Graphs. In: *SIAM Journal on Discrete Mathematics* 5 (1992), Nr. 1, S. 54–66. http://dx.doi.org/10.1137/0405004.

[92] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *Umweltmanagement - Ökobilanz - Grundsätze und Rahmenbedingungen (ISO 14040:2006 + Amd 1:2020): Deutsche Fassung EN ISO 14040:2006 + A1:2020*. Berlin, Februar 2021.

[93] KLÖPFFER, Walter ; GRAHL, Birgit: *Ökobilanz (LCA): Ein Leitfaden für Ausbildung und Beruf*. Weinheim : WILEY-VCH, 2009. http://dx.doi.org/10.1002/9783527627158.

[94] BADER, Hans-Peter ; BACCINI, Peter: System-Modelle und Simulations-Programme im Umweltmanagement – Eine kritische Analyse zum Stand der Technik. In: *GAIA - Ecological Perspectives for Science and Society* 5 (1996), Nr. 6, S. 263–275. http://dx.doi.org/10.14512/gaia.5.6.3.

[95] RANDALL, D. A. ; R.A. WOOD ; S. BONY ; R. COLMAN ; T. FICHEFET ; J. FYFE ; V. KATTSOV ; A. PITMAN ; J. SHUKLA ; J. SRINIVASAN ; R.J. STOUFFER ; A. SUMI: Climate Models and Their Evaluation. Version: 2007. https://www.ipcc.ch/report/ar4/wg1/climate-models-and-their-evaluation/ In: IPCC, 2007 (Hrsg.): *Climate Change 2007: The Physical Science Basis*. Cambridge, United Kingdom and New York : Cambridge University Press, 2007, S. 590–662.

[96] FRISCHKNECHT, Rolf: *Lehrbuch der Ökobilanzierung*. Berlin : Springer Spektrum, 2020 https://doi.org/10.1007/978-3-662-54763-2.

[97] HEIJUNGS, Reinout ; SUH, Sangwon: *Eco-Efficiency in Industry and Science*. Bd. 11: *The Computational Structure of Life Cycle Assessment*. Dordrecht : Springer, 2002. http://dx.doi.org/10.1007/978-94-015-9900-9.

[98] EUROPEAN COMMISSION. JOINT RESEARCH CENTRE. INSTITUTE FOR ENVIRONMENT AND SUSTAINABILITY.: *International Reference Life Cycle Data System (ILCD) Handbook :general guide for life cycle assessment : detailed guidance: Detailed guidance*. Publications Office, 2010. http://dx.doi.org/10.2788/38479.

[99] EUROPEAN COMMISSION. JOINT RESEARCH CENTRE. INSTITUTE FOR ENVIRONMENT AND SUSTAINABILITY.: *International Reference Life Cycle Data System (ILCD) Handbook :specific guide for Life Cycle Inventory (LCI) data sets*. Publications Office, 2010. http://dx.doi.org/10.2788/39726.

[100] EUROPEAN COMMISSION. JOINT RESEARCH CENTRE. INSTITUTE FOR ENVIRONMENT AND SUSTAINABILITY: *International Reference Life Cycle Data System (ILCD) handbook : framework and requirements for life cycle*

impact assessment models and indicators. Publications Office, 2010. http://dx.doi.org/10.2788/38719.

[101] BRUNDTLAND, Gro H.: *Report of the World Commission on Environment and Development: Our Common Future: Note by the Secretary-General*, 1987.

[102] PURVIS, Ben ; MAO, Yong ; ROBINSON, Darren: Three pillars of sustainability: in search of conceptual origins. In: *Sustainability Science* 14 (2019), Nr. 3, S. 681–695. http://dx.doi.org/10.1007/s11625-018-0627-5.

[103] PUFÉ, Iris: *utb-studi-e-book*. Bd. 3667: *Nachhaltigkeit*. 2. überarb. u. erw. Aufl. Konstanz and Stuttgart : UVK-Verl.-Ges and UTB, 2014 http://www.utb-studi-e-book.de/9783838540542.

[104] RENN, Ortwin ; DEUSCHLE, Jürgen ; JÄGER, Alexander ; WEIMER-JEHLE, Wolfgang: *Leitbild Nachhaltigkeit: Eine normativ-funktionale Konzeption und ihre Umsetzung*. Wiesbaden : VS Verlag für Sozialwissenschaften | GWV Fachverlage GmbH Wiesbaden, 2007. http://dx.doi.org/10.1007/978-3-531-90495-5.

[105] HAUFF, Michael von: *Nachhaltige Entwicklung: Grundlagen und Umsetzung*. 2., aktualisierte Auflage. München : De Gruyter Oldenbourg, 2014. http://dx.doi.org/10.1524/9783486856002.

[106] GRUNWALD, Armin: *Nachhaltigkeit verstehen: Arbeiten an der Bedeutung nachhaltiger Entwicklung*. München : oekom verlag, 2016 http://www.content-select.com/index.php?id=bib_view&ean=9783960061472.

[107] BÜRGER, Veit ; HESSE, Tilman ; QUACK, Dietlinde ; PALZER, Andreas ; KÖHLER, Benjamin ; HERKEL, Sebastian ; ENGELMANN, Peter: Klimaneutraler Gebäudebestand 2050. (2016). https://www.umweltbundesamt.de/sites/default/files/medien/378/publikationen/climate_change_06_2016_klimaneutraler_gebaeudebestand_2050.pdf.

[108] CLIMATEPARTNER GMBH: *Aufforstung in Nicaragua | ClimatePartner*. https://www.climatepartner.com/de/klimaschutzprojekte/aufforstung-nicaragua. Version: 2021.

[109] CLIMATEPARTNER GMBH: *CO2-Ausgleich für Unternehmen | ClimatePartner*. https://www.climatepartner.com/de/leistungen/co2-ausgleich. Version: 10.12.2021.

[110] CLIMATEPARTNER GMBH: *Corporate Carbon Footprint: Die CO2-Bilanz Ihres Unternehmens*. https://www.climatepartner.com/de/leistungen/ccf-corporate-carbon-footprint. Version: 24.09.2021.

[111] NATUREOFFICE GMBH: *CO2-Footprints (CCF + PCF) - natureoffice.com*. https://www.natureoffice.com/unsere-leistungen/co2-footprints.

[112] HELD, Christian ; TENNIGKEIT, Timm ; TECHEL, Grit ; SEEBAUER, Matthias ; UMWELTBUNDESAMT (Hrsg.): *Analyse und Bewertung von Waldprojekten und entsprechender Standards zur freiwilligen Kompensation von Treibhausgasemissionen: Kurzfassung*. https://digital.zlb.de/viewer/resolver?urn=urn:nbn:de:kobv:109-opus-98208.

[113] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *Umweltmanagement - Ökobilanz - Anforderungen und Anleitungen (ISO 14044:2006 + Amd 1:2017 + Amd 2:2020): Deutsche Fassung EN ISO 14044:2006 + A1:2018 + A2:2020*. Berlin, Februar 2021.

[114] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *Nachhaltigkeit von Bauwerken - Bewertung der umweltbezogenen Qualität von Gebäuden - Berechnungsmethode: Deutsche Fassung EN 15978:2011.* Berlin, Oktober 2012.

[115] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *Umweltkennzeichnungen und -deklarationen - Typ III Umweltdeklarationen - Grundsätze und Verfahren (ISO 14025:2006): Deutsche und Englische Fassung EN ISO 14025:2011.* Berlin, Oktober 2011.

[116] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *Nachhaltigkeit von Bauwerken - Umweltproduktdeklarationen - Grundregeln für die Produktkategorie Bauprodukte: Deutsche Fassung EN 15804:2012+A2:2019.* Berlin, März 2020.

[117] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *Nachhaltigkeit von Bauwerken - Umweltproduktdeklarationen - Methoden für die Auswahl und Verwendung von generischen Daten: Deutsche Fassung CEN/TR 15941:2010.* https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32008L0098&from=DE. Version: Mai 2010.

[118] IPCC, 2013 (Hrsg.): *Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change.* Cambridge : Cambridge University Press, 2013.

[119] BEHR, Arno ; AGAR, David W. ; JÖRISSEN, Jakob ; VORHOLT, Andreas J.: *Einführung in die Technische Chemie.* 2. Auflage. Berlin, Heidelberg : Springer Berlin Heidelberg, 2016 https://doi.org/10.1007/978-3-662-52856-3

[120] EUROPÄISCHES PARLAMENT UND RAT: *Richtlinie 2008/98/EG des europäischen Parlaments und des Rates über Abfälle und zur Aufhebung bestimmter Richtlinien (Text von Bedeutung für den EWR): (ABI. L 312 vom 22.11.2008, S. 3).* https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:02008L0098-20180705&from=DE. Version: 19. November 2008.

[121] PE INTERNATIONAL AG ; PE INTERNATIONAL AG (Hrsg.): *GaBi Manual: GaBi User Guide.* https://gabi.sphera.com/fileadmin/GaBi_Manual/GaBi_6_manual.pdf.

[122] HEIN, Helen ; SCHWARTE, Joachim: Subsystems and element groups in life cycle assessments with vague input data. In: *Otto Graf Journal* 19 (2020), S. 61–70.

[123] CIROTH, Andreas ; DI NOI, C. ; LOHSE, T. ; SROCKA, M.: *openLCA 1.9: Comprehensive User Manual.* https://www.openlca.org/wp-content/uploads/2019/07/openLCA-1-9_User-Manual.pdf

[124] DEUTSCHES INSTITUT FÜR NORMUNG E.V.: *DIN 1319-3:1996-05: Grundlagen der Meßtechnik: Teil 3: Auswertung von Messungen einer einzelnen Meßgröße, Meßunsicherheit.*

[125] STEINMANN, Schulze S.: *bilanx-Übersetzung im Latein Wörterbuch.* https://www.frag-caesar.de/lateinwoerterbuch/bilanx-uebersetzung.html. Version: 12/02/2021 10:59:50.

[126] ROTH, Stefan: *Konzept verallgemeinerungsfähiger Module für die Sachbilanz von Produktionsprozessen.* Berlin, Prozesswissenschaften, Dissertation, 2001.

[127] JUNGE, Mark: *Simulationsgestützte Entwicklung und Optimierung einer energieeffizienten Produktionssteuerung.* Kassel, Universität Kassel, Dissertation, 2007.

[128] CIROTH, Andreas ; FLEISCHER, Günter ; STEINBACH, Jörg: Uncertainty Calculation in Life Cycle Assessments: A Combined Model of Simulation and Approximation. In: *The International Journal of Life Cycle Assessment* 9 (2004), Nr. 4, S. 216–226. https://doi.org/10.1007/BF02978597.

[129] FRISCHKNECHT, Rolf ; KOLM, Petter: Modellansatz und Algorithmus zur Berechnung von Ökobilanzen im Rahmen der Datenbank ECOINVENT. In: SCHMIDT, Mario (Hrsg.) ; SCHORB, Achim (Hrsg.): *Stoffstromanalysen in Ökobilanzen und Öko-Audits*. Berlin Heidelberg New York Tokyo : Springer, 1995, S. 33–117.

[130] EICHSTÄDT, Patrick: *Berechnung von Ökobilanzen multifunktionaler Prozesssysteme in Matlab*. Aachen, RWTH Aachen, Seminararbeit, 2012.

[131] GREENDELTA: *Technical details | openLCA.org*. https://www.openlca.org/software/technical-details/. Version: 2021.

[132] SROCKA, Michael: openLCA: Open Source Software für Life Cycle Assessments – Stand und Weiterentwicklung. In: FEIFEL, S. (Hrsg.) ; WALK, W. (Hrsg.) ; WURSTHORN, S. (Hrsg.) ; SCHEBEK, L. (Hrsg.): *Ökobilanzierung 2009*. Karlsruhe : KIT Scientific Publishing, 2009, S. 11–19.

[133] HEIJUNGS, Reinout: *CMLCA: scientific software for LCA*. http://www.cmlca.eu/. Version: 2018.

[134] SIMAPRO: *SimaPro | The world's leading LCA software*. https://simapro.com/. Version: 2021.

[135] IPOINT-SYSTEMS GMBH: *Stoffstrommanagement Software Umberto | iPoint-systems*. https://www.ifu.com/de/umberto/. Version: 2021.

[136] SPHERA: *Ökobilanz Software*. https://gabi.sphera.com/deutsch/index/. Version: 2021.

[137] UMWELTBUNDESAMT ; BUNDESREPUBLIK DEUTSCHLAND (Hrsg.): *ProBas - Willkommen bei ProBas!* https://www.probas.umweltbundesamt.de/php/index.php. Version: 2021.

[138] BBR (Hrsg.): *ÖKOBAUDAT*. https://oekobaudat.de/. Version: 2021.

[139] IBU - INSTITUT BAUEN UND UMWELT E.V.: *Veröffentlichte EPDs | Institut Bauen und Umwelt e.V.* https://ibu-epd.com/veroeffentlichte-epds/. Version: 2021

[140] ECOINVENT: *ecoinvent Database*. https://ecoinvent.org/the-ecoinvent-database/. Version: 2021.

[141] GITHUB: *GitHub - JoachimSchwarte/MultiVaLCA: Multi Value Life Cycle Assessment*. https://github.com/JoachimSchwarte/MultiVaLCA. Version: 25.11.2021.

[142] TIK: *Institut für Werkstoffe im Bauwesen | Universität Stuttgart*. https://www.iwb.uni-stuttgart.de/. Version: 2022.

[143] JESCHKE, Sabina (Hrsg.) ; JAKOBS, Eva-Maria (Hrsg.) ; DRÖGE, Alicia (Hrsg.): *Exploring Uncertainty*. Wiesbaden : Springer Fachmedien Wiesbaden, 2013. http://dx.doi.org/10.1007/978-3-658-00897-0.

[144] WOHLGENANNT, Rudolf (Hrsg.): *Wissenschaftstheorie Wissenschaft und Philosophie*. Bd. 2: *Was ist Wissenschaft?* Wiesbaden and s.l. : Vieweg+Teubner Verlag, 1969. http://dx.doi.org/10.1007/978-3-322-99161-4.

[145] CIROTH, Andreas: Uncertainties in Life Cycle Assessment: Editorial and Call for Papers: Announcing the New Section 'Uncertainties'. In: *The International Journal of Life Cycle Assessment* 9 (2004), Nr. 3, 141–142. http://dx.doi.org/10.1007/bf02994186.

[146] HILLERBRAND, Rafaela ; SCHNEIDER, Christoph: Unwissenschaftlich weil unsicher? Unsicher weil wissenschaftlich! Version: 2013. http://dx.doi.org/10.1007/978-3-658-00897-0_7. In: JESCHKE, Sabina (Hrsg.) ; JAKOBS, Eva-Maria (Hrsg.) ; DRÖGE, Alicia (Hrsg.): *Exploring Uncertainty*. Wiesbaden : Springer Fachmedien Wiesbaden, 2013, S. 151–177.

[147] VIERTL, Reinhard ; YEGANEH, Shohreh M.: Mathematische Modelle für Ungewissheit. Version: 2013. http://dx.doi.org/10.1007/978-3-658-00897-0_11. In: JESCHKE, Sabina (Hrsg.) ; JAKOBS, Eva-Maria (Hrsg.) ; DRÖGE, Alicia (Hrsg.): *Exploring Uncertainty*. Wiesbaden : Springer Fachmedien Wiesbaden, 2013, S. 271–280.

[148] BRADLEY, Richard ; DRECHSLER, Mareile: Types of Uncertainty. In: *Erkenntnis* 79 (2014), Nr. 6, S. 1225–1248. http://dx.doi.org/10.1007/s10670-013-9518-4.

[149] METZGER, Olga ; SPENGLER, Thomas: Subjektiver Erwartungsnutzen und intuitionistische Fuzzy-Werte. Version: 2017. http://dx.doi.org/10.1007/978-3-658-17580-1_6. In: SPENGLER, Thomas (Hrsg.) ; FICHTNER, Wolf (Hrsg.) ; GEIGER, Martin J. (Hrsg.) ; ROMMELFANGER, Heinrich (Hrsg.) ; METZGER, Olga (Hrsg.): *Entscheidungsunterstützung in Theorie und Praxis*. Wiesbaden : Springer Gabler, 2017, S. 109–137.

[150] KRON, Thomas: „Uncertainty" – Das ungewisse Risiko der Hybriden. Version: 2013. http://dx.doi.org/10.1007/978-3-658-00897-0_4. In: *Exploring Uncertainty*. Springer Gabler, Wiesbaden, 2013, S. 55–82.

[151] BIPM: *JCGM 200:2012 International vocabulary of metrology - Basic and general concepts and associated terms (VIPM)*. 2012 https://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2012.pdf.

[152] CHRISTOF TANNERT ; HORST-DIETRICH ELVERS ; BURKHARD JANDRIG: The ethics of uncertainty. In: *EMBO reports* 8 (2007), Nr. 10, S. 892–896. http://dx.doi.org/10.1038/sj.embor.7401072.

[153] WALKER, W. E. ; HARREMOËS, P. ; ROTMANS, J. ; VAN DER SLUIJS, J. P. ; VAN ASSELT, M.B.A. ; JANSSEN, P. ; KRAYER VON KRAUSS, M. P.: Defining Uncertainty: A Conceptual Basis for Uncertainty Management in Model-Based Decision Support. In: *Integrated Assessment* 4 (2003), Nr. 1, S. 5–17. http://dx.doi.org/10.1076/iaij.4.1.5.16466.

[154] HUIJBREGTS, Mark A. J.: *Uncertainty and variability in environmental life cycle assessment*. Amsterdam, University of Amsterdam, Dissertation, 2001.

[155] ROSS, Stuart ; EVANS, David ; WEBBER, Michael: How LCA studies deal with uncertainty. In: *The International Journal of Life Cycle Assessment* 7 (2002), Nr. 1. http://dx.doi.org/10.1007/BF02978909.

[156] IGOS, Elorri ; BENETTO, Enrico ; MEYER, Rodolphe ; BAUSTERT, Paul ; OTHONIEL, Benoit: How to treat uncertainties in life cycle assessment studies? In: *The International Journal of Life Cycle Assessment* 24 (2019), Nr. 4, S. 794–807. http://dx.doi.org/10.1007/s11367-018-1477-1.

[157] SCHWARTE, J. ; HEIN, H. L.: Treatment of uncertainties in green engineering. In: *IOP Conference Series: Materials Science and Engineering* 615 (2019), 012109. http://dx.doi.org/10.1088/1757-899x/615/1/012109.

[158] BJÖRKLUND, Anna E.: Survey of approaches to improve reliability in lca. In: *The International Journal of Life Cycle Assessment* 7 (2002), Nr. 2. http://dx.doi.org/10.1007/BF02978849.

[159] HEIN, H. ; SCHWARTE, J.: Innovative insulation materials - specific issues performing LCA in early product development phases. In: *Advanced Materials Proceedings* 3 (2018), Nr. 9, S. 531–535. https://amp.iaamonline.org/article_16178.html.

[160] CORDIS: *HOMES Key INsulating material | HOMESKIN Project | Fact Sheet | H2020 | CORDIS | European Commission: Fördervertragsnr. 636709, Horizon 2020, Europäische Kommission*. https://cordis.europa.eu/project/id/636709/de. Version: 18.02.2022.

[161] HEIN, H. ; SCHWARTE, J.: Products in Early Development Phases: Ecological Classification and Evaluation Using an Interval Arithmetic Based Calculation Approach. In: *International Scholarly and Scientific Research & Innovation* 13 (2019), Nr. 5, S. 368–374. http://dx.doi.org/10.5281/ZENODO.3298777.

[162] EUROPÄISCHE KOMMISSION: *Wall Ace project | Wall Ace – Provide insulation solutions: Fördervertragsnr. 723574, Horizon 2020, Europäische Kommission*. https://www.wall-ace.eu/. Version: 18.02.2022.

[163] HOUGHTON, John (Hrsg.): *Climate change 1995: The Science of Climate Change*. Cambridge : Cambridge Univ. Press, 1996 https://www.ipcc.ch/report/ar2/wg1/.

[164] JOOS, F. ; ROTH, R. ; FUGLESTVEDT, J. S. ; PETERS, G. P. ; ENTING, I. G. ; BLOH, W. von ; BROVKIN, V. ; BURKE, E. J. ; EBY, M. ; EDWARDS, N. R. ; FRIEDRICH, T. ; FRÖLICHER, T. L. ; HALLORAN, P. R. ; HOLDEN, P. B. ; JONES, C. ; KLEINEN, T. ; MACKENZIE, F. T. ; MATSUMOTO, K. ; MEINSHAUSEN, M. ; PLATTNER, G.-K. ; REISINGER, A. ; SEGSCHNEIDER, J. ; SHAFFER, G. ; STEINACHER, M. ; STRASSMANN, K. ; TANAKA, K. ; TIMMERMANN, A. ; WEAVER, A. J.: Carbon dioxide and climate impulse response functions for the computation of greenhouse gas metrics: a multi-model analysis. In: *Atmospheric Chemistry and Physics* 13 (2013), Nr. 5, S. 2793–2825. http://dx.doi.org/10.5194/acp-13-2793-2013.

[165] LEIDEN UNIVERSITY: *CML-IA Characterisation Factors*. https://www.universiteitleiden.nl/en/research/research-output/science/cml-ia-characterisation-factors. Version: 17.02.2022.

[166] LASVAUX, S. ; SCHIOPU, N. ; HABERT, G. ; CHEVALIER, J. ; PEUPORTIER, B.: Influence of simplification of life cycle inventories on the accuracy of impact assessment: application to construction products. In: *Journal of Cleaner Production* 79 (2014), S. 142–151. http://dx.doi.org/10.1016/j.jclepro.2014.06.003.

[167] ROŠ, Matjaž: *Unsicherheit und Fuzziness in ökologischen Bewertungen: Orientierungen zu einer robusten Praxis der Oekobilanzierung*. Zürich, ETH Zürich, Dissertation, 1998. http://dx.doi.org/10.3929/ethz-a-001990213.

[168] POHL, Christian E.: *Auch zu präzis ist ungenau: Unsicherheitsanalyse in Oekobilanzen und Alternativen zu "use many methods"*. Zürich, ETH Zürich, Dissertation, 1999. http://dx.doi.org/10.3929/ethz-a-002058560.

[169] DINKEL, FREDY: Skript. Ökobilanzen-Lebenszyklusanalyse-Life cycle analysis. (Juli 2013). https://carbotech.ch/cms/wp-content/uploads/Oekobilanz-skript_2013.pdf.

[170] CIROTH, Andreas: *Fehlerrechnung in Ökobilanzen.* Berlin, Technische Universität Berlin, Dissertation, 2001. http://dx.doi.org/10.14279/DEPOSITONCE-363.

[171] LLOYD, Shannon M. ; RIES, Robert: Characterizing, Propagating, and Analyzing Uncertainty in Life-Cycle Assessment: A Survey of Quantitative Approaches. In: *Journal of Industrial Ecology* 11 (2007), Nr. 1, S. 161–179. http://dx.doi.org/10.1162/jiec.2007.1136.

[172] HEIJUNGS, Reinout ; HUIJBREGTS, Mark A. J.: A Review of Approaches to Treat Uncertainty in LCA. In: *Complexity and Integrated Resource Management.* Osnabrück : Elsevier, 2004

[173] STREINER, David L. ; NORMAN, Geoffrey R.: "Precision" and "Accuracy": Two Terms That Are Neither. In: *Journal of Clinical Epidemiology* 59 (2006), Nr. 4, S. 327–330. http://dx.doi.org/10.1016/j.jclinepi.2005.09.005.

[174] FINNVEDEN, Göran ; HAUSCHILD, Michael Z. ; EKVALL, Tomas ; GUINÉE, Jeroen ; HEIJUNGS, Reinout ; HELLWEG, Stefanie ; KOEHLER, Annette ; PENNINGTON, David ; SUH, Sangwon: Recent developments in Life Cycle Assessment. In: *Journal of Environmental Management* 91 (2009), Nr. 1, S. 1–21. http://dx.doi.org/10.1016/j.jenvman.2009.06.018.

[175] SCRUCCA, Flavio ; BALDASSARRI, Catia ; BALDINELLI, Giorgio ; BONAMENTE, Emanuele ; RINALDI, Sara ; ROTILI, Antonella ; BARBANERA, Marco: Uncertainty in LCA: An estimation of practitioner-related effects. In: *Journal of Cleaner Production* 268 (2020), S. 122304. http://dx.doi.org/10.1016/j.jclepro.2020.122304.

[176] VIERTL, Reinhard: Fuzzy Daten und Stochastik. Version: 1999. http://dx.doi.org/10.1007/978-3-663-10120-8_10. In: SEISING, Rudolf (Hrsg.): *Fuzzy Theorie und Stochastik.* Wiesbaden and s.l. : Vieweg+Teubner Verlag, 1999 (Computational Intelligence), S. 244–250.

[177] BANDEMER, Hans: Unscharfe Analyse unscharfer Daten. Version: 1999. http://dx.doi.org/10.1007/978-3-663-10120-8_11. In: SEISING, Rudolf (Hrsg.): *Fuzzy Theorie und Stochastik.* Wiesbaden and s.l. : Vieweg+Teubner Verlag, 1999 (Computational Intelligence), S. 251–267.

[178] REINHARD VIERTL ; DIETMAR HARETER: *Beschreibung und Analyse unscharfer Information: Statistische Methoden für unscharfe Daten.* Wien : Springer-Verlag, 2006. http://dx.doi.org/10.1007/3-211-32347-3.

[179] ECKLE-KOHLER, Judith ; KOHLER, Michael: *Eine Einführung in die Statistik und ihre Anwendungen.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2009. http://dx.doi.org/10.1007/978-3-642-00471-1.

[180] LAUX, Helmut ; GILLENKIRCH, Robert M. ; SCHENK-MATHES, Heike Y.: *Entscheidungstheorie.* 8. Auflage. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012. http://dx.doi.org/10.1007/978-3-642-23511-5.

[181] SCHIEFER, Hartmut ; SCHIEFER, Felix: *Statistik für Ingenieure: Eine Einführung mit Beispielen aus der Praxis.* Wiesbaden : Springer Vieweg, 2018. http://dx.doi.org/10.1007/978-3-658-20640-6.

[182] DINKEL, Fredy: *Ökobilanzen. Lebenszyklusanalyse. Life cycle analysis.* Basel, Fachhochschule Nordwestschweiz, Skript, 2013. https://carbotech.ch/cms/wp-content/uploads/Oekobilanz-skript_2013.pdf.

[183] FECK, Norbert: *Monte-Carlo-Simulation bei der Lebenszyklusanalyse eines Hot-Dry-Rock-Heizwerkes.* Bochum, Ruhr-Universität Bochum, Dissertation, 2007. https://hss-opus.ub.ruhr-uni-bochum.de/opus4/frontdoor/index/index/docId/1701.

[184] WOLTERS, Dirk: *Struktur-und akteursorientierte Szenarioanalyse eines nachhaltigen deutschen Energiesystems im internationalen Kontext.* Wuppertal, Universität Osnabrück, Dissertation, 2001. https://osnadocs.ub.uni-osnabrueck.de/handle/urn:nbn:de:gbv:700-2002021519.

[185] GANTNER, Johannes: *Wahrscheinlichkeitsbasierte Ökobilanzierung zur Berücksichtigung von Unsicherheiten in zukünftigen Entscheidungen und Ereignissen.* Stuttgart, Universität Stuttgart, Dissertation, 2017.

[186] REAP, John ; ROMAN, Felipe ; DUNCAN, Scott ; BRAS, Bert: A survey of unresolved problems in life cycle assessment. In: *The International Journal of Life Cycle Assessment* 13 (2008), Nr. 5, S. 374–388. http://dx.doi.org/10.1007/s11367-008-0009-9.

[187] SCHWARTE, Joachim ; HEIN, Helen: On the sustainability assessment of building materials and components. In: *Otto Graf Journal* (2021), Nr. 20, S. 191–198.

[188] MAURIS, Gilles ; LASSERRE, Virginie ; FOULLOY, Laurent: A fuzzy approach for the expression of uncertainty in measurement. In: *Measurement* 29 (2001), Nr. 3, S. 165–177. http://dx.doi.org/10.1016/S0263-2241(00)00036-1.

[189] TAN, Raymond R.: Using fuzzy numbers to propagate uncertainty in matrix-based LCI. In: *The International Journal of Life Cycle Assessment* 13 (2008), Nr. 7, S. 585–592. http://dx.doi.org/10.1007/s11367-008-0032-x.

[190] CHEVALIER, Jean-Luc ; LE TÉNO, Jean-FranÇois: Life cycle analysis with ill-defined data and its application to building products. In: *The International Journal of Life Cycle Assessment* 1 (1996), Nr. 2, S. 90–96. http://dx.doi.org/10.1007/BF02978652.

[191] SCHALTEGGER, S.: *Life Cycle Assessment (LCA) — Quo vadis?* Birkhäuser Basel, 1996 https://doi.org/10.1007/978-3-0348-9022-9.

Anhang A

Ergänzende Quellcodes

A.1 Solver hansen

Code A.1: Methoden für Solver hansen

```
1 // Solver auf Basis des Verfahrens nach Hansen
2 public IvariVector solverHansen(IvariVector s) {
3   IvariMatrix matrix = this;
4   IvariVector result = new IvariVector(numberRows);
5   IvariMatrix pLeft = calculatePleft(matrix);
6   for (int i = 0; i< numberRows; i++) {
7     double cSmall;
8     IvariVector pLeftVector = new IvariVector(numberRows);
9     for (int j = 0; j < numberRows; j++) {
10      pLeftVector.setValue(j, pLeft.getValue(i, j));
11    }
12    cSmall = calculateCsmall(matrix, i);
13    IvariVector sLeft = calculateSleft(s, matrix, i);
14    IvariVector sRight = calculateSright(s, matrix, i);
15    IvariScalar resultValue = new IvariScalar();
16    resultValue.setUpperBound(multTwoVectors(pLeftVector, sRight).getUpperBound());
17    if (resultValue.getUpperBound() < 0) {
18      resultValue.setUpperBound(resultValue.getUpperBound()*cSmall);
19    }
20    resultValue.setLowerBound(multTwoVectors(pLeftVector, sLeft).getLowerBound());
21    if (resultValue.getLowerBound() >= 0) {
22      resultValue.setLowerBound(resultValue.getLowerBound()*cSmall);
23    }
24    result.setValue(i, resultValue);
25  }
26
27  return result;
28 }
29
30 // Berechnung von cSmall
31 public double calculateCsmall (IvariMatrix matrix, int i) {
32   IvariMatrix pLeft = calculatePleft(matrix);
33   double cSmall = 1/(pLeft.getValue(i, i).getLowerBound()*2-1);
```

```
34   return cSmall;
35 }
36
37 // Berechnung der Inverse M^(-1) Pleft
38 public IvariMatrix calculatePleft (IvariMatrix matrix) {
39   IvariMatrix matrixMleft = calculateMleft(matrix);
40   IvariMatrix matrixPleft = inverse(matrixMleft);
41   return matrixPleft;
42 }
43
44 //  Berechnung der unteren Grenzwerte-Matrix Mleft = B*A
45 public IvariMatrix calculateMleft (IvariMatrix matrix) {
46   IvariMatrix matrixM = calculateM(matrix);
47   IvariMatrix matrixMleft = new IvariMatrix(numberRows);
48   for (int i=0; i < numberRows; i++) {
49     for (int j=0; j< numberRows; j++) {
50       IvariScalar scalar = new IvariScalar();
51       double scalarValue = matrixM.getValue(i,j).getLowerBound();
52       scalar.setUpperBound(scalarValue);
53       scalar.setLowerBound(scalarValue);
54       matrixMleft.setValue(i, j, scalar);
55     }
56   }
57   return matrixMleft;
58 }
59
60 // Berechnung der Matrix M = B*A
61 public IvariMatrix calculateM (IvariMatrix matrix) {
62   IvariMatrix inverseB = calculateB(matrix);
63   IvariMatrix matrixM = matrix.multMatrix(inverseB);
64   return matrixM;
65 }
66
67 // Berechnung der angenaeherten Matrix B als Inverse der Mittelpunktsmatrix C
68 public IvariMatrix calculateB (IvariMatrix matrix) {
69   IvariMatrix centerMatrix = calculateC(matrix);
70   IvariMatrix matrixB = inverse(centerMatrix);
71   return matrixB;
72 }
73
74 // Berechnung der Mittelpunktsmatrix C von A
75 public IvariMatrix calculateC (IvariMatrix matrix) {
76   IvariMatrix centerMatrix = new IvariMatrix(numberRows);
77   for (int i = 0; i < numberRows; i++) {
78     for (int j = 0; j < numberRows; j++) {
79       IvariScalar scalar = new IvariScalar();
80       double difference = (matrix.getValue(i, j).getUpperBound()+matrix.getValue(i, j).
             getLowerBound())/2;
81       scalar.setUpperBound(difference);
82       scalar.setLowerBound(difference);
83       centerMatrix.setValue(i, j, scalar);
84     }
85   }
86   return centerMatrix;
```

```
87 }
88
89 // Berechnung von Sleft
90 public IvariVector calculateSleft (IvariVector s, IvariMatrix matrix, int i) {
91   IvariVector r = calculateR(s, matrix);
92   IvariVector sLeft = new IvariVector(numberRows);
93   for (int j = 0; j < numberRows; j++) {
94     IvariScalar scalar = new IvariScalar();
95     if ((j == i) || ((j!=i) && (r.getValue(j).getLowerBound() <= (r.getValue(j).getUpperBound()
           *(-1))))) {
96       double scalarValue = r.getValue(j).getLowerBound();
97       scalar.setLowerBound(scalarValue);
98       scalar.setUpperBound(scalarValue);
99       sLeft.setValue(j, scalar);
100    } else {
101      double scalarValue = (r.getValue(j).getUpperBound()*(-1));
102      scalar.setLowerBound(scalarValue);
103      scalar.setUpperBound(scalarValue);
104      sLeft.setValue(j, scalar);
105    }
106  }
107  return sLeft;
108 }
109
110 // Berechnung von Sright
111 public IvariVector calculateSright (IvariVector s, IvariMatrix matrix, int i) {
112   IvariVector r = calculateR(s, matrix);
113   IvariVector sRight = new IvariVector(numberRows);
114   for (int j = 0; j < numberRows; j++) {
115     IvariScalar scalar = new IvariScalar();
116     if ((j == i) || ((j!=i) && (r.getValue(j).getUpperBound() >= (r.getValue(j).getLowerBound()
           *(-1))))) {
117       double scalarValue = r.getValue(j).getUpperBound();
118       scalar.setLowerBound(scalarValue);
119       scalar.setUpperBound(scalarValue);
120       sRight.setValue(j, scalar);
121     } else {
122       double scalarValue = (r.getValue(j).getLowerBound()*(-1));
123       scalar.setLowerBound(scalarValue);
124       scalar.setUpperBound(scalarValue);
125       sRight.setValue(j, scalar);
126     }
127   }
128   return sRight;
129 }
130
131 // Berechnung von r
132 public IvariVector calculateR (IvariVector s, IvariMatrix matrix) {
133   IvariVector r = new IvariVector(numberRows);
134   IvariMatrix B = calculateB(matrix);
135   r = B.multVector(s);
136   return r;
137 }
138
```

```
139 // Multiplikation zweier Vektoren
140 public IvariScalar multTwoVectors(IvariVector y, IvariVector z) throws ArithmeticException {
141   if (y.getSize() != z.getSize()) {
142     throw new ArithmeticException("Abweichende Dimensionen");
143   }
144   IvariScalar r = new IvariScalar(0,0);
145   for (int i=0; i<numberRows; i++) {
146       r = r.add(y.getValue(i).mult(z.getValue(i)));
147   }
148   return r;
149 }
```

A.2 Solver rohn

Code A.2: Methoden zur Implementierung der Rohn-Methode

```
1  // Solver auf Basis des Verfahrens nach Rohn
2  public IvariVector solverRohn(IvariVector s) {
3    IvariVector result = new IvariVector(numberRows);
4    IvariVector xSnakeL = calculateXsnakeL(s, this);
5    IvariVector xSnakeR = calculateXsnakeR(s, this);
6    for (int i = 0; i < s.getSize(); i++)
7    {
8      double cSmall = calculateCsmall(this, i);
9      if (xSnakeL.getValue(i).getLowerBound()< xSnakeL.getValue(i).mult(cSmall).getLowerBound()) {
10       result.getValue(i).setLowerBound(xSnakeL.getValue(i).getLowerBound());
11       } else {
12         result.getValue(i).setLowerBound(xSnakeL.getValue(i).mult(cSmall).getLowerBound());
13       }
14       if (xSnakeR.getValue(i).getUpperBound() > xSnakeR.getValue(i).mult(cSmall).getUpperBound())
             {
15         result.getValue(i).setUpperBound(xSnakeR.getValue(i).getUpperBound());
16       } else {
17         result.getValue(i).setUpperBound(xSnakeR.getValue(i).mult(cSmall).getUpperBound());
18     }
19   }
20   return result;
21 }
22
23 // Berechnung von XsnakeL als IvariVector result
24 public IvariVector calculateXsnakeL(IvariVector s, IvariMatrix matrix) {
25   IvariMatrix pLeft = calculatePleft(matrix);
26   IvariVector rMean = calculateRmean(s, matrix);
27   IvariVector rMeanAbs = calculateRmeanAbs(s, matrix);
28   IvariVector xStar = calculateXstar(s, matrix);
29   IvariVector result = new IvariVector  (s.getSize());
30   for (int i = 0; i < s.getSize(); i++) {
31     IvariScalar scalar = xStar.getValue(i).mult(-1).add(pLeft.getValue(i, i).mult(rMean.getValue(
           i).add(rMeanAbs.getValue(i))));
32     result.setValue(i, scalar);
33   }
34   return result;
35 }
```

```
36
37 // Berechnung von XsnakeR
38 public IvariVector calculateXsnakeR(IvariVector s, IvariMatrix matrix) {
39   IvariMatrix pLeft = calculatePleft(matrix);
40   IvariVector rMean = calculateRmean(s, matrix);
41   IvariVector rMeanAbs = calculateRmeanAbs(s, matrix);
42   IvariVector xStar = calculateXstar(s, matrix);
43   IvariVector result = new IvariVector  (s.getSize());
44   for (int i = 0; i < s.getSize(); i++) {
45     IvariScalar scalar = xStar.getValue(i).add(pLeft.getValue(i, i).mult(rMean.getValue(i).sub(
         rMeanAbs.getValue(i))));
46     result.setValue(i, scalar);
47   }
48   return result;
49 }
50
51 // Berechnung der Inverse M^(-1) Pleft
52 public IvariMatrix calculatePleft (IvariMatrix matrix) {
53   IvariMatrix matrixMleft = calculateMleft(matrix);
54   IvariMatrix matrixPleft = inverse(matrixMleft);
55   return matrixPleft;
56 }
57
58 // Berechnung der Mittelwerte von r als IvariVector rMedium
59 public IvariVector calculateRmean (IvariVector s, IvariMatrix matrix) {
60   IvariVector rMedium = new IvariVector(numberRows);
61   IvariVector r = calculateR(s, matrix);
62   for (int i = 0; i< s.getSize(); i++) {
63     double meanValue = (r.getValue(i).getUpperBound()+r.getValue(i).getLowerBound())/2;
64     IvariScalar scalar = new IvariScalar (meanValue, meanValue);
65     rMedium.setValue(i, scalar);
66   }
67   return rMedium;
68 }
69
70 // Berechnung von r
71 public IvariVector calculateR (IvariVector s, IvariMatrix matrix) {
72   IvariVector r = new IvariVector(numberRows);
73   IvariMatrix B = calculateB(matrix);
74   r = B.multVector(s);
75   return r;
76 }
77
78 // Berechnung von Rstar als IvariVector rStar
79 public IvariVector calculateRstar (IvariVector s, IvariMatrix matrix) {
80   IvariVector rMean = calculateRmean(s, matrix);
81   IvariVector rWidth = calculateRwidth(s, matrix);
82   IvariVector rStar = new IvariVector (s.getSize());
83   for (int i = 0; i< s.getSize(); i++) {
84     double star = Math.abs(rMean.getValue(i).getUpperBound())+rWidth.getValue(i).getUpperBound();
85     IvariScalar scalar = new IvariScalar (star, star);
86     rStar.setValue(i, scalar);
87   }
88   return rStar;
```

```
89 }
90
91 // Berechnung der halben Intervallweite von r als IvariVector rWidth
92 public IvariVector calculateRwidth (IvariVector s, IvariMatrix matrix) {
93   IvariVector rWidth = new IvariVector(numberRows);
94   IvariVector r = calculateR(s, matrix);
95   for (int i = 0; i< s.getSize(); i++) {
96     double width = (r.getValue(i).getUpperBound()-r.getValue(i).getLowerBound())/2;
97     IvariScalar scalar = new IvariScalar (width, width);
98     rWidth.setValue(i, scalar);
99   }
100  return rWidth;
101 }
102
103 // Berechnung der Absolutwerte von rMedium als IvariVector rmeanAbs
104 public IvariVector calculateRmeanAbs (IvariVector s, IvariMatrix matrix) {
105   IvariVector rmeanAbs = new IvariVector(numberRows);
106   IvariVector r = calculateRmean(s, matrix);
107   for (int i = 0; i< s.getSize(); i++) {
108     double meanAbsValue = Math.abs(r.getValue(i).getUpperBound());
109     IvariScalar scalar = new IvariScalar (meanAbsValue, meanAbsValue);
110     rmeanAbs.setValue(i, scalar);
111   }
112   return rmeanAbs;
113 }
114
115 // Berechnung von Xstar als IvariVector result
116 public IvariVector calculateXstar (IvariVector s, IvariMatrix matrix) {
117   IvariMatrix pLeft = calculatePleft(matrix);
118   IvariVector rStar = calculateRstar(s, matrix);
119   IvariVector result = pLeft.multVector(rStar);
120   return result;
121 }
```

A.3 Solver ning22

Code A.3: Methoden für Solver ning22

```
1 // Solver auf Basis des Theorems 2.2 nach Ning
2 public IvariVector solverNing22(IvariVector s) {
3   IvariVector result = new IvariVector(numberRows);
4   IvariVector betaI = calculateBetaI(s, this);
5   IvariVector alphaI = calculateAlphaI(s, this);
6   for (int i = 0; i < s.getSize(); i++) {
7     IvariScalar scalar = new IvariScalar();
8     scalar = ((s.getValue(i).add(betaI.getValue(i))).div(this.getValue(i, i).add(alphaI.getValue(
          i))));
9     result.setValue(i, scalar);
10  }
11  return result;
12 }
13
14 // Berechnung von [-beta, beta] als IvariVector betaI
```

```
15 public IvariVector calculateBetaI(IvariVector s, IvariMatrix matrix) {
16   IvariVector betaI = new IvariVector (s.getSize());
17   IvariVector beta = calculateBeta(s, matrix);
18   for (int i = 0; i < numberRows; i++) {
19     IvariScalar scalar = beta.getValue(i).mult(new IvariScalar(-1, 1));
20     betaI.setValue(i, scalar);
21   }
22   return betaI;
23 }
24
25 // Berechnung von beta als IvariVector beta
26 public IvariVector calculateBeta(IvariVector s, IvariMatrix matrix) {
27   IvariVector beta = new IvariVector (s.getSize());
28   IvariVector u = calculateU(s, matrix);
29   IvariMatrix compareMatrix = calcCompareMatrix(matrix);
30   IvariMatrix invCompMatrix = inverse(compareMatrix);
31   IvariVector bAbs = calculateBabs(s);
32   for (int i = 0; i < numberRows; i++) {
33     beta.setValue(i, (u.getValue(i).div(invCompMatrix.getValue(i, i))).sub(bAbs.getValue(i)));
34   }
35   return beta;
36 }
37
38 // Berechnung von u = <A>-1|b| als IvariVector u
39 public IvariVector calculateU(IvariVector s, IvariMatrix matrix) {
40   IvariVector u = new IvariVector(s.getSize());
41   IvariMatrix compareMatrix = calcCompareMatrix(matrix);
42   IvariMatrix invCompMatrix = inverse(compareMatrix);
43   IvariVector bAbs = calculateBabs(s);
44   u = invCompMatrix.multVector(bAbs);
45   return u;
46 }
47
48 // Berechnung der Vergleichsmatrix M(A) als IvariMatrix compareMatrix
49 public IvariMatrix calcCompareMatrix(IvariMatrix matrix) {
50   IvariMatrix compareMatrix = new IvariMatrix(numberRows);
51   for (int i = 0; i < numberRows; i++) {
52     for (int j = 0; j < numberRows; j++) {
53       double scalarValue;
54       if ((i == j) && (Math.abs(matrix.getValue(i, j).getLowerBound()) <= Math.abs(matrix.
             getValue(i, j).getUpperBound()))) {
55         scalarValue = Math.abs(matrix.getValue(i, j).getLowerBound());
56       } else if ((i == j) && (Math.abs(matrix.getValue(i, j).getLowerBound()) > Math.abs(matrix.
             getValue(i, j).getUpperBound()))) {
57         scalarValue = Math.abs(matrix.getValue(i, j).getUpperBound());
58       } else if ((i != j) && (Math.abs(matrix.getValue(i, j).getLowerBound()) <= Math.abs(matrix.
             getValue(i, j).getUpperBound()))) {
59         scalarValue = Math.abs(matrix.getValue(i, j).getUpperBound())*(-1);
60       } else {
61         scalarValue = Math.abs(matrix.getValue(i, j).getLowerBound())*(-1);
62       }
63       IvariScalar scalar = new IvariScalar (scalarValue, scalarValue);
64       compareMatrix.setValue(i, j, scalar);
65     }
```

```
66    }
67    return compareMatrix;
68  }
69
70  // Berechnung von |b| (Maximalwerte von b) als IvariMatrix bAbs
71  public IvariVector calculateBabs(IvariVector s) {
72    IvariVector bAbs = new IvariVector(s.getSize());
73      for (int i = 0; i < numberRows; i++) {
74      double scalarValue;
75      if (Math.abs(s.getValue(i).getLowerBound()) >= Math.abs(s.getValue(i).getUpperBound())) {
76        scalarValue = Math.abs(s.getValue(i).getLowerBound());
77      } else {
78        scalarValue = Math.abs(s.getValue(i).getUpperBound());
79      }
80      IvariScalar scalar = new IvariScalar (scalarValue, scalarValue);
81      bAbs.setValue(i, scalar);
82    }
83    return bAbs;
84  }
85
86  // Berechnung von [-alpha, alpha] als IvariVector alphaI
87  public IvariVector calculateAlphaI(IvariVector s, IvariMatrix matrix) {
88    IvariVector alphaI = new IvariVector (s.getSize());
89    IvariVector alpha = calculateAlpha(s, matrix);
90    for (int i = 0; i < numberRows; i++) {
91      IvariScalar scalar = alpha.getValue(i).mult(new IvariScalar(-1, 1));
92      alphaI.setValue(i, scalar);
93    }
94    return alphaI;
95  }
96
97  // Berechnung von alpha als IvariVector alpha
98  public IvariVector calculateAlpha(IvariVector s, IvariMatrix matrix) {
99    IvariVector alpha = new IvariVector (s.getSize());
100   IvariMatrix compareMatrix = calcCompareMatrix(matrix);
101   IvariMatrix invCompMatrix = inverse(compareMatrix);
102   for (int i = 0; i < numberRows; i++) {
103     IvariScalar scalar = compareMatrix.getValue(i, i).sub(invCompMatrix.getValue(i, i).inv());
104     alpha.setValue(i, scalar);
105   }
106   return alpha;
107 }
```

A.4 Solver ning26

Code A.4: Methoden für Solver ning26

```
1  // Solver auf Basis Theorem 2.6 nach Ning
2  public IvariVector solverNing26(IvariVector s) throws ArithmeticException {
3    IvariVector result = new IvariVector(s.getSize());
4    IvariMatrix temp = new IvariMatrix (this.getSize());
5    for (int i = 0; i<this.getSize(); i++) {
6      for (int j = 0; j<this.getSize(); j++) {
```

```
      temp.setValue(i, j, this.getValue(i, j));
    }
  }
  if (checkInvPositivity(this) == false) {
    throw new ArithmeticException(Methode ausserhalb des Gueltigkeitsbereichs);
  } else {
    IvariVector result1 = new IvariVector(numberRows);
    IvariVector result2 = new IvariVector(numberRows);
    IvariMatrix aLeft = calculateAleft(this);
    IvariMatrix aRight = calculateAright(this);
    IvariVector bLeft = calculateBleft(s);
    IvariVector bRight = calculateBright(s);
    IvariMatrix aMatrix1 = new IvariMatrix(numberRows);
    IvariMatrix aMatrix2 = new IvariMatrix(numberRows);
    IvariVector preResult = solverNingTheorem25(s);
    for (int i = 0; i< s.getSize(); i++) {
      for (int j = 0; j< s.getSize(); j++) {
        if (preResult.getValue(j).getLowerBound()>=0) {
          aMatrix1.setValue(i, j, aRight.getValue(i, j));
        } else {
          aMatrix1.setValue(i, j, aLeft.getValue(i, j));
        }
        if (preResult.getValue(j).getUpperBound()<=0) {
          aMatrix2.setValue(i, j, aRight.getValue(i, j));
        } else {
          aMatrix2.setValue(i, j, aLeft.getValue(i, j));
        }
      }
    }
    result1 = inverseOpt(aMatrix1).multVector(bLeft);
    result2 = inverseOpt(aMatrix2).multVector(bRight);
    for (int i = 0; i< s.getSize(); i++) {
      result.getValue(i).setLowerBound(result1.getValue(i).getLowerBound());
      result.getValue(i).setUpperBound(result2.getValue(i).getUpperBound());
    }
  }
  return result;
}

//Methode zum Ueberpruefen, ob Matrix invers-positiv ist
private boolean checkInvPositivity(IvariMatrix matrix) {
  boolean positivity= true;
  IvariMatrix inverse = inverseOpt(matrix);
  for (int i = 0; i < inverse.numberRows; i++) {
    for (int j = 0; j < inverse.numberColumns; j++) {
      if (inverse.getValue(i, j).getLowerBound() < 0) {
        positivity = false;
      }
    }
  }
  return positivity;
}

// Berechnung von Aleft
```

```
61 public IvariMatrix calculateAleft (IvariMatrix matrix) {
62   IvariMatrix aLeft = new IvariMatrix(numberColumns);
63   for (int i = 0; i < numberRows; i++) {
64     for (int j = 0; j < numberColumns; j++) {
65       double scalarValue;
66       scalarValue = matrix.getValue(i, j).getLowerBound();
67       IvariScalar scalar = new IvariScalar (scalarValue, scalarValue);
68       aLeft.setValue(i, j, scalar);
69     }
70   }
71   return aLeft;
72 }
73
74 // Berechnung von Aright
75 public IvariMatrix calculateAright (IvariMatrix matrix) {
76   IvariMatrix aLeft = new IvariMatrix(numberColumns);
77   for (int i = 0; i < numberRows; i++) {
78     for (int j = 0; j < numberColumns; j++) {
79       double scalarValue;
80       scalarValue = matrix.getValue(i, j).getUpperBound();
81       IvariScalar scalar = new IvariScalar (scalarValue, scalarValue);
82       aLeft.setValue(i, j, scalar);
83     }
84   }
85   return aLeft;
86 }
87 // Berechnung von Bleft
88 public IvariVector calculateBleft (IvariVector s) {
89   IvariVector r = new IvariVector(numberRows);
90   for (int i = 0; i< s.getSize(); i++) {
91     double value = s.getValue(i).getLowerBound();
92     IvariScalar scalar = new IvariScalar(value, value);
93     r.setValue(i, scalar);
94   }
95   return r;
96 }
97
98 // Berechnung von Bright
99 public IvariVector calculateBright (IvariVector s) {
100   IvariVector r = new IvariVector(numberRows);
101   for (int i = 0; i< s.getSize(); i++) {
102     double value = s.getValue(i).getUpperBound();
103     IvariScalar scalar = new IvariScalar(value, value);
104     r.setValue(i, scalar);
105   }
106   return r;
107 }
108
109 // Methode zur Berechnung nach Theorem 2.5 von Ning
110 public IvariVector solverNing25(IvariVector s) {
111   IvariVector result = new IvariVector(numberRows);
112   IvariMatrix aInverse = calculateAInverse(this);
113   result = aInverse.multVector(s);
114   return result;
```

```
115 }
116
117 // Berechnung der Inversen von A
118 public IvariMatrix calculateAInverse (IvariMatrix matrix) {
119   IvariMatrix aLeftInverse = calculateArightInverse(matrix);
120   IvariMatrix aRightInverse = calculateAleftInverse(matrix);
121   IvariMatrix aInverse = new IvariMatrix(numberRows);
122   for (int i = 0; i < numberRows; i++) {
123     for (int j = 0; j < numberColumns; j++) {
124       IvariScalar scalar = new IvariScalar();
125       if (aLeftInverse.getValue(i, j).getLowerBound()<= aRightInverse.getValue(i, j).
              getLowerBound()) {
126         double minValue = aLeftInverse.getValue(i, j).getLowerBound();
127         scalar.setLowerBound(minValue);
128       } else {
129         double minValue = aRightInverse.getValue(i, j).getLowerBound();
130         scalar.setLowerBound(minValue);
131       }
132       if (aLeftInverse.getValue(i, j).getUpperBound()>= aRightInverse.getValue(i, j).
              getUpperBound()) {
133         double maxValue = aLeftInverse.getValue(i, j).getUpperBound();
134         scalar.setUpperBound(maxValue);
135       } else {
136         double maxValue = aRightInverse.getValue(i, j).getUpperBound();
137         scalar.setUpperBound(maxValue);
138       }
139       aInverse.setValue(i, j, scalar);
140     }
141   }
142   return aInverse;
143 }
144
145 // Berechnung von AleftInverse
146 public IvariMatrix calculateAleftInverse (IvariMatrix matrix) {
147   IvariMatrix aLeft = calculateAleft(matrix);
148   IvariMatrix aLeftInverse = inverseOpt(aLeft);
149   return aLeftInverse;
150 }
151
152 // Berechnung von ArightInverse
153 public IvariMatrix calculateArightInverse (IvariMatrix matrix) {
154   IvariMatrix aRight = calculateAright(matrix);
155   IvariMatrix aRightInverse = inverseOpt(aRight); /
156   return aRightInverse;
157 }
158
159 // Berechnung der Inversen auf Basis der Gausschen Elimination
160 public IvariMatrix inverseOpt(IvariMatrix matrix) {
161     IvariMatrix temp = new IvariMatrix(numberRows, numberColumns);
162     IvariMatrix unityMatrix = new IvariMatrix(numberRows, numberColumns);
163     IvariScalar one = new IvariScalar(1, 1);
164     IvariScalar zero = new IvariScalar(0, 0);
165   for (int i = 0; i < matrix.numberRows; i++) {
166     for (int j = 0; j < matrix.numberColumns; j++) {
```

```
167       temp.setValue(i,j,matrix.getValue(i, j));
168     }
169   }
170 // Einheitsmatrix
171   for (int i = 0; i < matrix.numberRows; i++) {
172     for (int j = 0; j < matrix.numberColumns; j++) {
173       unityMatrix.setValue(i,j, (i==j ? one : zero));
174     }
175   }
176   // Berechnung
177   for (int j=0; j<matrix.numberRows; j++) {
178     int p = j;
179     while ((p < matrix.numberRows) && (temp.getValue(p,j).getLowerBound()== 0 && temp.getValue(p,
          j).getUpperBound()==0)) {
180         p++;
181     }
182     if (p == matrix.numberRows) {
183        return null;
184     }
185     if (p != j) {
186        temp.changeRows(j, p);
187        unityMatrix.changeRows(j, p);
188     }
189     IvariScalar f = temp.getValue(j,j);
190     for (int k=0; k<matrix.numberRows; k++) {
191         temp.setValue(j,k, (temp.getValue(j, k).div(f)));
192           unityMatrix.setValue(j,k, (unityMatrix.getValue(j, k).div(f)));
193     }
194     for (int i=0; i<matrix.numberRows; i++) {
195         if (i == j) continue;
196         IvariScalar g = temp.getValue(i,j);
197          for (int k=0; k<matrix.numberRows; k++) {
198               temp.setValue(i,k, temp.getValue(i,k).sub(g.mult(temp.getValue(j,k))));
199               unityMatrix.setValue(i,k,unityMatrix.getValue(i, k).sub(g.mult(unityMatrix.getValue
                    (j, k))));
200         }
201     }
202   }
203     return unityMatrix;
204 }
```

A.5 Solver beeck

Code A.5: Methoden für Solver beeck

```
1 // Solver nach Verfahren von Beeck
2 public IvariVector solverBeeck(IvariVector s) throws Exception  {
3   IvariVector result = new IvariVector(numberRows);
4   IvariMatrix techu = techu(this);
5   IvariMatrix techo = techo(this);
6   IvariVector bLow = generateBlow(s);
7   IvariVector bUpp = generateBupp(s);
8   IvariVector resultAuppBlow = inverseOpt(techo).multVector(bLow);
```

```
  IvariVector resultAlowBupp = inverseOpt(techu).multVector(bUpp);
  IvariVector resultAlowBlow = inverseOpt(techu).multVector(bLow);
  IvariVector resultAuppBupp = inverseOpt(techo).multVector(bUpp);
  int case1 = 0, case2 = 0, case3 = 0;
  for (int i =0; i < s.getSize(); i++) {
    if (s.getValue(i).getLowerBound() >= 0) {
      case1++;
    } else if (s.getValue(i).getUpperBound() <= 0) {
      case2++;
    } else if (s.getValue(i).getUpperBound() > 0 && s.getValue(i).getLowerBound() < 0) {
      case3++;
    } else throw new ArithmeticException(Ausserhalb des Gueltigkeitsbereichs dieser Methode);
  }
  if (case1 == s.getSize()) {
    for (int i = 0; i < s.getSize(); i++) {
      result.getValue(i).setLowerBound(resultAuppBlow.getValue(i).getLowerBound());
      result.getValue(i).setUpperBound(resultAlowBupp.getValue(i).getLowerBound());
    }
  } else if (case2 == s.getSize()) {
    for (int i = 0; i < s.getSize(); i++) {
      result.getValue(i).setLowerBound(resultAlowBlow.getValue(i).getLowerBound());
      result.getValue(i).setUpperBound(resultAuppBupp.getValue(i).getLowerBound());
    }
  } else if (case3 == s.getSize()) {
    for (int i = 0; i < s.getSize(); i++) {
      result.getValue(i).setLowerBound(resultAlowBlow.getValue(i).getLowerBound());
      result.getValue(i).setUpperBound(resultAlowBupp.getValue(i).getLowerBound());
    }
    } else throw new ArithmeticException(Ausserhalb des Gueltigkeitsbereichs dieser Methode);
  return result;
}

// Berechnung von Blow
public IvariVector generateBlow(IvariVector s) {
  IvariVector bLow = new IvariVector(s.getSize());
  for (int i = 0; i < s.getSize(); i++) {
    IvariScalar low = new IvariScalar(s.getValue(i).getLowerBound(), s.getValue(i).getLowerBound
        ());
    bLow.setValue(i, low);
  }
  return bLow;
}

// Berechnung von Bupp
public IvariVector generateBupp(IvariVector s) {
  IvariVector bUpp = new IvariVector(s.getSize());
  for (int i = 0; i < s.getSize(); i++) {
    IvariScalar upp = new IvariScalar(s.getValue(i).getUpperBound(), s.getValue(i).getUpperBound
        ());
    bUpp.setValue(i, upp);
  }
  return bUpp;
}

```

```
61 // Berechnung von techu
62 public IvariMatrix techu(IvariMatrix matrix) {
63   IvariMatrix techu = new IvariMatrix(numberColumns);
64   for (int i = 0; i < numberRows; i++) {
65     for (int j = 0; j < numberColumns; j++) {
66       IvariScalar scalarLow = new IvariScalar();
67       scalarLow.setLowerBound(matrix.getValue(i, j).getLowerBound());
68       scalarLow.setUpperBound(matrix.getValue(i, j).getLowerBound());
69       techu.setValue(i, j, scalarLow);
70     }
71   }
72   return techu;
73 }
74
75 // Berechnung von techo
76 public IvariMatrix techo(IvariMatrix matrix) {
77   IvariMatrix techo = new IvariMatrix(numberColumns);
78   for (int i = 0; i < numberRows; i++) {
79     for (int j = 0; j < numberColumns; j++) {
80       IvariScalar scalarUpp = new IvariScalar();
81       scalarUpp.setLowerBound(matrix.getValue(i, j).getUpperBound());
82       scalarUpp.setUpperBound(matrix.getValue(i, j).getUpperBound());
83       techo.setValue(i, j, scalarUpp);
84     }
85   }
86   return techo;
87 }
```

A.6 Solver picky

Code A.6: Solver picky

```
1 // Solver picky als modifiziertes Verfahren
2 protected IvariVector picky(IvariVector s) {
3   IvariMatrix matrix1 = new IvariMatrix(numberRows);
4   IvariMatrix matrix2 = new IvariMatrix(numberRows);
5   IvariMatrix matrix3 = new IvariMatrix(numberRows);
6   IvariMatrix matrix4 = new IvariMatrix(numberRows);
7   IvariMatrix matrix5 = new IvariMatrix(numberRows);
8   IvariMatrix matrix6 = new IvariMatrix(numberRows);
9   IvariVector result = new IvariVector(numberRows);
10  IvariVector result1 = new IvariVector(numberRows);
11  IvariVector vector1 = new IvariVector(numberRows);
12  IvariVector vector2 = new IvariVector(numberRows);
13  IvariVector vector3 = new IvariVector(numberRows);
14  IvariVector vector4 = new IvariVector(numberRows);
15  IvariVector vector5 = new IvariVector(numberRows);
16  IvariVector vector6 = new IvariVector(numberRows);
17  for (int i=0; i<result.getSize(); i++) {
18    vector1.setValue(i, s.getValue(i));
19    vector2.setValue(i, s.getValue(i));
20    vector3.setValue(i, s.getValue(i));
21    vector4.setValue(i, s.getValue(i));
```

```
    vector5.setValue(i, s.getValue(i));
    vector6.setValue(i, s.getValue(i));
    for (int j=0; j<result.getSize(); j++) {
      matrix1.setValue(i,j,this.getValue(i,j));
      matrix2.setValue(i,j,this.getValue(i,j));
      matrix3.setValue(i,j,this.getValue(i,j));
      matrix4.setValue(i,j,this.getValue(i,j));
      matrix5.setValue(i,j,this.getValue(i,j));
      matrix6.setValue(i,j,this.getValue(i,j));
    }
  }
  boolean r1exists=true;
  try {
    result1 = matrix1.gauss(Solver.siga, vector1);
  } catch (Exception e) {
    r1exists=false;
  }
  IvariVector r2 = new IvariVector(numberRows);
  boolean r2exists=true;
  try {
    r2 = matrix2.gauss(Solver.salt, vector2);
  } catch (Exception e) {
    r2exists=false;
  }

  //adivaOirign
  IvariVector r3 = new IvariVector(numberRows);
  boolean r3exists=true;
  try {
    r3 = matrix3.adivaOrigin(vector3);
  } catch (Exception e) {
    r3exists=false;
  }

  //adiperm
  IvariVector r4 = new IvariVector(numberRows);
  boolean r4exists=true;
  try {
    r4 = matrix4.adiperm(vector4);
  } catch (Exception e) {
    r4exists=false;
  }

  //hansen
  IvariVector r5 = new IvariVector(numberRows);
  boolean r5exists=true;
  try {
    r5 = matrix5.solverHansen(vector5);
  } catch (Exception e) {
    r5exists=false;
  }

  //ning26
  IvariVector r6 = new IvariVector(numberRows);
```

```
boolean r6exists=true;
try {
  r6 = matrix6.solverNingTheorem26(vector6);
} catch (Exception e) {
  r6exists=false;;
}
for (int i=0; i<result.getSize(); i++) {
  IvariScalar sca = new IvariScalar();
  Double r1l = result1.getValue(i).getLowerBound();
  Double r1u = result1.getValue(i).getUpperBound();
  Double r2l = r2.getValue(i).getLowerBound();
  Double r2u = r2.getValue(i).getUpperBound();
  Double r3l = r3.getValue(i).getLowerBound();
  Double r3u = r3.getValue(i).getUpperBound();
  Double r4l = r4.getValue(i).getLowerBound();
  Double r4u = r4.getValue(i).getUpperBound();
  Double r5l = r5.getValue(i).getLowerBound();
  Double r5u = r5.getValue(i).getUpperBound();
  Double r6l = r6.getValue(i).getLowerBound();
  Double r6u = r6.getValue(i).getUpperBound();
  if ((r1exists==false) || (r2exists==false) || (r3exists==false) || (r4exists==false) || (
      r5exists==false) || (r6exists==false)) {
  if (r1exists==false) {
      r1l=Double.MIN_VALUE;
      r1u=Double.MAX_VALUE;
    }
    if (r2exists==false) {
      r2l=Double.MIN_VALUE;
      r2u=Double.MAX_VALUE;
    }
    if (r3exists==false) {
      r3l=Double.MIN_VALUE;
      r3u=Double.MAX_VALUE;
    }
    if (r4exists==false) {
      r4l=Double.MIN_VALUE;
      r4u=Double.MAX_VALUE;
    }
    if (r5exists==false) {
      r5l=Double.MIN_VALUE;
      r5u=Double.MAX_VALUE;
    }
    if (r6exists==false) {
      r6l=Double.MIN_VALUE;
      r6u=Double.MAX_VALUE;
    }
  }
  if ((r1exists==true)||(r2exists==true) ||(r3exists==true) ||(r4exists==true) ||(r5exists==
      true)||(r6exists==true)) {
    if ((r1l>=r2l) && (r1l>=r3l) && (r1l>=r4l) && (r1l>=r5l) && (r1l>=r6l)) {
      sca.setLowerBound(r1l);
    } else if ((r2l>=r1l) && (r2l>=r3l) && (r2l>=r4l)  && (r2l>=r5l) && (r2l>=r6l)) {
      sca.setLowerBound(r2l);
    } else if ((r3l>=r1l) && (r3l>=r2l) && (r3l>=r4l) && (r3l>=r5l) && (r3l>=r6l)) {
```

```
128         sca.setLowerBound(r3l);
129       } else if ((r4l>=r1l) && (r4l>=r2l) && (r4l>=r3l) && (r4l>=r5l) && (r4l>=r6l)) {
130         sca.setLowerBound(r4l);
131       } else if ((r5l>=r1l) && (r5l>=r2l) && (r5l>=r3l) && (r5l>=r4l) && (r5l>=r6l)) {
132         sca.setLowerBound(r5l);
133       } else {
134         sca.setLowerBound(r6l);
135       }
136       if ((r1u<=r2u) && (r1u<=r3u) && (r1u<=r4u) && (r1u<=r5u) && (r1u<=r6u)){
137         sca.setUpperBound(r1u);
138       } else if ((r2u<=r1u) && (r2u<=r3u) && (r2u<=r4u) && (r2u<=r5u) && (r2u<=r6u)) {
139         sca.setUpperBound(r2u);
140       } else if ((r3u<=r1u) && (r3u<=r2u) && (r3u<=r4u) && (r3u<=r5u) && (r3u<=r6u)) {
141         sca.setUpperBound(r3u);
142       } else if ((r4u<=r1u) && (r4u<=r2u) && (r4u<=r3u) && (r4u<=r5u) && (r4u<=r6u)) {
143         sca.setUpperBound(r4u);
144       } else if ((r5u<=r1u) && (r5u<=r2u) && (r5u<=r3u) && (r5u<=r4u) && (r5u<=r6u)) {
145         sca.setUpperBound(r5u);
146       } else {
147         sca.setUpperBound(r6u);
148       }
149       result.setValue(i,sca);
150     } else {
151       result=null;
152     }
153   }
154   return result;
155 }
```

A.7 Solver quickpick

Code A.7: Solver quickpick

```
1  // Solver quickpick als modifiziertes Verfahren
2  protected IvariVector quickpick(IvariVector s) {
3    IvariMatrix matrix1 = new IvariMatrix(numberRows);
4    IvariMatrix matrix2 = new IvariMatrix(numberRows);
5    IvariMatrix matrix3 = new IvariMatrix(numberRows);
6    IvariMatrix matrix4 = new IvariMatrix(numberRows);
7    IvariVector result = new IvariVector(numberRows);
8    IvariVector result1 = new IvariVector(numberRows);
9    boolean r1exists=true;
10   IvariVector vector1 = new IvariVector(numberRows);
11   IvariVector vector2 = new IvariVector(numberRows);
12   IvariVector vector3 = new IvariVector(numberRows);
13   IvariVector vector4 = new IvariVector(numberRows);
14   for (int i=0; i<result.getSize(); i++) {
15     vector1.setValue(i, s.getValue(i));
16     vector2.setValue(i, s.getValue(i));
17     vector3.setValue(i, s.getValue(i));
18     vector4.setValue(i, s.getValue(i));
19     for (int j=0; j<result.getSize(); j++) {
20       matrix1.setValue(i,j,this.getValue(i,j));
```

```
      matrix2.setValue(i,j,this.getValue(i,j));
      matrix3.setValue(i,j,this.getValue(i,j));
      matrix4.setValue(i,j,this.getValue(i,j));
    }
  }
  try {
    result1 = matrix1.gauss(Solver.siga, vector1);
  } catch (Exception e) {
    r1exists=false;
  }
  IvariVector r2 = new IvariVector(numberRows);
  boolean r2exists=true;
  try {
    r2 = matrix2.gauss(Solver.salt, vector2);
  } catch (Exception e) {
    r2exists=false;
  }
  //hansen
  IvariVector r3 = new IvariVector(numberRows);
  boolean r3exists=true;
  try {
    r3 = matrix3.solverHansen(vector3);
  } catch (Exception e) {
    r3exists=false;
  }
  //ning26
  IvariVector r4 = new IvariVector(numberRows);
  boolean r4exists=true;
  try {
    r4 = matrix4.solverNingTheorem26(vector4);
  } catch (Exception e) {
    r4exists=false;
  }
  for (int i=0; i<result.getSize(); i++) {
    IvariScalar sca = new IvariScalar();
    Double r1l = result1.getValue(i).getLowerBound();
    Double r1u = result1.getValue(i).getUpperBound();
    Double r2l = r2.getValue(i).getLowerBound();
    Double r2u = r2.getValue(i).getUpperBound();
    Double r3l = r3.getValue(i).getLowerBound();
    Double r3u = r3.getValue(i).getUpperBound();
    Double r4l = r4.getValue(i).getLowerBound();
    Double r4u = r4.getValue(i).getUpperBound();
    if ((r1exists==false) || (r2exists==false) || (r3exists==false) || (r4exists==false)) {
      if (r1exists==false) {
        r1l=Double.MIN_VALUE;
        r1u=Double.MAX_VALUE;
      }
      if (r2exists==false) {
        r2l=Double.MIN_VALUE;
        r2u=Double.MAX_VALUE;
      }
      if (r3exists==false) {
        r3l=Double.MIN_VALUE;
```

```
75          r3u=Double.MAX_VALUE;
76        }
77        if (r4exists==false) {
78          r4l=Double.MIN_VALUE;
79          r4u=Double.MAX_VALUE;
80        }
81      }
82      if ((r1exists==true)||(r2exists==true) ||(r3exists==true) ||(r4exists==true)) {
83        if ((r1l>=r2l) && (r1l>=r3l) && (r1l>=r4l)) {
84          sca.setLowerBound(r1l);
85        } else if ((r2l>=r1l) && (r2l>=r3l) && (r2l>=r4l)) {
86          sca.setLowerBound(r2l);
87        } else if ((r3l>=r1l) && (r3l>=r2l) && (r3l>=r4l)) {
88          sca.setLowerBound(r3l);
89        } else {
90          sca.setLowerBound(r4l);
91        }
92        if ((r1u<=r2u) && (r1u<=r3u) && (r1u<=r4u)){
93          sca.setUpperBound(r1u);
94        } else if ((r2u<=r1u) && (r2u<=r3u) && (r2u<=r4u)) {
95          sca.setUpperBound(r2u);
96        } else if ((r3u<=r1u) && (r3u<=r2u) && (r3u<=r4u)) {
97          sca.setUpperBound(r3u);
98        } else {
99          sca.setUpperBound(r4u);
100       }
101       result.setValue(i,sca);
102     } else {
103       result=null;
104     }
105   }
106   return result;
107 }
```

A.8 Solver quickily

Code A.8: Solver quickily

```
1  // Solver quickily als modifiziertes Verfahren
2  protected IvariVector quickily(IvariVector s) {
3    IvariMatrix matrix1 = new IvariMatrix(numberRows);
4    IvariMatrix matrix2 = new IvariMatrix(numberRows);
5    IvariMatrix matrix3 = new IvariMatrix(numberRows);
6    IvariVector result = new IvariVector(numberRows);
7    IvariVector result1 = new IvariVector(numberRows);
8    boolean r1exists=true;
9    IvariVector vector1 = new IvariVector(numberRows);
10   IvariVector vector2 = new IvariVector(numberRows);
11   IvariVector vector3 = new IvariVector(numberRows);
12   for (int i=0; i<result.getSize(); i++) {
13     vector1.setValue(i, s.getValue(i));
14     vector2.setValue(i, s.getValue(i));
15     vector3.setValue(i, s.getValue(i));
```

```
    for (int j=0; j<result.getSize(); j++) {
      matrix1.setValue(i,j,this.getValue(i,j));
      matrix2.setValue(i,j,this.getValue(i,j));
      matrix3.setValue(i,j,this.getValue(i,j));
    }
  }
  try {
    result1 = matrix1.gauss(Solver.siga, vector1);
  } catch (Exception e) {
    r1exists=false;
  }
  IvariVector r2 = new IvariVector(numberRows);
  boolean r2exists=true;
  try {
    r2 = matrix2.gauss(Solver.salt, vector2);
  } catch (Exception e) {
    r2exists=false;
  }
  //ning26
  IvariVector r3 = new IvariVector(numberRows);
  boolean r3exists=true;
  try {
    r3 = matrix3.solve(Solver.ning26, vector3);
  } catch (Exception e) {
    r3exists=false;
  }
  for (int i=0; i<result.getSize(); i++) {
    IvariScalar sca = new IvariScalar();
    Double r1l = result1.getValue(i).getLowerBound();
    Double r1u = result1.getValue(i).getUpperBound();
    Double r2l = r2.getValue(i).getLowerBound();
    Double r2u = r2.getValue(i).getUpperBound();
    Double r3l = r3.getValue(i).getLowerBound();
    Double r3u = r3.getValue(i).getUpperBound();
    if ((r1exists==false) || (r2exists==false)|| (r3exists==false)) {
      if (r1exists==false) {
        r1l=Double.MIN_VALUE;
        r1u=Double.MAX_VALUE;
      }
      if (r2exists==false) {
        r2l=Double.MIN_VALUE;
        r2u=Double.MAX_VALUE;
      }
      if (r3exists==false) {
        r3l=Double.MIN_VALUE;
        r3u=Double.MAX_VALUE;
      }
    }
    if ((r1exists==true)||(r2exists==true) ||(r3exists==true)) {
      if ((r1l>=r2l) && (r1l>=r3l)) {
        sca.setLowerBound(r1l);
      } else if ((r2l>=r1l) && (r2l>=r3l)) {
        sca.setLowerBound(r2l);
      } else {
```

```
70          sca.setLowerBound(r3l);
71        }
72        if ((r1u<=r2u) && (r1u<=r3u)){
73          sca.setUpperBound(r1u);
74        } else if ((r2u<=r1u) && (r2u<=r3u)) {
75          sca.setUpperBound(r2u);
76        } else {
77          sca.setUpperBound(r3u);
78        }
79        result.setValue(i,sca);
80      } else {
81        result=null;
82      }
83    }
84    return result;
85  }
```

A.9 Solver goody

Code A.9: Solver goody

```
1  // Solver goody aufgerufen durch adiva_rekursiv
2  protected IvariVector goody(IvariVector s) {
3    IvariVector r = new IvariVector(getSize());
4    int counter = countIntervals(matrix);
5    if (counter < 20) {
6      r = gaussPlu2(vector);
7    } else if (vector.getSize() < 7) {
8      r = picky(vector);
9    } else if (vector.getSize() < 80) {
10     r = quicky(vector);
11   } else {
12     r = quickily(vector);
13   }
14   return r;
15 }
```

A.10 Solver goodSolution

Code A.10: Solver goodSolution

```
1  // Solver goodSolution als modifiziertes Verfahren
2  protected IvariVector goodSolution(IvariVector s) {
3    IvariVector r = new IvariVector(getSize());
4    int counter = countIntervals(this);
5    if (counter < 20) {
6      r = gaussPlu2(s);
7    } else if (s.getSize() < 7) {
8      r = picky(s);
9    } else if(s.getSize() < 1300) {
10     r = adiva(s);
```

```
11    } else
12      try {
13        r = gauss(Solver.siga, s);
14      } catch (Exception e) {
15        e.printStackTrace();
16      }
17    return r;
18  }
```

A.11 Unit-Tests der Parameterstudien

Im folgenden Code (vgl. A.13) ist beispielhaft ein JUnit-Test am Produktsystem *testGP1* der Testreihe *3x3-linear* und für die Berechnung mit dem Solver *adivaOrigin* dargestellt.

Code A.11: JUnit-Test für Produktsysetm testGP1

```
1  package de.unistuttgart.iwb.ivaritest;
2  import static org.junit.jupiter.api.Assertions.*;
3  import org.junit.jupiter.api.Test;
4  import de.unistuttgart.iwb.ivari.*;
5  class TestHH01 {
6    @Test
7    public void testGP1() throws Exception {
8      IvariScalar a00 = new IvariScalar(1., 1.);
9      IvariScalar a01 = new IvariScalar(-1.05, -0.95);
10     IvariScalar a02 = new IvariScalar(0., 0.);
11     IvariScalar a10 = new IvariScalar(0., 0.);
12     IvariScalar a11 = new IvariScalar(1., 1.);
13     IvariScalar a12 = new IvariScalar(-1.1, -0.9);
14     IvariScalar a20 = new IvariScalar(0., 0.);
15     IvariScalar a21 = new IvariScalar(0., 0.);
16     IvariScalar a22 = new IvariScalar(1., 1.);
17     IvariScalar b0 = new IvariScalar(0., 0.);
18     IvariScalar b1 = new IvariScalar(0., 0.);
19     IvariScalar b2 = new IvariScalar(1., 1.);
20     IvariMatrix ma = new IvariMatrix(3);
21     IvariVector vb = new IvariVector(3);
22     IvariVector vx = new IvariVector(3);
23     ma.setValue(0, 0, a00);
24     ma.setValue(0, 1, a01);
25     ma.setValue(0, 2, a02);
26     ma.setValue(1, 0, a10);
27     ma.setValue(1, 1, a11);
28     ma.setValue(1, 2, a12);
29     ma.setValue(2, 0, a20);
30     ma.setValue(2, 1, a21);
31     ma.setValue(2, 2, a22);
32     vb.setValue(0, b0);
33     vb.setValue(1, b1);
34     vb.setValue(2, b2);
35     vx = ma.solve(Solver.adivaOrigin, vb);
36     for (int i=0; i<3; i++) {
37       System.out.println("x("+i+")␣lower␣bound␣=␣"+vx.getValue(i).getLowerBound()+"␣");
```

```
38        System.out.println("x("+i+") upper bound = "+vx.getValue(i).getUpperBound()+" ");
39      }
40      IvariMatrix inv = new IvariMatrix(3); IvariMatrix comp = new IvariMatrix(3); IvariMatrix inv2
            = new IvariMatrix(3);
41      inv = ma.inverse(ma); inv.matrizenDraw(inv); comp = ma.calcCompareMatrix_k(ma); inv2 = comp.
            inverse(comp); inv2.matrizenDraw(inv2);
42      assertTrue("inverse positive?", inv.getValue(0, 0).getLowerBound() >= 0.0);
43      assertTrue("inverse positive?", inv.getValue(0, 1).getLowerBound() >= 0.0);
44      assertTrue("inverse positive?", inv.getValue(0, 2).getLowerBound() >= 0.0);
45      assertTrue("inverse positive?", inv.getValue(1, 0).getLowerBound() >= 0.0);
46      assertTrue("inverse positive?", inv.getValue(1, 1).getLowerBound() >= 0.0);
47      assertTrue("inverse positive?", inv.getValue(1, 2).getLowerBound() >= 0.0);
48      assertTrue("inverse positive?", inv.getValue(2, 0).getLowerBound() >= 0.0);
49      assertTrue("inverse positive?", inv.getValue(2, 1).getLowerBound() >= 0.0);
50      assertTrue("inverse positive?", inv.getValue(2, 2).getLowerBound() >= 0.0);
51      if ((inv.getValue(0, 0).getLowerBound() >= 0.0) && (inv.getValue(0, 1).getLowerBound() >=
            0.0) && (inv.getValue(0, 2).getLowerBound() >= 0.0)
52          && (inv.getValue(1, 0).getLowerBound() >= 0.0) && (inv.getValue(1, 1).getLowerBound() >=
              0.0) && (inv.getValue(1, 2).getLowerBound() >= 0.0)
53          && (inv.getValue(2, 0).getLowerBound() >= 0.0) && (inv.getValue(2, 1).getLowerBound() >=
                0.0) && (inv.getValue(2, 2).getLowerBound() >= 0.0)) {
54        System.out.println("POS: " + 1);
55      }
56      if ((inv2.getValue(0, 0).getLowerBound() >= 0.0) && (inv2.getValue(0, 1).getLowerBound() >=
            0.0) && (inv2.getValue(0, 2).getLowerBound() >= 0.0)
57          && (inv2.getValue(1, 0).getLowerBound() >= 0.0) && (inv2.getValue(1, 1).getLowerBound()
              >= 0.0) && (inv2.getValue(1, 2).getLowerBound() >= 0.0)
58          && (inv2.getValue(2, 0).getLowerBound() >= 0.0) && (inv2.getValue(2, 1).getLowerBound()
              >= 0.0) && (inv2.getValue(2, 2).getLowerBound() >= 0.0)) {
59        System.out.println("H: " + 1);
60      }
61    }
62    ...
63  }
```

A.12 Unit-Tests für adisplit

Im folgenden Code ist beispielhaft ein JUnit-Test zur Erprobung der Methode *adisplit* dargestellt.

Code A.12: Unit-Tests zur Erprobung von adisplit

```
1  package de.unistuttgart.iwb.ivaritest;
2  import static org.junit.jupiter.api.Assertions.*;
3  import org.junit.jupiter.api.Test;
4  import de.unistuttgart.iwb.ivari.*;
5  @Test
6  public void testSplittedPS2() throws Exception {
7    System.out.println();
8    System.out.println("testSplittedPS2");
9    IvariScalar a00 = new IvariScalar(1., 1.);
10   IvariScalar a01 = new IvariScalar(0., 0.);
11   IvariScalar a02 = new IvariScalar(0., 0.);
12   IvariScalar a03 = new IvariScalar(-3., -3.);
13   IvariScalar a04 = new IvariScalar(0., 0.);
```

```
14   IvariScalar a05 = new IvariScalar(0., 0.);
15   IvariScalar a06 = new IvariScalar(0., 0.);
16   IvariScalar a07 = new IvariScalar(0., 0.);
17   IvariScalar a08 = new IvariScalar(0., 0.);
18   IvariScalar a09 = new IvariScalar(0., 0.);
19
20   IvariScalar a10 = new IvariScalar(0., 0.);
21   IvariScalar a11 = new IvariScalar(1., 1.);
22   IvariScalar a12 = new IvariScalar(0., 0.);
23   IvariScalar a13 = new IvariScalar(0., 0.);
24   IvariScalar a14 = new IvariScalar(0., 0.);
25   IvariScalar a15 = new IvariScalar(0., 0.);
26   IvariScalar a16 = new IvariScalar(0., 0.);
27   IvariScalar a17 = new IvariScalar(-2., -2.);
28   IvariScalar a18 = new IvariScalar(0., 0.);
29   IvariScalar a19 = new IvariScalar(0., 0.);
30
31   IvariScalar a20 = new IvariScalar(0., 0.);
32   IvariScalar a21 = new IvariScalar(0., 0.);
33   IvariScalar a22 = new IvariScalar(1., 1.);
34   IvariScalar a23 = new IvariScalar(0., 0.);
35   IvariScalar a24 = new IvariScalar(0., 0.);
36   IvariScalar a25 = new IvariScalar(0., 0.);
37   IvariScalar a26 = new IvariScalar(0., 0.);
38   IvariScalar a27 = new IvariScalar(0., 0.);
39   IvariScalar a28 = new IvariScalar(0., 0.);
40   IvariScalar a29 = new IvariScalar(0., 0.);
41
42   IvariScalar a30 = new IvariScalar(0., 0.);
43   IvariScalar a31 = new IvariScalar(0., 0.);
44   IvariScalar a32 = new IvariScalar(0., 0.);
45   IvariScalar a33 = new IvariScalar(1., 1.);
46   IvariScalar a34 = new IvariScalar(0., 0.);
47   IvariScalar a35 = new IvariScalar(0., 0.);
48   IvariScalar a36 = new IvariScalar(0., 0.);
49   IvariScalar a37 = new IvariScalar(0., 0.);
50   IvariScalar a38 = new IvariScalar(0., 0.);
51   IvariScalar a39 = new IvariScalar(-2., -2.);
52   IvariScalar a40 = new IvariScalar(0., 0.);
53   IvariScalar a41 = new IvariScalar(0., 0.);
54   IvariScalar a42 = new IvariScalar(-3., -3.);
55   IvariScalar a43 = new IvariScalar(0., 0.);
56   IvariScalar a44 = new IvariScalar(1., 1.);
57   IvariScalar a45 = new IvariScalar(0., 0.);
58   IvariScalar a46 = new IvariScalar(0., 0.);
59   IvariScalar a47 = new IvariScalar(0., 0.);
60   IvariScalar a48 = new IvariScalar(0., 0.);
61   IvariScalar a49 = new IvariScalar(0., 0.);
62
63   IvariScalar a50 = new IvariScalar(0., 0.);
64   IvariScalar a51 = new IvariScalar(0., 0.);
65   IvariScalar a52 = new IvariScalar(0., 0.);
66   IvariScalar a53 = new IvariScalar(0., 0.);
67   IvariScalar a54 = new IvariScalar(0., 0.);
```

```
68   IvariScalar a55 = new IvariScalar(1., 1.);
69   IvariScalar a56 = new IvariScalar(-2., -2.);
70   IvariScalar a57 = new IvariScalar(0., 0.);
71   IvariScalar a58 = new IvariScalar(0., 0.);
72   IvariScalar a59 = new IvariScalar(0., 0.);
73
74   IvariScalar a60 = new IvariScalar(0., 0.);
75   IvariScalar a61 = new IvariScalar(0., 0.);
76   IvariScalar a62 = new IvariScalar(0., 0.);
77   IvariScalar a63 = new IvariScalar(0., 0.);
78   IvariScalar a64 = new IvariScalar(0., 0.);
79   IvariScalar a65 = new IvariScalar(0., 0.);
80   IvariScalar a66 = new IvariScalar(1., 1.);
81   IvariScalar a67 = new IvariScalar(-4., -4.);
82   IvariScalar a68 = new IvariScalar(0., 0.);
83   IvariScalar a69 = new IvariScalar(0., 0.);
84
85   IvariScalar a70 = new IvariScalar(0., 0.);
86   IvariScalar a71 = new IvariScalar(0., 0.);
87   IvariScalar a72 = new IvariScalar(-2., -2.);
88   IvariScalar a73 = new IvariScalar(0., 0.);
89   IvariScalar a74 = new IvariScalar(0., 0.);
90   IvariScalar a75 = new IvariScalar(0., 0.);
91   IvariScalar a76 = new IvariScalar(0., 0.);
92   IvariScalar a77 = new IvariScalar(1., 1.);
93   IvariScalar a78 = new IvariScalar(0., 0.);
94   IvariScalar a79 = new IvariScalar(0., 0.);
95
96   IvariScalar a80 = new IvariScalar(0., 0.);
97   IvariScalar a81 = new IvariScalar(0., 0.);
98   IvariScalar a82 = new IvariScalar(0., 0.);
99   IvariScalar a83 = new IvariScalar(0., 0.);
100  IvariScalar a84 = new IvariScalar(-3., -3.);
101  IvariScalar a85 = new IvariScalar(0., 0.);
102  IvariScalar a86 = new IvariScalar(0., 0.);
103  IvariScalar a87 = new IvariScalar(0., 0.);
104  IvariScalar a88 = new IvariScalar(1., 1.);
105  IvariScalar a89 = new IvariScalar(0., 0.);
106
107  IvariScalar a90 = new IvariScalar(0., 0.);
108  IvariScalar a91 = new IvariScalar(0., 0.);
109  IvariScalar a92 = new IvariScalar(-1., -1.);
110  IvariScalar a93 = new IvariScalar(0., 0.);
111  IvariScalar a94 = new IvariScalar(0., 0.);
112  IvariScalar a95 = new IvariScalar(0., 0.);
113  IvariScalar a96 = new IvariScalar(0., 0.);
114  IvariScalar a97 = new IvariScalar(0., 0.);
115  IvariScalar a98 = new IvariScalar(0., 0.);
116  IvariScalar a99 = new IvariScalar(1., 1.);
117
118  IvariScalar b0 = new IvariScalar(0., 0.);
119  IvariScalar b1 = new IvariScalar(0., 0.);
120  IvariScalar b2 = new IvariScalar(1., 1.);
121  IvariScalar b3 = new IvariScalar(0., 0.);
```

```
122   IvariScalar b4 = new IvariScalar(0., 0.);
123   IvariScalar b5 = new IvariScalar(0., 0.);
124   IvariScalar b6 = new IvariScalar(0., 0.);
125   IvariScalar b7 = new IvariScalar(0., 0.);
126   IvariScalar b8 = new IvariScalar(0., 0.);
127   IvariScalar b9 = new IvariScalar(0., 0.);
128
129   IvariMatrix ma = new IvariMatrix(10);
130   IvariVector vb = new IvariVector(10);
131   IvariVector vx = new IvariVector(10);
132
133   ma.setValue(0, 0, a00);
134   ma.setValue(0, 1, a01);
135   ma.setValue(0, 2, a02);
136   ma.setValue(0, 3, a03);
137   ma.setValue(0, 4, a04);
138   ma.setValue(0, 5, a05);
139   ma.setValue(0, 6, a06);
140   ma.setValue(0, 7, a07);
141   ma.setValue(0, 8, a08);
142   ma.setValue(0, 9, a09);
143
144   ma.setValue(1, 0, a10);
145   ma.setValue(1, 1, a11);
146   ma.setValue(1, 2, a12);
147   ma.setValue(1, 3, a13);
148   ma.setValue(1, 4, a14);
149   ma.setValue(1, 5, a15);
150   ma.setValue(1, 6, a16);
151   ma.setValue(1, 7, a17);
152   ma.setValue(1, 8, a18);
153   ma.setValue(1, 9, a19);
154
155   ma.setValue(2, 0, a20);
156   ma.setValue(2, 1, a21);
157   ma.setValue(2, 2, a22);
158   ma.setValue(2, 3, a23);
159   ma.setValue(2, 4, a24);
160   ma.setValue(2, 5, a25);
161   ma.setValue(2, 6, a26);
162   ma.setValue(2, 7, a27);
163   ma.setValue(2, 8, a28);
164   ma.setValue(2, 9, a29);
165
166   ma.setValue(3, 0, a30);
167   ma.setValue(3, 1, a31);
168   ma.setValue(3, 2, a32);
169   ma.setValue(3, 3, a33);
170   ma.setValue(3, 4, a34);
171   ma.setValue(3, 5, a35);
172   ma.setValue(3, 6, a36);
173   ma.setValue(3, 7, a37);
174   ma.setValue(3, 8, a38);
175   ma.setValue(3, 9, a39);
```

```
176
177   ma.setValue(4, 0, a40);
178   ma.setValue(4, 1, a41);
179   ma.setValue(4, 2, a42);
180   ma.setValue(4, 3, a43);
181   ma.setValue(4, 4, a44);
182   ma.setValue(4, 5, a45);
183   ma.setValue(4, 6, a46);
184   ma.setValue(4, 7, a47);
185   ma.setValue(4, 8, a48);
186   ma.setValue(4, 9, a49);
187
188   ma.setValue(5, 0, a50);
189   ma.setValue(5, 1, a51);
190   ma.setValue(5, 2, a52);
191   ma.setValue(5, 3, a53);
192   ma.setValue(5, 4, a54);
193   ma.setValue(5, 5, a55);
194   ma.setValue(5, 6, a56);
195   ma.setValue(5, 7, a57);
196   ma.setValue(5, 8, a58);
197   ma.setValue(5, 9, a59);
198
199   ma.setValue(6, 0, a60);
200   ma.setValue(6, 1, a61);
201   ma.setValue(6, 2, a62);
202   ma.setValue(6, 3, a63);
203   ma.setValue(6, 4, a64);
204   ma.setValue(6, 5, a65);
205   ma.setValue(6, 6, a66);
206   ma.setValue(6, 7, a67);
207   ma.setValue(6, 8, a68);
208   ma.setValue(6, 9, a69);
209
210   ma.setValue(7, 0, a70);
211   ma.setValue(7, 1, a71);
212   ma.setValue(7, 2, a72);
213   ma.setValue(7, 3, a73);
214   ma.setValue(7, 4, a74);
215   ma.setValue(7, 5, a75);
216   ma.setValue(7, 6, a76);
217   ma.setValue(7, 7, a77);
218   ma.setValue(7, 8, a78);
219   ma.setValue(7, 9, a79);
220
221   ma.setValue(8, 0, a80);
222   ma.setValue(8, 1, a81);
223   ma.setValue(8, 2, a82);
224   ma.setValue(8, 3, a83);
225   ma.setValue(8, 4, a84);
226   ma.setValue(8, 5, a85);
227   ma.setValue(8, 6, a86);
228   ma.setValue(8, 7, a87);
229   ma.setValue(8, 8, a88);
```

```
230   ma.setValue(8, 9, a89);
231
232   ma.setValue(9, 0, a90);
233   ma.setValue(9, 1, a91);
234   ma.setValue(9, 2, a92);
235   ma.setValue(9, 3, a93);
236   ma.setValue(9, 4, a94);
237   ma.setValue(9, 5, a95);
238   ma.setValue(9, 6, a96);
239   ma.setValue(9, 7, a97);
240   ma.setValue(9, 8, a98);
241   ma.setValue(9, 9, a99);
242   vb.setValue(0, b0);
243   vb.setValue(1, b1);
244   vb.setValue(2, b2);
245   vb.setValue(3, b3);
246   vb.setValue(4, b4);
247   vb.setValue(5, b5);
248   vb.setValue(6, b6);
249   vb.setValue(7, b7);
250   vb.setValue(8, b8);
251   vb.setValue(9, b9);
252
253   vx = ma.solve(Solver.adisplit, vb);
254   for (int i=0; i<10; i++) {
255     System.out.println("x("+i+")␣lower␣bound␣=␣"+vx.getValue(i).getLowerBound()+"␣");
256     System.out.println("x("+i+")␣upper␣bound␣=␣"+vx.getValue(i).getUpperBound()+"␣");
257   }
258   assertEquals(vx.getValue(0).getLowerBound(), 6., .001);
259   assertEquals(vx.getValue(0).getUpperBound(), 6., .001);
260   assertEquals(vx.getValue(1).getLowerBound(), 4., .001);
261   assertEquals(vx.getValue(1).getUpperBound(), 4., .001);
262   assertEquals(vx.getValue(2).getLowerBound(), 1., .001);
263   assertEquals(vx.getValue(2).getUpperBound(), 1., .001);
264   assertEquals(vx.getValue(3).getLowerBound(), 2., .001);
265   assertEquals(vx.getValue(3).getUpperBound(), 2., .001);
266   assertEquals(vx.getValue(4).getLowerBound(), 3., .001);
267   assertEquals(vx.getValue(4).getUpperBound(), 3., .001);
268   assertEquals(vx.getValue(5).getLowerBound(), 16., .001);
269   assertEquals(vx.getValue(5).getUpperBound(), 16., .001);
270   assertEquals(vx.getValue(6).getLowerBound(), 8., .001);
271   assertEquals(vx.getValue(6).getUpperBound(), 8., .001);
272   assertEquals(vx.getValue(7).getLowerBound(), 2., .001);
273   assertEquals(vx.getValue(7).getUpperBound(), 2., .001);
274   assertEquals(vx.getValue(8).getLowerBound(), 9., .001);
275   assertEquals(vx.getValue(8).getUpperBound(), 9., .001);
276   assertEquals(vx.getValue(9).getLowerBound(), 1., .001);
277   assertEquals(vx.getValue(9).getUpperBound(), 1., .001);
278 }
```

A.13 Unit-Tests zur Erprobung des Rechenzeitaufwands

Im folgenden Code ist beispielhaft ein JUnit-Test zur Erprobung des Rechenzeitaufwands größerer Matrizen und dem Solver *quickily* dargestellt.

Code A.13: Unit-Test zur Erprobung des Rechenzeitaufwands großer Produktmatrizen

```
1 package de.unistuttgart.iwb.ivaritest;
2 import static org.junit.jupiter.api.Assertions.*;
3 import org.junit.jupiter.api.Test;
4 import de.unistuttgart.iwb.ivari.*;
5 @Test
6 public void testBigPS100v1() throws Exception {
7   IvariMatrix matrix = new IvariMatrix(100,100);
8   IvariScalar scalarOut = new IvariScalar(10.0, 10.0);
9   IvariScalar scalarIn = new IvariScalar(-10.01,-9.99);
10  IvariScalar scalarIn2 = new IvariScalar(-0.011,-0.009);
11  IvariScalar scalarZero = new IvariScalar(0,0);
12  for (int i=0; i < matrix.getSize(); i++) {
13    for (int j=0; j < matrix.getSize(); j++) {
14      if (i==j) {
15        matrix.setValue(i, j, scalarOut);
16      } else if (j==(matrix.getSize()-1)) {
17        matrix.setValue(i, j, scalarIn);
18      } else {
19        matrix.setValue(i, j, scalarIn2);
20      }
21    }
22  }
23  IvariVector vector = new IvariVector(100);
24  for (int i=0; i < vector.getSize(); i++) {
25    if (i!=(vector.getSize()-1)) {
26      vector.setValue(i, scalarZero);
27    } else {
28      vector.setValue(i, scalarOut);
29    }
30  }
31  IvariVector vx= new IvariVector(100);
32  vx = matrix.solve(Solver.quickily, vector);
33  for (int i=0; i<100; i++) {
34    System.out.println("x("+i+") lower bound = "+vx.getValue(i).getLowerBound()+" ");
35    System.out.println("x("+i+") upper bound = "+vx.getValue(i).getUpperBound()+" ");
36  }
37 }
```

Anhang B

Ergänzende Informationen

B.1 Produktsysteme der Testreihe 3x3-linear

Tabelle B.1: Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe *3x3-linear*

Produktsystem	**Parameter**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP1	$a_0 b_0 c_0 d_0$	M, H, Z	Lineares PS	keine
testGP2	$a_0 b_0 c_0 d_1$	M, H, Z	Schleife durch Verzweigung	keine
testGP3	$a_0 b_0 c_0 d_2$	H	Pseudoschleife	keine
testGP4	$a_0 b_0 c_1 d_0$	M, H, Z	Schleife durch Verzweigung	keine
testGP5	$a_0 b_0 c_1 d_1$	M, H, Z	Schleife durch Verzweigung	keine
testGP6	$a_0 b_0 c_1 d_2$	H	Mischstruktur	keine
testGP7	$a_0 b_0 c_2 d_0$	H	Pseudoschleife	keine
testGP8	$a_0 b_0 c_2 d_1$	H	Mischstruktur	keine
testGP9	$a_0 b_0 c_2 d_2$	H	Pseudoschleife	keine
testGP10	$a_0 b_1 c_0 d_0$	M, H, Z	Schleife (Verzweigung)	keine
testGP11	$a_0 b_1 c_0 d_1$	M, H, Z	Schleife (Verzweigung)	keine
testGP12	$a_0 b_1 c_0 d_2$	H	Mischstruktur	keine
testGP13	$a_0 b_1 c_1 d_0$	M, H, Z	Schleife (Verzweigung)	keine
testGP14	$a_0 b_1 c_1 d_1$	M, H, Z	Schleife (Verzweigung)	hansen
testGP15	$a_0 b_1 c_1 d_2$	H	Mischstruktur	keine

Fortsetzung auf der nächsten Seite

Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe 3x3-linear

Produktsystem	**Variablen**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP16	$a_0 b_1 c_2 d_0$	H	Mischstruktur	keine
testGP17	$a_0 b_1 c_2 d_1$	H	Mischstruktur	keine
testGP18	$a_0 b_1 c_2 d_2$	H	Mischstruktur	keine
testGP19	$a_0 b_2 c_0 d_0$	H	Pseudoschleife	keine
testGP20	$a_0 b_2 c_0 d_1$	H	Mischstruktur	keine
testGP21	$a_0 b_2 c_0 d_2$	H	Pseudoschleife	keine
testGP22	$a_0 b_2 c_1 d_0$	H	Mischstruktur	keine
testGP23	$a_0 b_2 c_1 d_1$	H	Mischstruktur	salt
testGP24	$a_0 b_2 c_1 d_2$	H	Mischstruktur	keine
testGP25	$a_0 b_2 c_2 d_0$	H	Pseudoschleife	keine
testGP26	$a_0 b_2 c_2 d_1$	H	Mischstruktur	keine
testGP27	$a_0 b_2 c_2 d_2$	H	Pseudoschleife	keine
testGP28	$a_1 b_0 c_0 d_0$	M, H, Z	Pseudobaum	keine
testGP29	$a_1 b_0 c_0 d_1$	M, H, Z	Mischstruktur	keine
testGP30	$a_1 b_0 c_0 d_2$	H	Mischstruktur	keine
testGP31	$a_1 b_0 c_1 d_0$	H, H, Z	Mischstruktur	keine
testGP32	$a_1 b_0 c_1 d_1$	M, H, Z	Mischstruktur	keine
testGP33	$a_1 b_0 c_1 d_2$	H	Mischstruktur	keine
testGP34	$a_1 b_0 c_2 d_0$	H	Mischstruktur	keine
testGP35	$a_1 b_0 c_2 d_1$	H	Mischstruktur	keine
testGP36	$a_1 b_0 c_2 d_2$	H	Mischstruktur	keine
testGP37	$a_1 b_1 c_0 d_0$	M, H, Z	Mischstruktur	keine
testGP38	$a_1 b_1 c_0 d_1$	M, H, Z	Mischstruktur	keine
testGP39	$a_1 b_1 c_0 d_2$	H	Mischstruktur	keine
testGP40	$a_1 b_1 c_1 d_0$	H, H, Z	Mischstruktur	keine
testGP41	$a_1 b_1 c_1 d_1$	M, Z	Mischstruktur	alle

Fortsetzung auf der nächsten Seite

Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe 3x3-linear

Produktsystem	**Variablen**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP42	$a_1\,b_1\,c_1\,d_2$	keine	Mischstruktur	keine
testGP43	$a_1\,b_1\,c_2\,d_0$	H	Mischstruktur	keine
testGP44	$a_1\,b_1\,c_2\,d_1$	keine	Mischstruktur	keine
testGP45	$a_1\,b_1\,c_2\,d_2$	keine	Mischstruktur	keine
testGP46	$a_1\,b_2\,c_0\,d_0$	H	Mischstruktur	keine
testGP47	$a_1\,b_2\,c_0\,d_1$	H	Mischstruktur	salt
testGP48	$a_1\,b_2\,c_0\,d_2$	H	Mischstruktur	keine
testGP49	$a_1\,b_2\,c_1\,d_0$	H	Mischstruktur	salt
testGP50	$a_1\,b_2\,c_1\,d_1$	keine	Mischstruktur	salt
testGP51	$a_1\,b_2\,c_1\,d_2$	keine	Mischstruktur	keine
testGP52	$a_1\,b_2\,c_2\,d_0$	keine	Mischstruktur	salt
testGP53	$a_1\,b_2\,c_2\,d_1$	keine	Mischstruktur	salt
testGP54	$a_1\,b_2\,c_2\,d_2$	keine	Mischstruktur	keine
testGP55	$a_2\,b_0\,c_0\,d_0$	H	Schleife durch Zusammenfluss	keine
testGP56	$a_2\,b_0\,c_0\,d_1$	H, inv.-pos.	Schleife	keine
testGP57	$a_2\,b_0\,c_0\,d_2$	H, inv.-pos.	Mischstruktur	keine
testGP58	$a_2\,b_0\,c_1\,d_0$	H	Schleife	keine
testGP59	$a_2\,b_0\,c_1\,d_1$	H, inv.-pos.	Schleife	keine
testGP60	$a_2\,b_0\,c_1\,d_2$	H, inv.-pos.	Mischstruktur	keine
testGP61	$a_2\,b_0\,c_2\,d_0$	H	Mischstruktur	keine
testGP62	$a_2\,b_0\,c_2\,d_1$	H	Mischstruktur	keine
testGP63	$a_2\,b_0\,c_2\,d_2$	H	Mischstruktur	keine
testGP64	$a_2\,b_1\,c_0\,d_0$	H, inv.-pos.	Schleife	keine
testGP65	$a_2\,b_1\,c_0\,d_1$	H, inv.-pos.	Schleife	keine
testGP66	$a_2\,b_1\,c_0\,d_2$	H	Mischstruktur	keine
testGP67	$a_2\,b_1\,c_1\,d_0$	H, inv.-pos.	Schleife	keine

Fortsetzung auf der nächsten Seite

Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe 3x3-linear

Produktsystem	**Variablen**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP68	$a_2 b_1 c_1 d_1$	keine, inv.-pos.	Schleife	siga, salt, hansen
testGP69	$a_2 b_1 c_1 d_2$	keine	Mischstruktur	keine
testGP70	$a_2 b_1 c_2 d_0$	keine	Mischstruktur	keine
testGP71	$a_2 b_1 c_2 d_1$	H	Mischstruktur	keine
testGP72	$a_2 b_1 c_2 d_2$	keine	Mischstruktur	keine
testGP73	$a_2 b_2 c_0 d_0$	keine	Mischstruktur	keine
testGP74	$a_2 b_2 c_0 d_1$	H	Mischstruktur	keine
testGP75	$a_2 b_2 c_0 d_2$	H	Mischstruktur	keine
testGP76	$a_2 b_2 c_1 d_0$	H	Mischstruktur	keine
testGP77	$a_2 b_2 c_1 d_1$	H	Mischstruktur	keine
testGP78	$a_2 b_2 c_1 d_2$	keine	Mischstruktur	keine
testGP79	$a_2 b_2 c_2 d_0$	keine	Mischstruktur	keine
testGP80	$a_2 b_2 c_2 d_1$	H	Mischstruktur	keine
testGP81	$a_2 b_2 c_2 d_2$	keine	Mischstruktur	keine

B.2 Produktsysteme der Testreihe 3x3-baum

Tabelle B.2: Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe *3x3-baum*

Produktsystem	**Variablen**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP1b	$a_0 b_0 c_0 d_0$	M, H, Z	Gegenströmiger Baum	keine
testGP2b	$a_0 b_0 c_0 d_1$	M, H, Z	Schleife (Verzweigung)	keine
testGP3b	$a_0 b_0 c_0 d_2$	H	Pseudoschleife	keine
testGP4b	$a_0 b_0 c_1 d_0$	M, H, Z	Schleife (Verzweigung)	keine
testGP5b	$a_0 b_0 c_1 d_1$	M, H, Z	Schleife (Verzweigung)	keine
testGP6b	$a_0 b_0 c_1 d_2$	H	Mischstruktur	keine
testGP7b	$a_0 b_0 c_2 d_0$	H	Pseudoschleife	keine

Fortsetzung auf der nächsten Seite

Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe 3x3-baum

Produktsystem	**Variablen**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP8b	$a_0 b_0 c_2 d_1$	H	Mischstruktur	keine
testGP9b	$a_0 b_0 c_2 d_2$	H	Pseudoschleife	keine
testGP10b	$a_0 b_1 c_0 d_0$	M, H, Z	Schleife (Verzweigung)	keine
testGP11b	$a_0 b_1 c_0 d_1$	M, H, Z	Schleife (Verzweigung)	keine
testGP12b	$a_0 b_1 c_0 d_2$	H	Mischstruktur	keine
testGP13b	$a_0 b_1 c_1 d_0$	M, H, Z	Schleife (Verzweigung)	keine
testGP14b	$a_0 b_1 c_1 d_1$	M, H, Z	Schleife (Verzweigung)	keine
testGP15b	$a_0 b_1 c_1 d_2$	H	Mischstruktur	keine
testGP16b	$a_0 b_1 c_2 d_0$	H	Mischstruktur	keine
testGP17b	$a_0 b_1 c_2 d_1$	H	Mischstruktur	keine
testGP18b	$a_0 b_1 c_2 d_2$	H	Mischstruktur	keine
testGP19b	$a_0 b_2 c_0 d_0$	H	Pseudoschleife	keine
testGP20b	$a_0 b_2 c_0 d_1$	H	Mischstruktur	keine
testGP21b	$a_0 b_2 c_0 d_2$	H	Pseudoschleife	keine
testGP22b	$a_0 b_2 c_1 d_0$	H	Mischstruktur	keine
testGP23b	$a_0 b_2 c_1 d_1$	H	Mischstruktur	keine
testGP24b	$a_0 b_2 c_1 d_2$	H	Mischstruktur	keine
testGP25b	$a_0 b_2 c_2 d_0$	H	Pseudoschleife	keine
testGP26b	$a_0 b_2 c_2 d_1$	H	Mischstruktur	keine
testGP27b	$a_0 b_2 c_2 d_2$	H	Pseudoschleife	keine
testGP28b	$a_1 b_0 c_0 d_0$	M, H, Z	Pseudobaum	keine
testGP29b	$a_1 b_0 c_0 d_1$	M, H, Z	Mischstruktur	keine
testGP30b	$a_1 b_0 c_0 d_2$	H	Pseudobaum und -schleife	keine
testGP31b	$a_1 b_0 c_1 d_0$	M, H, Z	Mischstruktur	keine
testGP32b	$a_1 b_0 c_1 d_1$	M, H, Z	Mischstruktur	keine
testGP33b	$a_1 b_0 c_1 d_2$	H	Mischstruktur	keine

Fortsetzung auf der nächsten Seite

Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe 3x3-baum

Produktsystem	**Variablen**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP34b	$a_1 b_0 c_2 d_0$	H	Pseudobaum und -schleife	keine
testGP35b	$a_1 b_0 c_2 d_1$	H	Mischstruktur	keine
testGP36b	$a_1 b_0 c_2 d_2$	H	Pseudobaum und -schleife	keine
testGP37b	$a_1 b_1 c_0 d_0$	M, H, Z	Mischstruktur	keine
testGP38b	$a_1 b_1 c_0 d_1$	M, H, Z	Mischstruktur	keine
testGP39b	$a_1 b_1 c_0 d_2$	H	Mischstruktur	keine
testGP40b	$a_1 b_1 c_1 d_0$	M, H, Z	Mischstruktur	keine
testGP41b	$a_1 b_1 c_1 d_1$	M, H, Z	Mischstruktur	hansen
testGP42b	$a_1 b_1 c_1 d_2$	H	Mischstruktur	keine
testGP43b	$a_1 b_1 c_2 d_0$	H	Mischstruktur	keine
testGP44b	$a_1 b_1 c_2 d_1$	H	Mischstruktur	keine
testGP45b	$a_1 b_1 c_2 d_2$	H	Mischstruktur	keine
testGP46b	$a_1 b_2 c_0 d_0$	H	Pseudobaum und -schleife	keine
testGP47b	$a_1 b_2 c_0 d_1$	H	Mischstruktur	salt
testGP48b	$a_1 b_2 c_0 d_2$	H	Pseudobaum und -schleife	salt
testGP49b	$a_1 b_2 c_1 d_0$	H	Mischstruktur	salt
testGP50b	$a_1 b_2 c_1 d_1$	H	Mischstruktur	salt
testGP51b	$a_1 b_2 c_1 d_2$	H	Mischstruktur	salt
testGP52b	$a_1 b_2 c_2 d_0$	H	Pseudobaum und -schleife	keine
testGP53b	$a_1 b_2 c_2 d_1$	H	Mischstruktur	salt
testGP54b	$a_1 b_2 c_2 d_2$	H	Mischstruktur	salt
testGP55b	$a_2 b_0 c_0 d_0$	H	Pseudoschleife	keine
testGP56b	$a_2 b_0 c_0 d_1$	H	Mischstruktur	keine
testGP57b	$a_2 b_0 c_0 d_2$	H	Pseudoschleife	keine
testGP58b	$a_2 b_0 c_1 d_0$	H	Mischstruktur	keine
testGP59b	$a_2 b_0 c_1 d_1$	H	Mischstruktur	keine

Fortsetzung auf der nächsten Seite

Matrixtyp, Produktsystemstruktur und Lösbarkeit der Testreihe 3x3-baum

Produktsystem	**Variablen**[1]	**Matrixtyp**[2]	**Systemstruktur**	**Exceptions**[3]
testGP60b	$a_2 b_0 c_1 d_2$	H	Mischstruktur	keine
testGP61b	$a_2 b_0 c_2 d_0$	H	Pseudoschleife	keine
testGP62b	$a_2 b_0 c_2 d_1$	H	Mischstruktur	keine
testGP63b	$a_2 b_0 c_2 d_2$	H	Pseudoschleife	keine
testGP64b	$a_2 b_1 c_0 d_0$	H	Mischstruktur	keine
testGP65b	$a_2 b_1 c_0 d_1$	H	Mischstruktur	keine
testGP66b	$a_2 b_1 c_0 d_2$	H	Mischstruktur	keine
testGP67b	$a_2 b_1 c_1 d_0$	H	Mischstruktur	keine
testGP68b	$a_2 b_1 c_1 d_1$	H	Mischstruktur	siga
testGP69b	$a_2 b_1 c_1 d_2$	H	Mischstruktur	keine
testGP70b	$a_2 b_1 c_2 d_0$	H	Mischstruktur	keine
testGP71b	$a_2 b_1 c_2 d_1$	H	Mischstruktur	keine
testGP72b	$a_2 b_1 c_2 d_2$	H	Mischstruktur	keine
testGP73b	$a_2 b_2 c_0 d_0$	H	Pseudoschleife	keine
testGP74b	$a_2 b_2 c_0 d_1$	H	Mischstruktur	keine
testGP75b	$a_2 b_2 c_0 d_2$	H	Pseudoschleife	keine
testGP76b	$a_2 b_2 c_1 d_0$	H	Mischstruktur	keine
testGP77b	$a_2 b_2 c_1 d_1$	H	Mischstruktur	keine
testGP78b	$a_2 b_2 c_1 d_2$	H	Mischstruktur	keine
testGP79b	$a_2 b_2 c_2 d_0$	H	Pseudoschleife	keine
testGP80b	$a_2 b_2 c_2 d_1$	H	Mischstruktur	keine
testGP81b	$a_2 b_2 c_2 d_2$	H	Pseudoschleife	keine

[1] Die Parameter sind in Tabelle 8.4 des Kapitels 8.6.3 definiert.

[2] Es werden M-, H-, oder Z-Matrizen berücksichtigt.

[3] Es werden die Solver *hansen, siga, adivaOrigin, adiheu* und *adiperm* berücksichtigt.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit mit dem Titel *Entwicklung von intervallarithmetischen Methoden zur Berücksichtigung von Unsicherheiten in der Ökobilanzierung* selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe; aus fremden Quellen entnommene Passagen und Gedanken sind als solche kenntlich gemacht.

Stuttgart, Juni 2022

Helen Hein