Initialization and Reformulation Strategies for Improved Solving of Nonlinear Algebraic Equation Systems Tested on Chemical Process Models

vorgelegt von
M. Sc.
Saskia Bublitz

an der Fakultät III – Prozesswissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktorin der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Thomas William Brown
Gutachter:	Prof. Dr.-Ing. habil. Jens-Uwe Repke
Gutachter:	Prof. Dr. Michael Bortz

Tag der wissenschaftlichen Aussprache: 27. Oktober 2023

Berlin 2023

To my Family

Affidavit

- ✓ Die Arbeit enthält eine deutsche Zusammenfassung.
- ✓ Die Arbeit enthält eine englische Zusammenfassung.
- ✓ Die Arbeit enthält eine Selbständigkeitserklärung nach § 60 Abs. 8 AllgStuPO:

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

M. Sc.
Saskia Bublitz
Berlin, November 22, 2023

- ✓ The thesis contains a German abstract.
- ✓ The thesis contains an English abstract.
- ✓ The thesis includes a declaration that I completed the work independently in accordance with section § 60 Abs. 8 AllgStuPO:

I hereby confirm that I prepared this thesis independently and by exclusive reliance on literature or tools indicated herein.

M. Sc.
Saskia Bublitz
Berlin, November 22, 2023

Acknowledgements

Nicht nur die Hard- und Software sind wichtig um Problemstellungen zu lösen, sondern auch das Input von Personen ohne die es nicht möglich gewesen wäre, die Arbeit in ihrem jetzigen Umfang zu vollenden. Bei diesen Personen möchte ich mich recht herzlich bedanken. Zunächst gilt mein Dank Sandra Fillinger, die mir überhaupt erst den Weg einer Promotion aufgezeigt hat und Herrn Repke, der mir diesen Weg ermöglicht hat. Angekommen am Fachgebiet möchte ich mich vorallem bei Erik Esche bedanken für seine zahlreichen Gespräche und Anregungen fachlicher Natur, aber auch Geduld und Hilfsbereitschaft, wenn Software-Pakete nicht so funktionierten, wie sie es theoretisch tun sollten. Ohne Gregor Tolksdorf würde MOSAICmodeling nicht so reibungslos laufen und umfangreich ausgestattet sein, dafür danke ich ihm, da sonst längst nicht so viele Prozessmodelle und Lösungsstrategien hätten entwickelt und getestet werden können. Danke auch an das aktuelle Entwicklungsteam von MOSAICmodeling Marlos Sotto Maior, Alexander Jung und Erik Esche mit denen die Zusammenarbeit und die Workshops stets eine Freude waren. Besonderem Dank gilt Ali Baharev, der mir die Intervall-Arithmetik überhaupt erst näher gebracht hat und dessen kritisches Feedback stets sehr hilfreich war. Das Erstellen von Prozessmodellen, die Unterstützung bei der Weiterentwicklung der Algorithmen und schließlich das Testen des Hybridverfahrens habe ich vorallem Sangitha Rajes, Fabian Ebert, und Angelina Reum zu verdanken, die mich im Rahmen ihrer Bachelor- und Masterarbeiten tatkräftig unterstützt haben. Allgemein habe ich mich sehr wohl am Fachgebiet dbta gefühlt und danke dem gesamten Team für unterhaltsame Klausurkontrollabende und Anlagenfahrten, spannende Kickertuniere sowie dynamische Wandertage. Henning Reinken, Charlotte Kugler, Alicia Niedzwiecka und Julien Fuchs danke ich für unsere produktiven und fachübergreifenden Schreibsonntage im letzten halben Jahr. Zu guter Letzt möchte ich mich noch bei meiner Familie und meinen Freunden dafür bedanken, dass sie

immer für mich da sind, an mich glauben und meinen Alltag um viele tolle Erlebnisse bereichern, die auch für den Ausgleich in der Promotionsphase für mich sehr wichtig waren.

Abstract

Deutsch

Das Lösen nichtlinear algebraischer Gleichungssysteme mittels numerischer Verfahren stellt in der chemischen Verfahrenstechnik häufig eine zeitintensive Aufgabe dar, insbesondere wenn Systeme schlecht konditioniert sind und / oder keine gut geschätzten Startwerte für den numerischen Löser vorhanden sind. In dieser Arbeit wird ein Hybridverfahren entwickelt, um solche Systeme unabhgängig von einer aufwendigen Schätzung der Startwerte lösen zu können. Das Hybridverfahren macht sich Methoden aus der Intervallarithmetik zu Nutze, um nicht realisierbare Wertebereiche der gesuchten Variablen auszuschließen und im übrigen Bereich Lösungen effizient über Newton-basierte Methoden zu lokalisieren. Der Anwender muss lediglich vorab Wertebereiche der gesuchten Variablen definieren. Zur Gewährleistung der Unabhängigkeit des Verfahrens von einer bestimmten Newton-basierten Methode, werden verschiedenste dieser Art eingesetzt, nämlich: ein selbstimplementiertes Newton-Verfahren, Scipys SLSQP und Fsolve sowie Ipopt. Das Hybridverfahren wird in Python implementiert und an verfahrenstechnischen Beispielen getestet. Diese Systeme sind allesamt komplex, unterscheiden sich aber in ihrer Dimension, Kondition und Nichtlinearität. Es wird für alle Beispiele mindestens eine physikalisch realisierbare Lösung in wenigen Minuten gefunden. Für einige Systeme können sogar alle Lösungen im unbeschränkten Variablenraum ausfindig gemacht werden. Die Intervallarithmetik bietet hier die Möglichkeit mathematisch zu beweisen, dass es keine weiteren Lösungen geben kann. Dies ist theoretisch auch für alle anderen Testbeispiele möglich, allerdings benötigt in den größeren Systemen die Intervallarithmetikbasierte Reduktion des Lösungsraumes zu viele Reduktionsschritte, um in die Nähe der reellen Lösung(en) zu kommen. Die Effektivität der Reduktion von

Variablengrenzen ist besonders stark von der Initialisierung dieser und der Formulierung der Gleichungen abhängig. Im Rahmen der Arbeit wurden verschiedene Initialisierungen und Formulierungen der Gleichungen untersucht und die wichtigsten Erkenntnisse in Leitfäden zusammengefasst. Außerdem wurde ein erste Kategorisierung der untersuchten Gleichungssysteme gemessen an deren Komplexität vorgenommen. Auf dieser Basis kann abgeschätzt werden, welche der drei Lösungsstrategien (intervallarithmetisches Verfahren, Newton-basiertes Verfahren oder Hybridverfahren) im individuellen Fall am geeignetsten ist. Die problemunabhängige Anwendbarkeit des Hybridverfahrens sollte an weiteren großen, komplexen, nichtlinear algebraischen Prozessmodellen überprüft werden. Viele Schritte innerhalb des Verfahrens bieten die Möglichkeit parallel durchgeführt zu werden und könnten erheblich zu dessen Beschleunigung beitragen. Somit könnte der Ansatz auch zum Lösen von Optimierungsproblemen oder diskretisierten, differential algebraischen Systemen interessant werden.

Schlüsselwörter: *Initialisierung nichtlinear algebraischer Gleichungssysteme; Hybridverfahren; Intervalarithmetik; Newton-basierte Verfahren; Reformulierung von Gleichungen; numerische Lösungsverfahren; Komplexität von Prozessmodellen; Konvergenz*

English

Solving nonlinear algebraic systems of equations by numerical methods is often a time-consuming challenge in chemical engineering, especially when systems are ill-conditioned and/or no well estimated initial values for the numerical solver are available. In this work, a hybrid method is developed to solve such systems independently of an insufficient initialization. The hybrid method makes use of methods from interval arithmetic to exclude infeasible ranges of values of the unkown variables and to efficiently locate solutions in the remaining feasible region by Newton-based methods. The user only has to set the bounds of the unknown variables in advance. To ensure the independence of the approach from Newton-based methods, several of these are applied, namely: A self implemented Newton method, Scipy's SLSQP and Fsolve as well as Ipopt. The hybrid method is implemented in Python and tested on process engineering examples. These systems are all complex, but differ in dimension, condition, and nonlinearity. For

all systems at least one physically feasible solution is found in a few minutes. All solutions in the unrestricted variable space can even be found for some systems. The interval arithmetic offers here the possibility to prove mathematically that there can be no further solutions. This is theoretically possible for all other test examples as well, but in the larger systems the interval arithmetic based reduction requires too many box reduction steps to get close to the real-valued solution(s). The effectiveness of the reduction of variable bounds is particularly dependent on the initialization of these and the formulation of the equations. As part of this work, a wide variety of initializations and formulations of the equations were examined and the most important findings were collected in the form of guidelines. Furthermore, a first classification of the investigated systems of equations was carried out, measured by their complexity. Based on this, it can be estimated which of the three solution strategies (interval arithmetic method, Newton-based method or hybrid approach) is most suitable in the individual case. The problem-independent applicability of the hybrid approach should be verified on further large, complex, nonlinear algebraic process models. Many steps within the procedure offer the possibility to be performed in parallel and could contribute significantly to its acceleration. Thus, the approach could also become interesting for solving optimization problems or discretized, differential algebraic equation systems.

Keywords: *Initialization of Nonlinear Algebraic Equation Systems; Hybrid Approach; Interval Arithmetic; Newton based Solvers; Reformulation of Equations; Numeric Solvers; Complexity of Process Models; Convergence*

Publications

This thesis is partially based on already published contributions. In the following, these are divided into journal articles, papers within conference proceedings, oral presentations without papers, and a list of all supervised theses. They are sorted chronologically.

Journal Articles

- S. Bublitz, E. Esche, G. Tolksdorf, V. Mehrmann, J.-U. Repke (2017a): Analysis and Decomposition for Improved Convergence of Nonlinear Process Models in Chemical Engineering. *Chemie Ingenieur Technik* 89 (11), 1503–1514. ISSN: 0009286X. DOI: 10.1002/cite.201700041

- S. Bublitz, E. Esche, J.-U. Repke (2021b): Automatic Initial Value Generation with Interval Arithmetic for Nonlinear Process Models. *Computers & Chemical Engineering* 151, 107342. DOI: 10.1016/j.compchemeng.2021.107342

- S. Bublitz, E. Esche, J.-U. Repke (2023 submitted): Interval Arithmetic based Cutting and Splitting in an Automatic Initial Value Generation Procedure for Nonlinear Process Models

Conference Papers

- S. Bublitz, E. Esche, G. Tolksdorf, J.-U. Repke (2018): Improving Convergence Behavior of Nonlinear Equation Systems in Intensified Process Models by Decomposition Methods. In: *28th European Symposium on Computer Aided*

Process Engineering. Ed. by S. Radl. Vol. 43. Computer Aided Chemical Engineering. Graz: Elsevier, 403–408. ISBN: 9780444642356. DOI: 10.1016/B978-0-444-64235-6.50073-5

– E. Esche, S. Bublitz, G. Tolksdorf, J.-U. Repke (2018): Automatic Decomposition of Nonlinear Equation Systems for Improved Initialization and Solution of Chemical Engineering Process Models. *13th International Symposium on Process Systems Engineering*. Vol. 44. Computer Aided Chemical Engineering. San Diego: Elsevier, 1387–1392. ISBN: 9780444642417. DOI: 10.1016/B978-0-444-64241-7.50226-3

– S. Bublitz, E. Esche, J.-U. Repke (2020): Investigation of a Hybrid Approach to Find all Solutions of Nonlinear Equation Systems. In: *30th European Symposium on Computer Aided Process Engineering*. Vol. 48. Computer Aided Chemical Engineering. Elsevier, 481–486. ISBN: 9780128233771. DOI: 10.1016/B978-0-12-823377-1.50081-1

Oral Presentations and Posters Without Proceedings

– S. Bublitz, E. Esche, G. Tolksdorf, J.-U. Repke (2017b): Zerlegung nichtlinearer Prozessmodelle zur verbesserten numerischen Konvergenz. Jahrestreffen der ProcessNet-Fachgemeinschaft "Prozess-, Apparate- und Anlagentechnik". Würzburg

– V. Kozachynsky, S. Bublitz, M. Illner, J. Weigert, C. Hoffmann, E. Esche, J.-U. Repke (2018): Conceptual data model based on an OPC UA architecture, its benefits and implementation. Jahrestreffen der ProcessNet-Fachgemeinschaft "Prozess-, Apparate- und Anlagentechnik". Köln

– S. Bublitz, E. Esche, M. Sotto Maior, J.-U. Repke (2019): MOSAICmodeling - A Fully Equation-oriented, Web-basd Tool for Modeling, Simulation and Optimization in Chemcial Engineering. deRSE 2019 - Konferenz für ForschungssoftwareenticklerInnen in Deutschland. Potsdam. DOI: 10.5446/42511

– V. Kozachynsky, S. Bublitz, M. Illner, J. Weigert, C. Hoffmann, E. Esche, J.-U. Repke (2019): OPC UA-based Concept for Online Implementation of Model-based Advanced Process Control Tools. 12th European Congress of Chemical Engineering and 5th European Congress of applied Biotechnology. Florence

– S. Bublitz, E. Esche, J.-U. Repke (2021a): Application of Automatic Interval-Arithmetic Based Initialization Algorithm to Find Solutions of Nonlinear Algebraic Process Models. Jahrestreffen der ProcessNet-Fachgemeinschaft "Prozess-, Apparate- und Anlagentechnik"

Supervised Theses

– A. Maas (2018): Informationsextraktion aus Rohrleitungs- und Instrumentenfließbildern mithilfe künstlicher Intelligenz. *Master Thesis*. Berlin: Technische Universität Berlin

– P. F. Reixach (2018): Laboratory setup for the evaluation of COD removal in the metalworking wastewater treatment. *Master Thesis*. Barcelona: Universitat Politecnica de Catalunya

– K. Kaminska (2018): Weiterführende Untersuchungen zur verbesserten Entspannungsverdampfung von organischen Lösemittel und niederviskosen Polymerlösungen. *Master Thesis*. Berlin: Technische Universität Berlin

– D. E. Chaparro Bernal (2019): Model development of the oligomerization reaction kinetics of the Budimat plant located in Ludwigshafen am Rhein. *Master Thesis*. Berlin: Technische Universität Berlin

– S. Rajes (2020): Modellierung und Modellvalidierung des Hydroalkylierungsprozesses. *Master Thesis*. Berlin: Technische Universität Berlin

– M. Schönhof (2020): Datenkonsistenz in der Anlagenplanung mithilfe von Datenbankstrukturen zur ganzheitlichen technischen Dokumentation einer Anlage. *Master Thesis*. Berlin: Technische Universität Berlin

– A. D. Sobczak (2020): General Solution of Nonlinear System in Chemical Engineering. *Master Thesis*. Wroclaw: Politechnika Wroclawska

– F. Ebert (2021): Untersuchung verschiedener Intervall-Arithmetik basierter Methoden zur Reduktion des Lösungsraumes von nichtlinear algebraischen Gleichungssystemen. *Master Thesis*. Berlin: Technische Universität Berlin

– M. Baumgart (2021): Aufbau einer Datenbank zur generischen Erstellung und Verwaltung/Modifikation von Projektrohrklassen für 3D-Rohrleitungskomponenten. *Master Thesis*. Berlin: Technische Universität Berlin

– A. Reum (2022): Efficiency Improvement of an Interval-Arithmetic-based Solution Method applied on a stationary Flash Model for Separating a Multi-Component System. *Bachelor Thesis*. Berlin: Technische Universität Berlin

– P. Polgar (2022): Entwicklung eines Konzeptes zur Messung von Entrainment auf Kolonnenböden. *Master Thesis*. Berlin: Technische Universität Berlin

Contents

List of Figures

List of Tables

List of Symbols

Constants

Symbol	Description	Unit
π	Ratio of circle's circumference to its diameter = 3.141 59...	1
R	Ideal gas constant = 8.314 46	$\mathrm{J\,mol^{-1}\,K^{-1}}$

Greek Symbols

Symbol	Description	Unit
α	Parameter equation of state	1
α	Relative volatility	1
α	Step size control	1
Δ	Difference	various
Δ	Trust region radius	1
γ	Parameter in scaling	1
κ	Condition number	1
Λ	Diagonal matrix with eigenvalues	1
λ	Eigenvalue	1
λ	Lagrange multiplier	1
μ	Lagrange multiplier	1
ω	Weighing factor	1
ρ	Density	1
ρ	Parameter in scaling	1
σ	Correcting factor	1
ε	Tolerance	1
φ	Fugacity coefficient	1

Indices

Symbol	Description
b	Index for boxes
i	Index for components
i	Index for variables
j	Index for functions
k	Index for inequalities
k	Index for second components
l	Index for subboxes
p	Index for parameters
s	Index for solutions
s	Index for split boxes
tr	Index for trays

Latin Symbols

Symbol	Description	Unit
$\boldsymbol{A}$	Matrix	1
$\boldsymbol{B}$	Approximation of Jacobian matrix	1
$\boldsymbol{b}$	Vector with functions	various
$\boldsymbol{D}$	Diagonal matrix	1
$\boldsymbol{d}$	Iteration step	various
$\boldsymbol{E}$	Correction matrix	1
$\boldsymbol{f}$	Vector with functions	various
$\boldsymbol{G}$	Preconditioner for matrix	1
$\boldsymbol{g}$	Vector with functions	various
$\boldsymbol{H}$	Hessian matrix	various
$\boldsymbol{h}$	Vector with functions	various
$\boldsymbol{I}$	Identity matrix	1
$\boldsymbol{J}$	Jacobian matrix	various
$\boldsymbol{k}$	Vector of variables	various
$\boldsymbol{m}$	Vector with functions	various
$\boldsymbol{P}$	Row permutation matrix	1
$\boldsymbol{Q}$	Column permutation matrix	1
$\boldsymbol{r}$	Function residuals	various
$\boldsymbol{V}$	Eigenvector matrix	1
$\boldsymbol{x}$	Vector of variables	various

$\boldsymbol{y}$	Vector of variables	various
$\boldsymbol{z}$	Vector of variables	various
A	Area	m^2
a	Parameter equation of state	$\mathrm{Pam}^6/\mathrm{mol}^2$
a	Scalar value	various
aux	Auxiliary variable	various
av	Auxiliary variable	various
B	Parameter equation of state	1
b	Parameter equation of state	$\mathrm{mol\,m}^{-3}$
b	Scalar function	various
b	Scalar value	various
c	Scalar value	1
cb	Boolean true if box can be cut	1
D	Parameter equation of state	1
d	Boolean true if box is discontinuous	1
d	Scalar iteration step	various
F	Flow rate	$\mathrm{mol\,s}^{-1}$
f	Scalar function	various
$fslv$	Solver fsolve from SciPy package	1
g	Scalar function	various
h	Molar enthalpy	$\mathrm{J\,mol}^{-1}$
h	Pressure	Pa
h	Scalar function	various
$ipopt$	Solver IPOPT from COINOR foundation	1
K	Equilibrium constant	1
k	Heat transfer coefficient	$\mathrm{W\,m}^{-2}\,\mathrm{K}^{-1}$
k	Scalar variable	various
l	Box index regarding current reduction step's box order	1
$lnphi$	Logarithmic fugacity coefficient value	1
m	Scalar function	various
n	Decision variable	1
n	Quantity	1
npz	Path to npz archive	1
num	Set of real solutions	various
nwt	Solver newton from modopt package	1
p	Constant parameter	various
p	Tuple with reduction step and index of parent box	1
par	Boolean true for parallel processing	1
Q	Heat flux	W
q	Parameter equation of state	1

q	Scalar value	1
R	Reflux ratio	1
r	Boolean true if box can be reduced	1
r	Scalar residual	various
results	Dictionary with results	various
s	Boolean true if box is consistent	1
s	Boolean true if box is solved	1
slsqp	Solver SLSQP from SciPy package	1
T	Temperature	K
unique	Boolean true if box is unique	1
v	Molar volume	$\mathrm{mol\,m^{-3}}$
v	Variable name	1
x	Mole fraction (liquid)	1
x	Scalar variable	various
y	Mole fraction (vapor)	1
y	Scalar variable	various
Z	Compressibility factor	1
z	Mole fraction (liquid)	1
z	Scalar variable	various

Operators

Symbol	Description	Definition
$\Gamma(\boldsymbol{f}(\boldsymbol{x}), \overline{\underline{\boldsymbol{x}}})$	Contractor	$\Gamma(\boldsymbol{f}(\boldsymbol{x}), \overline{\underline{\boldsymbol{x}}}) \subset \overline{\underline{\boldsymbol{x}}}$
$\nabla_{\boldsymbol{xx}}$	Second partial derivatives	$\nabla_{\boldsymbol{x}}(\nabla_{\boldsymbol{x}})$
$\nabla_{\boldsymbol{x}}$	First partial derivatives	$\left[\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right]^T$

Superscripts

Symbol	Description
$*$	At Solution
(k)	K-th iteration or reduction step
2ph	Two phase region
AA	Active
Abs	Absolute
BL	Box length
B	Bottom

C	Condenser
c	Cool
D	Distillate
fTol	Function tolerance
F	Feed
h	Hot
IA	Inactive
in	Inlet
new	New element
n	Molar
old	Former element
out	Outlet
parent	Parent box
RABL	Relative average box length
RADL	Relative average domain length
RA	Relative average
RBL	Relative box length
Rel	Relative
RHL	Relative hypercubic length
RW	Relative width
R	Reboiler
sc	Subcooled
sh	Superheated
uni	Unified

Subscripts

Symbol	Description
bc	Box consistency
box	Box
C	Cauchy
c	Consistent
c	Point of expansion
hc	Hull consistency
l	Left
lb	Lower bound
ln	Logarithmic
ls	Largest subsystem
m	Prefix for maximum
max	Maximum

min	Minimum
mix	Mixture
nl	Nonlinear
nwt	Newton
nz	Nonzero
par	Parallel
R	Reduced
r	Right
$rsol$	Resolution
s	Prefix for sub
sd	singular or discontinuous
seq	Sequential
sol	Solution
$solved$	Solved
$step$	Reduction step
t	Tear variable
ub	Upper bound
var	Variable

List of Abbreviations

Numerics

BBTF	**B**ordered **B**lock **T**riangular **F**orm
BFGS	**B**royden, **F**letcher, **G**oldfarb, and **S**hanno
BL	**B**ox **L**ength
CMAES	**C**ovariance **M**atrix **A**daption **E**volution **S**trategy
CPU	**C**omputational **P**rocessing **U**nit
DAE	**D**ifferential **A**lgebraic **E**quation system
DM	**D**ulmage **M**endelsohn
IA	**I**nterval **A**rithmetic
INGB	**I**nterval **N**ewton with **G**eneralized **B**isection
LU	**L**ower **U**pper
MINLP	**M**ixed **I**nteger **N**on **L**inear **P**roblem
MSO	**M**odeling, **S**imulation, and **O**ptimization
NLE	**N**on **L**inear **E**quation system
NLP	**N**on **L**inear **P**roblem
OS	**O**perating **S**ystem
PDAE	**P**artial **D**ifferential **A**lgebraic **E**quation system
RABL	**R**elative **A**verage **B**ox **L**ength
RADL	**R**elative **A**verage **D**omain **L**ength
RBL	**R**elative **B**ox **L**ength
RHL	**R**elative **H**ypercubic **L**ength
SQP	**S**equential **Q**uadratic **P**rogramming
UDLS	**U**ser **D**efined **L**anguage **S**pecificator

Chemical Engineering

CSTR	**C**ontinuous **S**tirred-**T**ank **R**eactor
DIPPR	**D**esign **I**nstitute for **P**hysical **PR**operties
HDA	**H**ydro**D**e**A**lkylation process
MESH	**M**aterial balance, **E**quilibrium, **S**ummation, **H**eat equations
SRK	**S**oave **R**edlich **K**wong's equation of state
VLE	**V**apor **L**iquid **E**quilibrium

1 Introduction

"The important thing about a problem is not the solution, but the strength we gain in finding a solution." — Seneca

1.1 Motivation

Chemical processes are expected to operate safely, economically and, to an increasing extent, ecologically, taking into account the growing challenges posed by resource shortages and climate change. In order to achieve these goals efficiently, computer-aided ***M**odeling, **S**imulation, and **O**ptimization* (MSO) of processes have become an indispensable part of plant design and are steadily gaining importance in the operation of existing processes (Bortz and Asprion, 2022), (Bröcker et al., 2021), (Martin et al., 2022). The fields of application of MSO referring to Biegler (2014) can be subdivided into

- **Process and model development**: Transfer from experimental data to process model by experimental design, parameter estimation, comparison of different modeling approaches

- **Process synthesis and design**: Determination of optimal process structure and conditions as well as equipment design

- **Process operation**: Optimal control of fluctuating process conditions, planning and scheduling of process operations

But how is MSO carried out? The flow chart by Asprion & Bortz (2018) shown in Fig. 1.1 illustrates quite well the connection between the three disciplines. For basic research, modeling and simulation are of particular importance. Here,

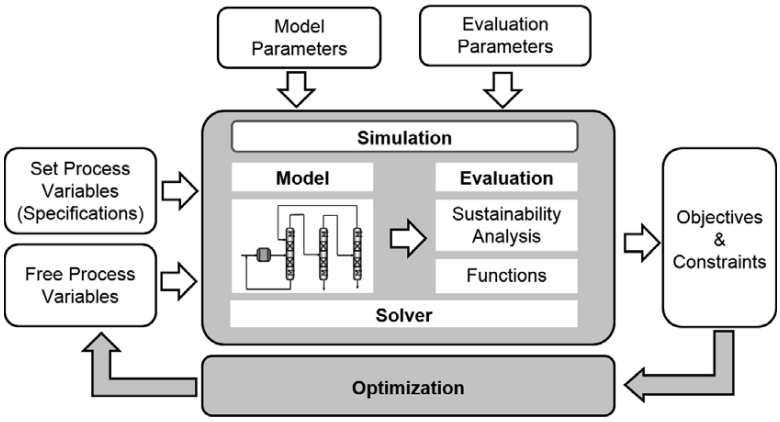


Fig. 1.1: Connection between process modeling, simulation and optimization according to Asprion & Bortz (2018).

the emphasis is on representing the real problem as accurately as necessary in a mathematical model. This usually cannot be solved analytically and so process simulation comes into play, which is primarily reliant on the use of numerical solution methods. By means of an existing model and a converging simulation, different process conditions can be tested, which form the basis for, e.g., new process concepts. The simulation results of a process model can be evaluated to define objective functions with respect to sustainability and/or economic efficiency under consideration of certain operating conditions and/or safety aspects. In optimization, these objective functions are then minimized or maximized with subject to the constraints. Possible objectives can be, e.g., the minimization of energy and resource demand, the reduction of waste streams, or to find the total cost minimum considering fixed and operational costs under variable process conditions according to a multi criteria optimization. It is not always the case that a process simulation and thus also the process optimization depending on it can find the desired solution for the process model. Frequently process models are solved by equation-based algorithms such as fixed-point iteration, root-finding methods or an optimization algorithm that relies on the latter. They often fail due to

– **Infeasible regions or points** of variables where a function from the mathematical system of interest is not defined, e.g. $1/x$ has no solution at $x = 0$. If x does not represent only one, but many variables, it is difficult to react to

the infeasible combination of variables within the iteration process. Further critical expressions can be found in Amarger et al. (1992).

- **Ill-conditioning** in the linear system that is supposed to be solved in the iteration step of a derivative-based numerical method. Huge differences in the Jacobian's nonzero entries result in the matrix to become singular or almost singular. Such a system either has no or infinitely many solutions causing the solver to abort or to take an inaccurate iteration step (Dennis and Schnabel, 1996, pp. 52).

- **Multiple solutions** may exists to the mathematical system. Locally convergent solvers only converge to one solution. There is no easily verifiable condition under which it can be guaranteed that the solver converges exactly to the desired physically feasible solution(s) (Mazumder, 2016, pp. 417-418).

Challenging are process models, in which waste streams are decreased through recycling or the heat duty can be saved through heat integration. Both lead to strongly coupled equations that can no longer be solved independently of each other. The more variables in these equations, the larger the possible solution space and the chance for infeasible points to occur if critical function expressions are present. In addition, variables of different orders of magnitude are very likely to occur in process models, e.g., mass transfer related quantities such as particle diameter and film thickness with a scale around 1 µm compared to quantities related to the process unit itself such as its diameter, height or length in the range of 1 m (Grossmann and Westerberg, 2000). This can cause ill-conditioned systems. Through preconditioning methods, as explained in section 2.4, the condition of a system often improves, but it may remain bad, especially in large, strongly coupled equation systems. An ***N**on **L**inear **E**quation system* (NLE) can have numerous solutions, e.g., in case complex reaction kinetics or multicomponent phase equilibria are part of the process model. Convergence of the associated simulation to a solution is often only achieved, when the numerical solver starts from an initial point in the solution's immediate vicinity. Such a point is difficult to find especially in large NLEs. To overcome these issues, some successful strategies and alternative numerical methods to fixed-point and root-finding based algorithms have already been developed.

Continuation methods attempt to find a sufficiently close initial point for fixed-point or root-finding based algorithms. First, a simplified problem is solved. The original system is extended by a homotopy parameter h, which is decreased step-wise from the simplified model ($h = 1$) to the original one ($h = 0$). The solution from the current step is the initial point for the next step. More details on homotopy methods are given in Guddat et al. (1990). In the pseudo-transient continuation approach an NLE is solved by first converting it into a ***D**ifferential **A**lgebraic **E**quation system* (DAE) and then searching for the stationary point in pseudo-time, which corresponds to the solution of the original problem (Pattison and Baldea, 2014). In Seo et al. (2020) it could already be shown that this methodology increases the solution's region of attraction for solvers designed to solve root-finding problems so that they also converge from further distanced initial points. Nevertheless, there is no guarantee for homotopy and pseudo continuation methods to converge at all and to the expected solution (Neumaier et al., 2005, p. 291-292).

Constraint-propagation methods in combination with bisectioning algorithms can guarantee to find all solutions in initial variable bounds (Schnepper and Stadtherr, 1996). They make use of the ***I**nterval **A**rithmetic* (IA) and successively remove all infeasible variable domains. However, since there can be many infeasible regions, especially in large NLEs, there are many possible solution spaces left in between. All of them must be checked individually for solutions, which results in slow to intractable convergence speeds. Nonetheless, the potential of such methods is great, since the examination of solution spaces can be parallelized very well. If the computers of tomorrow have significantly more cores available, the efficiency of constraint propagation methods can also be substantially increased (Hansen and Walster, 2003, p. 13).

Sequential modular approach and tearing algorithms use the downstream structure of a process flowsheet (Biegler et al., 1999, pp. 271-285) or decompose the entire equation system of a process model into a sequence of subsystems that are successively solved (Duff et al., 2017a, pp. 108-136). The output of the former subsystem is the input of the next. Due to recycle streams or strongly-coupled equations, the subsystem at the beginning of the sequence also depends on the output at the end of the sequence. This results in an outer iteration, in which the linking variables, also called tearing variables, are reduced. The algorithm

then alternates between inner and outer iteration. As long as very good estimates for the tearing variables are initially available, this procedure converges in a few steps and the computational time can be well reduced (Bublitz et al., 2017a). However, such algorithms are also very sensitive to the choice of the tearing variables' initial values and tend to behave in an unstable manner in ill-conditioned systems (Baharev, 2016).

Problem specific algorithms make use of the mathematical structure of a particular process unit. A prominent example is the general model of a column with equilibrium trays, which is often used in distillation, absorption and extraction processes. The cascade of separation trays is exploited in the Thomas algorithm, an algorithm developed specifically for its mathematical structure (Thomas, 1949), and combined with a tearing method. Thus, only initial values for the tearing variables have to be given, which are the trays' temperatures and internal streams of the light phase. The output of the Thomas algorithm are the trays' mole fractions of the heavy phase. This data is the input to the general phase equilibrium model and the trays' total material and heat balances, which then update the tearing variables and the procedure continues until the values of the tearing variable do not change anymore, i.e., the system is solved. For phase equilibrium models, specific solution strategies exist as well. The bubble point and sum rate method are used for narrow- and wide-boiling mixtures respectively (Friday and Smith, 1964) and have been extended by Tsuboka & Katayamak (1976) for isotherm extraction. A schematic representation of the bubble point method is shown in Appendix A.1. Another column algorithm is the inside-out method, introduced by Boston & Sullivan (1974). This algorithm is particularly suitable for solving systems with complex thermodynamics, due to its speed and robustness in this case. The inside-out method creates a simplified model from the complex thermodynamics in an outer loop at constant flow rates, mole fractions and temperatures. The inner loop iterates the latter variables by using this simplified thermodynamic model. As it requires much less computational operations, the simplified model makes the inner iteration very fast. Only when simplified and complex thermodynamics deviate strongly from each other, the simplified model is updated according to the latest iteration variables from the inner loop. Wang et al. (2020) have recently extended the inside-out method for reactive distillation. The presented methods are faster and more robust than simply trying to solve the entire equation system

by a fixed-point or root-finding algorithm. However, no generalization for other process units has been found and it takes a lot of effort to identify the individual structure of a new process unit and to develop an appropriate solution strategy.

The author believes that

- the number of nonlinear terms in equations,
- the differences in the scales within a model,
- the number of coupled equations, and
- the dimensions of the equation systems

will increase in rigorous process models in future, to simulate and optimize even more complex relations in processes or even larger NLEs resulting from discretized processes in time and/or space, in order to cope with the growing demands on chemical production in line with the requirements of the environment and market forces (Biegler et al., 2022, p. 33). In the author's opinion, a generally valid approach for initialization and solving such systems is currently missing. Ideally, it should create a connection between process model and numerical iteration method independent of the problem itself.

1.2 Research Goal

Presently, there is no problem-independent method for initializing NLEs that attempts to ensure a robust numerical iteration by root-finding based algorithms. In this work, such a method is to be developed. The idea is to combine ***I**nterval **A**rithmetic* (IA)-based methods with state-of-the-art root-finding based algorithms. The former theoretically guarantee to find all solutions, while the latter show much higher convergence speeds. The combination of both is relatively promising and has been tested before for example in van Iwaarden & Lodwick (1996, pp. 54-69), Figueiredo et al. (1997) and Baharev et al. (2011). While the latter focused on solving problems globally, the aim of this work is to locate one or a few solution(s) quickly. Therefore, the so-called hybrid approach should take the following points into account:

- Filtering infesaible points or regions within the initial variable domain by IA

- Efficient reduction of the initial variable domain by IA

- Recurrent local searches by root-finding based algorithms, started from promising points in IA-reduced variable domain.

The application range of such a hybrid approach shall be further investigated by computational experiments. The following questions will be addressed:

RQ1. How well does the hybrid approach perform in locating a desired solution compared to state-of-the-art root-finding algorithms?

RQ2. How well does the hybrid approach perform to solve an NLE globally compared to a pure IA-based solver?

To ensure the independence of problems, the hybrid approach needs to be applied to several process models. Complex problems are of particular interest here, which can otherwise only be solved by existing numerical methods after a costly initialization. However, in this matter, there are a few unresolved points that will be investigated in this work:

RQ3. How can the complexity of an NLE be measured?

RQ4. Which equation formulations are useful to ensure an efficient numerical iteration?

RQ5. Can structural properties of an NLE be used to conclude, which method is the appropriate to solve it?

Finally, during the development of the hybrid approach, attention will be paid to create features or tools that assist in debugging failed numerical iterations, especially whether the failure is caused by a structural error in the model or a numerical error of the method.

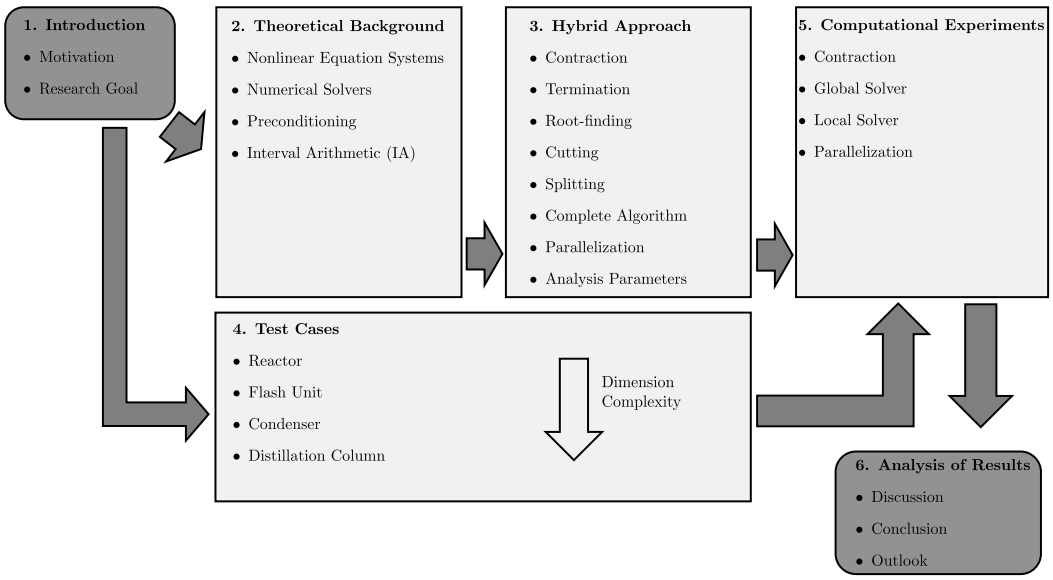


Fig. 1.2: Outline of this work.

1.3 Outline of Work

Figure 1.2 sketches the key points of each chapter. First, the theoretical background is given to subsequently introduce the novel hybrid approach. The developed process models are presented in chapter four and tested in chapter five. Finally the application area with regard to the set research goals and questions are discussed, and an outlook for further method development is given.

2 Theoretical Background

This chapter contains state-of-the-art real arithmetic and IA-based methods, which are relevant for the development of the novel hybrid approach. It also introduces the standard form of the NLE, which the hybrid approach will be designed for.

2.1 Notation

The mathematical notation introduced now is used throughout this thesis. The novel hybrid approach aims to find numerical solutions to well-determined problems of the type

$$\boldsymbol{f}(\boldsymbol{x}) = 0 \quad \boldsymbol{x} \in \mathbb{R}^n \quad , \tag{2.1}$$

for the real-valued functions

$$\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n \quad .$$

A set of nj solutions $\boldsymbol{x}_j^*$, which solve such a problem, is denoted as

$$\begin{aligned} \{\boldsymbol{x}^*\} &:= \{\boldsymbol{x}_j^* \in \mathbb{R}^n \mid j = 1, \ldots, nj;\ \boldsymbol{f}(\boldsymbol{x}_j^*) = 0\} \\ &= \{\boldsymbol{x}_{j=1}^*, \ldots, \boldsymbol{x}_{j=nj}^*\} \quad . \end{aligned}$$

Bold letters are used for multi-dimensional functions and vectors, while regular letters denote scalar quantities. Real-valued matrices will be presented by bold capital letters.

The hybrid approach searches for one or multiple solutions to problem 2.1 within the user-defined variables space $\overline{\underline{\boldsymbol{x}}}$ that is defined as

$$\begin{aligned}\overline{\underline{\boldsymbol{x}}} :=& \{\boldsymbol{x} \in \mathbb{R}^n \mid \forall x_i \in \mathbb{R} : \ \underline{x}_i \leq x_i \leq \overline{x}_i; \ \underline{x}_i, \overline{x}_i \in \mathbb{R}; \\ & i = 1, \ldots, n\} \\ =& \begin{pmatrix} \overline{\underline{x}}_{i=1} \\ \vdots \\ \overline{\underline{x}}_{i=n} \end{pmatrix} \quad .\end{aligned}$$

Such a variable space is also called a box, as it is spanned by n closed variable intervals $\overline{\underline{x}}_i$. A real-valued interval $\overline{\underline{x}}$ is defined as the closed set of all numbers x within the range of a real-valued lower bound $\underline{x}$ to a real-valued upper bound $\overline{x}$

$$\overline{\underline{x}} := \{x \in \mathbb{R} \mid \ \underline{x} \leq x \leq \overline{x}; \ \underline{x}, \overline{x} \in \mathbb{R}\} \quad .$$

Width w, midpoint m and absolute value $|\dot{-}|$ of an interval $\overline{\underline{x}}$ are defined as

$$\begin{aligned} w(\overline{\underline{x}}) &:= \overline{x} - \underline{x} \\ m(\overline{\underline{x}}) &:= \frac{\overline{x} + \underline{x}}{2} \\ |\overline{\underline{x}}| &:= \max\{|\underline{x}|, |\overline{x}|\} \quad . \end{aligned}$$

These quantities can also be calculated for a box $\overline{\underline{\boldsymbol{x}}}$

$$\begin{aligned} w(\overline{\underline{\boldsymbol{x}}}) &:= (w(\overline{\underline{x}}_{i=1}), \ldots, w(\overline{\underline{x}}_{i=n})) \\ m(\overline{\underline{\boldsymbol{x}}}) &:= (m(\overline{\underline{x}}_{i=1}), \ldots, m(\overline{\underline{x}}_{i=n})) \\ |\overline{\underline{\boldsymbol{x}}}| &:= \max\{|\overline{\underline{x}}_{i=1}|, \ldots, |\overline{\underline{x}}_{i=n}|\} \quad . \end{aligned}$$

An interval is termed degenerate, when it contains only one real point. A box is degenerate when it only consists of degenerate intervals. Lowest and highest real

point of a box $\underline{\overline{\boldsymbol{x}}}$ are denoted as

$$\underline{\boldsymbol{x}} := \begin{pmatrix} \underline{x}_{i=1} \\ \vdots \\ \underline{x}_{i=n} \end{pmatrix} \qquad \overline{\boldsymbol{x}} := \begin{pmatrix} \overline{x}_{i=1} \\ \vdots \\ \overline{x}_{i=n} \end{pmatrix} \qquad \underline{\boldsymbol{x}}, \overline{\boldsymbol{x}} \in \mathbb{R}^n$$

During the reduction of an initial box, infeasible regions in the interior of intervals might be removed that cannot contain any solution(s) to Eq. 2.1. A convex, infeasible region within an interval is called a gap. By discarding gaps, the initial box splits up into subboxes of feasible variable domains. A set of subboxes $\{\underline{\overline{\boldsymbol{x}}}\}$ has the form

$$\{\underline{\overline{\boldsymbol{x}}}\} := \{\underline{\overline{\boldsymbol{x}}}_{b=1}, \ldots, \underline{\overline{\boldsymbol{x}}}_{b=nb}\} \quad .$$

The exact image set related to a scalar function applied on a certain box $\underline{\overline{\boldsymbol{x}}}$ is

$$\underline{\overline{f}}^*(\underline{\overline{\boldsymbol{x}}}) = \underline{\overline{f}}^*(\underline{\overline{x}}_{i=1}, \ldots, \underline{\overline{x}}_{i=n}) := \{f(x_{i=1}, \ldots, x_{i=n}) \in \mathbb{R} \mid x_{i=1} \in \underline{\overline{x}}_{i=1}, \ldots, x_{i=n} \in \underline{\overline{x}}_{i=n}\}.$$

A one-dimensional example is the following

$$f(x) = 2 \cdot x - x, \quad x \in [0, 1] \implies \underline{\overline{f}}^*([0, 1]) = [0, 1] \tag{2.2}$$

For non-monotonic function expressions with multiple variable dependencies the calculation of $\underline{\overline{f}}^*(\underline{\overline{\boldsymbol{x}}})$ becomes computationally expensive. An alternative way is to overestimate the range of a real function's values by its interval extension. The interval extension $\underline{\overline{f}}(\underline{\overline{\boldsymbol{x}}})$ of a scalar function f is denoted as

$$\begin{aligned} \underline{\overline{f}}(\underline{\overline{\boldsymbol{x}}}) := \{ & y \in \mathbb{R} \mid \\ & \underline{y} = \min f(\underline{\overline{\boldsymbol{x}}}) \leq y \leq \overline{y} = \max f(\underline{\overline{\boldsymbol{x}}}); \\ & \underline{y}, \overline{y} \in \mathbb{R}\} \quad . \end{aligned}$$

The result differs from problem 2.2 as the interval $\underline{\overline{x}}$ is directly processed in the function instead of its real-valued elements.

$$\underline{\overline{f}}(\underline{\overline{x}}) = 2 \cdot \underline{\overline{x}} - \underline{\overline{x}}, \quad \underline{\overline{x}} = [0, 1] \implies \underline{\overline{f}}([0, 1]) = [-1, 2] \tag{2.3}$$

Images and interval extensions can be also formulated for systems with several functions. The image set of a system consisting of m functions and n variables is

$$\overline{\underline{\boldsymbol{f}}}^{*}(\overline{\underline{\boldsymbol{x}}}) = \overline{\underline{\boldsymbol{f}}}^{*}(\overline{\underline{x}}_{i=1},\ldots,\overline{\underline{x}}_{i=n}) := \{\boldsymbol{f}(x_{i=1},\ldots,x_{i=n}) \in \mathbb{R}^m \mid x_{i=1} \in \overline{\underline{x}}_{i=1},\ldots,x_{i=n} \in \overline{\underline{x}}_{i=n}\}$$

and the interval extended version $\overline{\underline{\boldsymbol{f}}}$ is

$$\begin{aligned}\overline{\underline{\boldsymbol{f}}}(\overline{\underline{\boldsymbol{x}}}) :=& \{\boldsymbol{y} \in \mathbb{R}^m \mid \forall y_i \in \mathbb{R}: \\ & \underline{y}_i = \min f_i(\overline{\underline{\boldsymbol{x}}}) \leq y_i \leq \overline{y}_i = \max f_i(\overline{\underline{\boldsymbol{x}}}); \\ & \underline{y}_i, \overline{y}_i \in \mathbb{R};\ i = 1,\ldots,n\} \quad .\end{aligned}$$

It should be noted that an image set is always tighter or as tight as the enclosure given by the interval extension. An $m \times n$ interval matrix $\overline{\underline{\boldsymbol{A}}}$ has interval entries so that

$$\overline{\underline{\boldsymbol{A}}} := \begin{bmatrix} \overline{\underline{a}}_{j=1,i=1} & \cdots & \overline{\underline{a}}_{j=1,i=n} \\ \vdots & \ddots & \vdots \\ \overline{\underline{a}}_{j=m,i=1} & \cdots & \overline{\underline{a}}_{j=m,i=n} \end{bmatrix} \quad .$$

Hence, an interval matrix $\overline{\underline{\boldsymbol{A}}}$ contains the set of all real-valued matrices $\boldsymbol{A}$ within the given ranges. An interval matrix is regular if it does not contain any real-valued matrix that is singular, otherwise it is termed irregular (Hansen and Walster, 2003, p. 84). The vectors of the j-th row or i-th column of a real-valued matrix $\boldsymbol{A}$ are given in abbreviated form as

$$a_{j,:} := (a_{j,i=1} \ldots a_{j,i=n}) \qquad a_{:,i} := \begin{pmatrix} a_{j=1,i} \\ \vdots \\ a_{j=m,i} \end{pmatrix} \quad .$$

2.2 Nonlinear Equation Systems

In general, all mathematical models associated to chemical engineering processes can be formulated as an optimization problem. Common use cases arise from

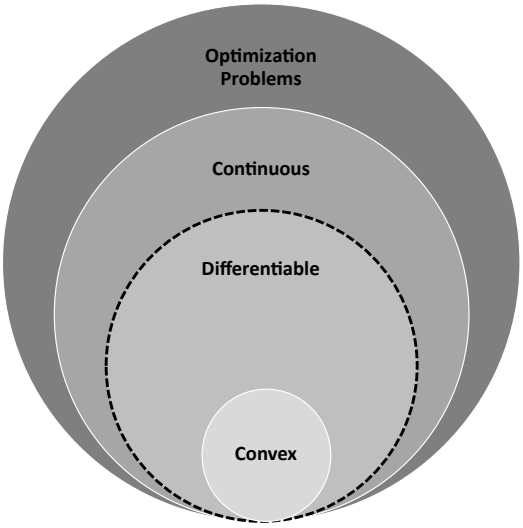


Fig. 2.1: Classification of optimization problems.

model development, process design, process operation, process control, and real-time optimization (Biegler, 2010, p. 1). In order to make numerical methods easily applicable to solve the equation systems of such problems, a standard mathematical formulation has been established:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y}} \quad & f(\boldsymbol{x},\boldsymbol{y}) \\ s.t. \quad & h_j(\boldsymbol{x},\boldsymbol{y}) = 0, \quad j := 1,\ldots,m \\ & g_k(\boldsymbol{x},\boldsymbol{y}) \leq 0, \quad k := 1,\ldots,p \\ & \boldsymbol{x} \in \mathbb{R}^n, \quad \boldsymbol{y} \in \mathbb{Z}^t \quad , \end{aligned} \tag{2.4}$$

with the scalar objective function $f(\boldsymbol{x},\boldsymbol{y})$, m equality constraints $\boldsymbol{h}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{0}$, p inequality constraints $\boldsymbol{g}(\boldsymbol{x},\boldsymbol{y}) \leq \boldsymbol{0}$, n continuous variables $\boldsymbol{x}$ and t discrete variables $\boldsymbol{y}$ with a finite number of integer values. Every optimization problem can be easily formulated in form of Eq. 2.4. For example a maximized objective function $\max_{\boldsymbol{x},\boldsymbol{y}} f(\boldsymbol{x},\boldsymbol{y})$, is equivalent to $\min_{\boldsymbol{x},\boldsymbol{y}} -f(\boldsymbol{x},\boldsymbol{y})$, similarly an inequality constraint $g_k(\boldsymbol{x},\boldsymbol{y}) \geq 0$ is equivalent to $-g_k(\boldsymbol{x},\boldsymbol{y}) \leq 0$. Eq. 2.4 can be simplified if certain conditions hold. The classification relevant for the nonlinear equation systems investigated in this work is shown in figure 2.1. In case no discrete variables are present, Eq. 2.4 reduces to a continuous ***N**on **L**inear **P**roblem* (NLP)

of the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ s.t. \quad & h_j(\boldsymbol{x}) = 0, \quad j := 1, \ldots, m \\ & g_k(\boldsymbol{x}) \leq 0, \quad k := 1, \ldots, p \\ & \boldsymbol{x} \in \mathbb{R}^n \quad . \end{aligned} \tag{2.5}$$

The suggested hybrid approach is designed for this group of problems. Although it should be mentioned that discrete optimization problems, so-called ***M**ixed **I**nteger **N**on **L**inear **P**roblems* (MINLPs), are frequently relaxed to NLPs for example in branch and bound algorithms and therefore rely on NLP formulations and solvers as well (Schewe and Schmidt, 2019, pp. 159-171). If all function derivatives of $f(\boldsymbol{x})$, $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ exist, the problem is termed differentiable. In numerical iterations of chemical engineering related models this is not always the case. A simple example is the logarithmic temperature difference ΔT_{ln} frequently used for heat transfer calculations in heat exchangers

$$\Delta T_{ln} = \frac{\Delta T_l - \Delta T_r}{\ln\left(\frac{\Delta T_l}{\Delta T_r}\right)} \quad . \tag{2.6}$$

This equation and also its derivatives with respect to ΔT_l and ΔT_r are not defined at $\Delta T_l = \Delta T_r$. A derivative based iteration method would abort at this point. In the suggested hybrid approach such points shall be filtered out by IA so that ideally only variable domains with differentiable functions remain. For differentiable NLPs fast numerical solvers exist that can then be efficiently applied. For the sake of completeness the subgroup of convex, differentiable NLPs shall be shortly discussed. A problem from this group consists of a convex variable domain and convex functions $f(\boldsymbol{x})$, $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$. When both conditions hold, a locally found solution is always the global solution, if no better solution is available in its vicinity (Biegler, 2010, p. 4). In this work only well-determined, continuous NLPs are treated. This means the degrees of freedom of such a problem are zero and it owns as many unkowns, so-called iteration variables, as linearly independent equations. Well-determined, algebraic NLPs shall be denoted as NLEs in the following to emphasize their additional properties. In order to solve NLEs numerically, they can be formulated as root-finding, fixed-iteration or as

scalar minimization problems all originating from Eq. 2.5. In the first case the NLE is set up as

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}) &= \boldsymbol{0} \\ \boldsymbol{g}(\boldsymbol{x}) &\leq \boldsymbol{0} \quad , \end{aligned} \tag{2.7}$$

which is equivalent to Eq. 2.5 without an objective function to minimize. The inequality constraints are not mandatory but can restrict the ranges of the iteration variables $\boldsymbol{x}$ further so that a numerical solver focuses on this feasible region. Especially in chemical engineering problems non-physical solutions can be avoided this way. In case an iteration variable's range of validity is $\overline{\underline{x}}$ and can be restricted by a lower bound $\underline{x}$ and an upper bound $\overline{x}$ this can be formulated in terms of two inequality constraints by

$$\underline{x} - x \leq 0 \tag{2.8}$$

$$x - \overline{x} \leq 0 \quad , \tag{2.9}$$

to meet the structure of Eq. 2.7. Highly related to Eq. 2.7 is the formulation as a fixed-point problem given by

$$\begin{aligned} \boldsymbol{h}^*(\boldsymbol{x}) &= \boldsymbol{x} \\ \boldsymbol{g}(\boldsymbol{x}) &\leq \boldsymbol{0} \quad , \end{aligned} \tag{2.10}$$

whereas the equality constraints from Eq. 2.7 can be easily transformed to Eq. 2.10 by direction substitution shown in Eq. 2.11.

$$\boldsymbol{h}^* := \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{x} = \boldsymbol{x} \tag{2.11}$$

The third way to formulate and solve an NLE is as an optimization problem without equality constraints

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ s.t. \quad & \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \quad . \end{aligned} \tag{2.12}$$

When additionally no inequality constraints are present Eq. 2.12 becomes an unconstrained optimization problem. The equality constraints of Eq. 2.5 are indirectly used in the objective function. A common formulation is the so-called least-squares minimization problem with

$$f(\boldsymbol{x}) := 0.5 \cdot \sum_{j=1}^{m} h_j(\boldsymbol{x})^2 \quad . \tag{2.13}$$

The advantage is that Eq. 2.13 is a convex function in the vicinity of its roots and is bounded below by zero. Nevertheless, a solution to the least-squares problem is not necessarily a root of Eq. 2.7, because methods might stop at local minima of 2.13, which are non-zero. The next section presents methods to solve these three types of problems.

2.3 Numerical Solvers

Iterative methods aim to find a sequence of points $\boldsymbol{x}^{(k)}$ for $k = 1,2,\ldots,\infty$ converging to a solution $\boldsymbol{x}^*$ of an NLE of interest. The converging sequence is

$$\lim_{k\to\infty} \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_p = \boldsymbol{0} \quad , \tag{2.14}$$

with $\|\cdot\|_p$ being any of the p-vector norms. Typical norms are listed in table 2.1. The speed an iterative method converges with is usually estimated by the inequality

$$\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^*\|_p \leq c \cdot \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_p^q \quad c > 0, \quad q \geq 1 \quad , \tag{2.15}$$

with c and q denoted as rate and order of convergence. The rate of convergence is defined as

$$c := \lim_{k\to\infty} \frac{\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^*\|_p}{\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^*\|_p^q} \quad . \tag{2.16}$$

An iterative method only converges for $c < 1$. The lower the value of c is, the higher the speed of convergence. Methods with an order of convergence $q = 1$ are said to converge linearly, those with $1 < q < 2$ converge superlinearly and the ones with $q = 2$ converge quadratically. Hence, low convergence rates and high orders of convergence are desired to achieve fast converging processes.

Nevertheless, iterative methods with orders higher than $q = 2$ are rarely in use in large NLEs as the additional effort to compute their respective iteration step does not pay off for the obtained increase in convergence speed. Fixed-point iteration methods seek for a so-called fixed point $\boldsymbol{x}^*$ that solves

$$\boldsymbol{x} = \boldsymbol{h}^*(\boldsymbol{x}) \quad , \tag{2.17}$$

by the calculation specification

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{h}^*(\boldsymbol{x}^{(k)}) \quad . \tag{2.18}$$

The advantage of this approach is that this computation is relatively cheap as each iteration point can be directly calculated by one evaluation of the equation system at the last iterate. The disadvantage is that it only converges when

$$\max_j |\lambda_j(\boldsymbol{J}(\boldsymbol{x}^*))| < 1 \quad , \tag{2.19}$$

where $max_j|\lambda_j(\boldsymbol{J}(\boldsymbol{x}^*))|$ is the maximum absolute eigenvalue of the Jacobian matrix $\boldsymbol{J}(\boldsymbol{x})$ of $\boldsymbol{h}^*(\boldsymbol{x})$ at the fixed point (Heath, 2002, p. 238). The method converges mostly linearly with a convergence rate $\max_j |\lambda_j(\boldsymbol{J}(\boldsymbol{x}^*))|$. Hence, the lower the value of $\max_j |\lambda_j(\boldsymbol{J}(\boldsymbol{x}^*))|$ the faster the method converges. On the other hand, if this value is close to one, the method becomes very slow. Some relaxation methods such as Wegstein (1958), Orbach & Crowe (1971), Steffensen (1933) and Anderson (1965) exist to accelerate the convergence. However, to achieve the goals of this work, fixed-point iteration methods are not applied as their linear convergence rate and their initialization with points from rather coarse variable domains are expected to result in slow numerical iteration processes, especially for large NLEs.

Root-finding algorithms are better suited for this use case. They all have in common that they use some approximation of the functions' slopes at the iterated points. The probably best known root-finding algorithm is Newton's method. It applies the first order Taylor approximation on the iterate $\boldsymbol{x}^{(k)}$

$$\boldsymbol{m}(\boldsymbol{x}) := \boldsymbol{h}(\boldsymbol{x}^{(k)}) + \boldsymbol{J}(\boldsymbol{x}^{(k)}) \cdot (\boldsymbol{x} - \boldsymbol{x}^{(k)}) \quad , \tag{2.20}$$

to obtain a linear model $\boldsymbol{m}(\boldsymbol{x})$ of a root-finding problem referring to Eq. 2.7 with $\boldsymbol{J}(\boldsymbol{x}^{(k)})$ being the Jacobian matrix of the root functions $\boldsymbol{h}(\boldsymbol{x})$ evaluated at $\boldsymbol{x}^{(k)}$ (Dennis and Schnabel, 1996, pp. 69-77). The root $\boldsymbol{x}^{(k+1)}$, where $\boldsymbol{m}(\boldsymbol{x}^{(k+1)}) = \boldsymbol{0}$, can be analytically determined by

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{J}^{-1}(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{h}(\boldsymbol{x}^{(k)}) \quad . \tag{2.21}$$

Newton's method can solve linear root-finding problems in one iteration step because they are equivalent to $\boldsymbol{m}(\boldsymbol{x})$. According to Dahmen & Reusken (2008, p. 193) the method converges quadratically to a certain root if

- the iteration starts in a convex, Lipschitz-continuous vicinity of the root,
- the Jacobian matrix is Lipschitz-continuous in the vicinity of the root and,
- the inverse of the Jacobian matrix is not singular or almost singular in the vicinity of the root.

The drawbacks of the Newton method are that the determination of $\boldsymbol{J}(\boldsymbol{x}^{(k)})$ is rather expensive and no convergence is guaranteed whenever these conditions do not hold (Dennis and Schnabel, 1996, pp. 86-89). In practice, this often means that an initial guess close to the root must be already known, before Newton's method is started, but if the method converges, it does so quite quickly. Alternatively to Newton's method, the Quasi-Newton procedures avoid the costly computation of $\boldsymbol{J}(\boldsymbol{x}^{(k)})^{-1}$. A popular representative is Broyden's method (Broyden, 1965). The Jacobian matrix is replaced by an approximation matrix $\boldsymbol{B}^{(k)}$, which results from the multidimensional secant method under the condition that the change of $\boldsymbol{B}^{(k)}$ from one iteration step to the next is kept minimal in the Frobenius norm. The linear system solved equals

$$\boldsymbol{d}^{(k)} := \boldsymbol{B}^{-1(k)} \cdot \boldsymbol{f}^{(k)} \tag{2.22}$$

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} + \boldsymbol{d}^{(k)} \quad , \tag{2.23}$$

whereas $\boldsymbol{B}^{(0)}$ is initialized by either the actual Jacobian matrix, an approximation by finite differences or simply the unity matrix. It is then iteratively updated by

$$\boldsymbol{y}^{(k)} := \boldsymbol{f}^{(k+1)} - \boldsymbol{f}^{(k)} \tag{2.24}$$

$$\boldsymbol{B}^{(k+1)} := \boldsymbol{B}^{(k)} + \frac{(\boldsymbol{y}^{(k)} - \boldsymbol{B}^{(k)} \cdot \boldsymbol{d}^{(k)}) \cdot \boldsymbol{d}^{T(k)}}{\|\boldsymbol{d}^{(k)}\|^2} \quad . \tag{2.25}$$

Broyden's method has been further developed for the approximation of the inverse Jacobian matrix. This method is denoted as Broyden rank 1. The associated update formula becomes

$$\boldsymbol{B}^{-1(k+1)} := \boldsymbol{B}^{-1(k)} + \frac{(\boldsymbol{d}^{(k)} - \boldsymbol{B}^{-1(k)}) \cdot \boldsymbol{y}^{(k)}}{\boldsymbol{d}^{T(k)} \cdot \boldsymbol{B}^{-1(k)} \cdot \boldsymbol{y}^{(k)}} \quad . \tag{2.26}$$

Hence, Eq. 2.26 can be directly inserted into Eq. 2.22 and avoids the costly inversion

$$\boldsymbol{d}^{(k+1)} := \left(\boldsymbol{B}^{-1(k)} + \frac{(\boldsymbol{d}^{(k)} - \boldsymbol{B}^{-1(k)}) \cdot \boldsymbol{y}^{(k)}}{\boldsymbol{d}^{T(k)} \cdot \boldsymbol{B}^{-1(k)} \cdot \boldsymbol{y}^{(k)}} \right) \cdot \boldsymbol{f}^{(k+1)} \quad . \tag{2.27}$$

Both methods, Broyden and Broyden rank 1, converge superlinearly and thus slower than Newton's. In turn, each iteration step is less computationally expensive. (Quasi-)Newton methods are also used to solve optimization problems of the form Eq. 2.5. A minimum $\boldsymbol{x}^*$ has been found, when the problem fulfills the required constraints as well as the first and second order optimality conditions

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = 0 \tag{2.28}$$

$$\boldsymbol{y}^T \cdot \overbrace{\nabla_{\boldsymbol{x}\boldsymbol{x}} f(\boldsymbol{x})}^{\boldsymbol{H}(\boldsymbol{x})} \cdot \boldsymbol{y} \geq 0 \qquad \forall \boldsymbol{y} \in \mathbb{R}^n \quad . \tag{2.29}$$

In case an NLE is formulated as a least-squares problem, the objective function is zero at the roots of the NLE and $\boldsymbol{x}^*$ should be in the variable bounds $\overline{\underline{\boldsymbol{x}}}$. The latter can be considered through additional inequality constraints as shown in Eq. 2.8 and 2.9. If an iterate sits on one or multiple variable bounds the associated inequality constraint(s) are termed active, otherwise inactive. Such a general, constrained optimization problem can be replaced by a minimization of

the Lagrangian function $L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu})$

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu}} \quad & L(\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \cdot \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T \cdot \boldsymbol{g}(\boldsymbol{x}) \qquad \boldsymbol{x} \in \mathbb{R}^n \\ s.t. \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0} \qquad \boldsymbol{h},\boldsymbol{\lambda} \in \mathbb{R}^m \\ & \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \qquad \boldsymbol{g},\boldsymbol{\mu} \in \mathbb{R}^p \\ & \boldsymbol{\mu} \geq \boldsymbol{0} \quad . \end{aligned} \tag{2.30}$$

The Karush-Kuhn-Tucker conditions generalize the first order optimality conditions for Eq. 2.30. Hence, the associated root-finding problem needs to be solved to fulfill Eq. 2.28

$$\begin{aligned} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \sum_{j=1}^{m} \lambda_j \cdot \nabla_{\boldsymbol{x}} h_j + \sum_{k=1}^{p} \mu_k \cdot \nabla_{\boldsymbol{x}} g_k = 0 \\ \boldsymbol{h}(\boldsymbol{x}) = 0 \\ \boldsymbol{\mu}^T \cdot \boldsymbol{g}(\boldsymbol{x}) = 0 \\ \boldsymbol{\mu} \geq 0 \quad . \end{aligned} \tag{2.31}$$

The value of μ_k equals zero for any inactive inequality constraints g_k. Thus, only active inequality constraints have an influence on the minimum. The equations in Eq. 2.31 can be solved with a (Quasi-)Newton method. To ensure that the respective Hessian matrix $\boldsymbol{H}(\boldsymbol{x})$ is positive definite according to the second order optimality condition (Eq. 2.29) and does not become singular, several modifications exist. The modified Hessian matrix shall be denoted as $\boldsymbol{B}(\boldsymbol{x})$. One strategy is the Levenberg-Marquardt correction based on the work from Levenberg (1944) and Marquardt (1963), which ensures positive definiteness of $\boldsymbol{B}(\boldsymbol{x})$ by forcing it to have positive eigenvalues only. The method determines the eigenvalues of $\boldsymbol{H}(\boldsymbol{x})$ via a singular value decomposition and revises them if necessary to values that are greater than or equal to a minimum positive threshold

$$\boldsymbol{B}^{(k)} = \overbrace{\boldsymbol{V}^{(k)} \cdot \Lambda^{(k)} \cdot \boldsymbol{V}^{T(k)}}^{\boldsymbol{H}(\boldsymbol{x}^{(k)})} + \overbrace{\boldsymbol{V}^{(k)} \cdot \sigma \cdot I \cdot \boldsymbol{V}^{T(k)}}^{\boldsymbol{E}^{(k)}} \quad . \tag{2.32}$$

Here $V^{(k)}$ is a matrix containing the eigenvectors, $\Lambda^{(k)}$ is a diagonal matrix that consists of the respective eigenvalues and $\boldsymbol{E}^{(k)}$ is a diagonal matrix with σ being

the correcting value that is zero for a positive definite $\boldsymbol{H}(\boldsymbol{x})$ and greater equal the positive threshold otherwise. Another method is called steepest-descent, which uses the well-conditioned, positive definite identity matrix $\boldsymbol{I}$ in the Newton step

$$\boldsymbol{B}^{(k)} := \boldsymbol{I} \quad . \tag{2.33}$$

This method converges quite fast far away from the minimum but rather slow in its vicinity and can easily diverge without any additional step-size control mechanism. A third method by ***B**royden, **F**letcher, **G**oldfarb, and **S**hanno* (BFGS) determines the inverse Hessian matrix iteratively through

$$\begin{aligned} \boldsymbol{B}^{-1(k+1)} := \boldsymbol{B}^{-1(k)} + \frac{(\boldsymbol{d}^{T(k)} \cdot \boldsymbol{y}^{(k)} + \boldsymbol{y}^{T(k)} \cdot B^{-1(k)} \cdot \boldsymbol{y}^{(k)}) \cdot \boldsymbol{d}^{(k)} \cdot \boldsymbol{d}^{T,(k)}}{(\boldsymbol{d}^{T(k)} \cdot \boldsymbol{y}^{(k)})^2} \\ - \frac{\boldsymbol{B}^{-1(k)} \cdot \boldsymbol{y}^{(k)} \cdot \boldsymbol{d}^{T(k)} + \boldsymbol{d}^{(k)} \cdot \boldsymbol{y}^{T(k)} \cdot \boldsymbol{B}^{-1(k)}}{\boldsymbol{d}^{T(k)} \cdot \boldsymbol{y}^{(k)}} \quad . \end{aligned} \tag{2.34}$$

This formulation is highly related to the already introduced Broyden rank 1 method but additionally ensures the positive definiteness of $\boldsymbol{B}^{(k)}(\boldsymbol{x})$. To avoid slow convergence in (Quasi-)Newton methods, because of amplified oscillations (alternating iteration sequence in one search direction around root (Dennis and Schnabel, 1996, p. 24)) and zigzagging (alternating iteration sequence in two orthogonal search directions towards root (Nocedal and Wright, 2005, p. 148)), line search and trust region algorithms are often used. The former try to find a suitable step-size in the direction of the calculated iteration step with a sufficiently large reduction in the objective function value. An optimal step-size results from the minimization problem

$$\begin{aligned} \min_{\alpha^{(k)}} \quad & f(\boldsymbol{x}^{(k)} + \alpha^{(k)} \cdot \boldsymbol{d}^{(k)}) \\ s.t. \quad & \alpha^{(k)} \geq 0 \quad . \end{aligned} \tag{2.35}$$

To lower computational costs, the exact problem from Eq. 2.35 is usually not solved directly. Instead $\alpha^{(k)}$ is initially chosen very large in an iteration step and gradually reduced until certain conditions such as the ones from Armijo and

Goldstein are met,

$$f(\boldsymbol{x}^{(k)} + \alpha^{(k)} \cdot \boldsymbol{d}^{(k)}) \leq f(\boldsymbol{x}^{(k)}) + \omega \cdot \alpha^{(k)} \cdot \nabla_x f(\boldsymbol{x}^{(k)})^T \cdot \boldsymbol{d}^{(k)} \tag{2.36}$$

$$f(\boldsymbol{x}^{(k)} + \alpha^{(k)} \cdot \boldsymbol{d}^{(k)}) \geq f(\boldsymbol{x}^{(k)}) + (1-\omega) \cdot \alpha^{(k)} \cdot \nabla_x f(\boldsymbol{x}^{(k)})^T \cdot \boldsymbol{d}^{(k)} \quad , \tag{2.37}$$

with $0 < \omega \leq 0.5$. These conditions guarantee a sufficient descent in $f(\boldsymbol{x})$ but also an adequately large step-size. The detailed deviation is explained in Armijo (1966). With increasing number of iterations the initial value of each $\alpha^{(k)}$ is reduced so that the influence of the line search method vanishes during the process.

In contrast to line search methods, the search direction can also change in trust region methods, which leads to more flexibility and higher convergence speeds of the algorithm. At the same time, computational costs of one iteration step increase compared to line search methods. Trust region methods solve minimization problems of the form Eq. 2.5 by seeking for the optimal iteration step $\boldsymbol{d}^*$ minimizing the change in their objective $\Delta f(\boldsymbol{d}) := f(\boldsymbol{x}^{(k)} + \boldsymbol{d}) - f(\boldsymbol{x}^{(k)})$ in a given trust region. $\Delta f(\boldsymbol{d})$ is modeled by a second order Taylor approximation $m(\boldsymbol{d})$

$$\min_{\boldsymbol{d}} \quad \overbrace{\nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d} + \frac{1}{2} \cdot \boldsymbol{d}^T \cdot \boldsymbol{H}(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d}}^{m(\boldsymbol{d})} \tag{2.38}$$

$$s.t. \quad \|\boldsymbol{d}\| \leq \Delta \quad . \tag{2.39}$$

$\boldsymbol{H}(\boldsymbol{x}^{(k)})$ is the Hessian matrix of the scalar objective or any Quasi-Newton approximation of it. Inequality constraint 2.39 forces the iteration step length $\|\boldsymbol{d}\|$ commonly expressed in the Euclidian norm to stay within the current trust region's radius Δ. Several trust region algorithms exist that differ in the way the optimal iteration step is determined. A detailed overview and description of trust region methods is given in Conn et al. (2000). In Powell's trust-region-dogleg implementation (Powell, 1970) each iteration step equals a convex combination of the Cauchy step $\boldsymbol{d}_C$ and the Newton step $\boldsymbol{d}_{nwt}$

$$\boldsymbol{d} := \omega \cdot \boldsymbol{d}_{nwt} + (1-\omega) \cdot \boldsymbol{d}_C \quad , \tag{2.40}$$

with ω being the largest value in $[0, 1]$ so that condition 2.39 is fulfilled. Newton step and Cauchy step are given by

$$\boldsymbol{d}_{nwt} := -\frac{f(\boldsymbol{x}^{(k)})}{\nabla_x f(\boldsymbol{x}^{(k)})} \tag{2.41}$$

$$\boldsymbol{d}_C := -\alpha \cdot \nabla_x f^T(\boldsymbol{x}^{(k)}) \cdot f(\boldsymbol{x}^{(k)}) \quad . \tag{2.42}$$

The iteration step $\boldsymbol{d}_C$ equals the optimal choice in the direction of the steepest descent for Eq. 2.38 bounded upwards by Δ. The scalar α adjusts the respective step size of $\boldsymbol{d}_C$. Three cases for ω and the related step $\boldsymbol{d}$ are possible:

1) If $\Delta \geq \|\boldsymbol{d}_{nwt}\| \implies$ the Newton step is trustful and can be taken ($\omega = 1$)

2) Else If $\Delta = \|\boldsymbol{d}_C\| \implies$ the minimum of Eq. 2.38 in the steepest descent direction with subject to $\|d\| \leq \Delta$ lies at the trust region's bound and is fully taken ($\omega = 0$)

3) Else $(\Delta > \|\boldsymbol{d}_C\| \wedge \Delta < \|\boldsymbol{d}_{nwt}\|) \implies$ the convex combination referring to Eq. 2.40 adjusts ω in the interior of [0,1] so that $\|\boldsymbol{d}(\omega)\| = \Delta$

Hence, except for the first case, the next iterate sits always on the trust region's bound. If

$$\frac{\Delta f(\boldsymbol{d})}{m(\boldsymbol{d})} < \underline{\varepsilon} \quad , \tag{2.43}$$

the deviation between $m(\boldsymbol{x})$ and $f(\boldsymbol{x})$ in the current trust region is inacceptable and Δ needs to be decreased. A common choice is

$$\Delta := \underline{\varepsilon} \cdot \|\boldsymbol{d}\| \quad . \tag{2.44}$$

Whenever Δ needs to be decreased by Eq. 2.44 the iteration step $\boldsymbol{d}$ is rejected and its calculation is restarted from the last iterate. If on the other hand

$$\frac{\Delta f(\boldsymbol{d})}{m(\boldsymbol{d})} > \overline{\varepsilon} \quad , \tag{2.45}$$

the approximation $m(\boldsymbol{x})$ is able to predict the change of $f(\boldsymbol{x})$ very accurately. Hence, Δ can be increased to accelerate the procedure. Typically, Δ is doubled

until reaching a preset maximum value for Δ denoted as $\overline{\Delta}$:

$$\Delta := \min\{2 \cdot \Delta, \overline{\Delta}\} \quad . \tag{2.46}$$

Similar to trust region algorithms, ***S**equential **Q**uadratic **P**rogramming* (SQP) methods minimize the second order Taylor approximation $m(\boldsymbol{d})$ of the change in the original objective function $\Delta f(\boldsymbol{d})$ according to Eq. 2.38. Advantageous is the convex form of $m(\boldsymbol{x})$ for any type of $f(\boldsymbol{x})$. Variable limits are taken into account via inequality constraints in the way shown in condition 2.8 and 2.9. The whole so-called quadratic sub-problem becomes

$$\begin{aligned} \min_{\boldsymbol{d}} \quad & \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d} + \frac{1}{2} \cdot \boldsymbol{d}^T \cdot \boldsymbol{H}(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d} \\ s.t. \quad & \boldsymbol{h}(\boldsymbol{x}^{(k)} + \boldsymbol{d}) = \boldsymbol{0} \\ & \boldsymbol{g}(\boldsymbol{x}^{(k)} + \boldsymbol{d}) \geq \boldsymbol{0} \quad . \end{aligned} \tag{2.47}$$

Note that inequalities are only greater than or equal to zero in Eq. 2.47. The inequality constraints are divided into active constraints $\boldsymbol{g}^A(\boldsymbol{x})$ where the current iteration point $\boldsymbol{x}^{(k)}$ sits on their boundary, i.e., a constraint g_j fulfills

$$g_j(\boldsymbol{x}^{(k)}) = 0 \quad , \tag{2.48}$$

and inactive constraints $\boldsymbol{g}^{IA}(\boldsymbol{x})$ where the constraint does not affect the current iterate of the sub-problem. Solving a root-finding problem by least-squares minimization in a given variable domain is a special case of Eq. 2.47, in which only already linear inequalities occur given by the variable bounds. They become active whenever the iterate hits their associated variable bounds. No step into their direction $\boldsymbol{d}^A$ is taken in the upcoming iteration steps until their activity status changes again. Hence, Eq. 2.47 becomes

$$\begin{aligned} \min_{\boldsymbol{d}=\boldsymbol{d}^A \cup \boldsymbol{d}^{IA}} \quad & \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d} + \frac{1}{2} \cdot \boldsymbol{d}^T \cdot \boldsymbol{H}(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d} \\ s.t. \quad & \boldsymbol{d}^A = \boldsymbol{0} \\ & \boldsymbol{g}^{IA}(\boldsymbol{x}^{(k)} + \boldsymbol{d}^{IA}) > \boldsymbol{0} \quad . \end{aligned} \tag{2.49}$$

A SQP algorithm now solves this quadratic problem using (Quasi-)Newton and step-size control mechanisms to keep the iteration points within the feasible region. The iteration continues until the iteration step falls below the required tolerance and a local minimum is found. The latter is checked by the Lagrange multipliers of the active constraints $\boldsymbol{\mu}^A$, which need to be greater than or equal to zero. An active constraint with a negative multiplier implies that the objective can be further reduced in its feasible direction. Hence, the constraint is deactivated and a new sub-problem is generated. SQP algorithms are particularly suitable in case a minimum is located at one or more variable bounds. However, they only work, if a feasible initial point is known, i.e., a point where all functional terms of Eq. 2.47 are defined. They can be slow, whenever the sub-problem needs to be frequently updated due to changes from inactive to active constraints or vice versa (Nocedal and Wright, 2005, pp. 560-561).

Finally, an NLE, reformulated into a least-squares minimization problem according to Eq. 2.13, can be solved by a barrier-interior-point algorithm. Generally, the latter extends an objective function $f(\boldsymbol{x})$ by a barrier term $b(\boldsymbol{x})$ so that the constrained minimization problem has the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) + \mu \cdot b(\boldsymbol{x}) \\ s.t. \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0} \\ & \boldsymbol{x} > \boldsymbol{0} \quad . \end{aligned} \tag{2.50}$$

A typical choice for $b(\boldsymbol{x})$ is

$$b(\boldsymbol{x}) := -\sum_{i=1}^{n} \ln x_i \quad , \tag{2.51}$$

which increases the objective towards infinity whenever an x_i of $\boldsymbol{x}$ gets close to zero, i.e., iteration points are driven away from the boundaries. The barrier term also convexifies the objective in this way, which is more pronounced the larger the scalar barrier parameter μ is. In the special case of solving an NLE in given

variable bounds as least-squares minimization problem Eq. 2.50 becomes

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) - \mu \cdot \sum_{i=1}^{n} \ln\left(x_i - \underline{x}_i\right) - \mu \cdot \sum_{i=1}^{n} \ln\left(\overline{x}_i - x_i\right) \\ s.t. \quad & \boldsymbol{x} - \underline{\boldsymbol{x}} > \mathbf{0} \\ & \overline{\boldsymbol{x}} - \boldsymbol{x} > \mathbf{0} \quad . \end{aligned} \tag{2.52}$$

Eq. 2.50 is then solved by an inner iteration for a fixed μ, the so-called barrier problem, in which a (Quasi-)Newton method with some step-size control mechanism searches for a local minimum fulfilling a relaxed version of the Karush-Kuhn-Tucker conditions (Eq. 2.31). The inner iteration is followed by an outer iteration, in which the optimality conditions for the original problem at the current barrier problem's solution are checked. If they are fulfilled in a preset tolerance the algorithm stops. Otherwise, it decreases μ and solves the next barrier problem. Several techniques exist to successively decrease μ. The Fiacco-McCormicks approach decreases μ monotonously by

$$\mu^{(k+1)} = \sigma^{(k)} \cdot \mu^{(k)}, \qquad 0 < \sigma < 1 \quad , \tag{2.53}$$

for a constant scalar σ. In Nocedal et al. (2009), adaptive approaches are presented and studied, in which μ is already modified during the optimization of the barrier problem. Basically these methods aim to decrease μ more strongly, whenever progress in the minimization problem is high and dampen it otherwise. The applied interior-point solvers in Nocedal et al. (2009) can find the solutions of the studied problems much faster than the Fiacco-McCormicks approach. Barrier interior-point methods are very well suitable, if the optimum lies in the interior of the variable intervals. If this is not the case for some variables, the algorithm may need a very long time to sufficiently reduce μ before a required tolerance is accomplished (Nocedal and Wright, 2005, p. 593). More about SQP and interior-point algorithms can be found in Biegler (2010, pp. 135-179).

In order to solve linear equation systems of the (Quasi-)Newton steps efficiently, various linear solvers exist, whose discussion goes beyond the scope of this work. Two established methods are the ***L**ower **U**pper* (LU) decomposition and Gauss elimination, which are explained in detail in Dahmen & Reusken (2008, pp. 68-82).

For large, sparse linear systems there are moreover further enhancements of the Gauss elimination for example the MA27 solver (Duff et al., 2017a, p. 236).

2.4 Preconditioning

Every iteration step of a Newton-based numerical iteration method consists of solving a linear system of the form

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b} \quad , \tag{2.54}$$

whereas $\boldsymbol{A}$ equals the current Jacobian matrix, $\boldsymbol{x}$ the Newton step and $\boldsymbol{b}$ the negative vector of the current function residuals.

The application of numerical iteration algorithms is always error-prone, in other words there is always a discrepancy between the exact solution of a mathematical problem and the computed one. Three major types of errors are responsible, namely round-off errors, inaccuracies in the approximating algorithm and errors in the input data the problem is initialized with, i.e., if $\boldsymbol{x}$ is searched for in Eq. 2.54, but $\boldsymbol{A}$ and $\boldsymbol{b}$ already contain errors, e.g., due to round-off errors in their computation. The first two types are associated with the algorithm and further examined in a stability analysis. An algorithm is termed stable if the order of the error it generates stays within the range of the error related to the input data. The influence of the error-prone input data on the problem is investigated in condition analysis and is only problem-dependent. It is assumed that the state-of-the-art solvers applied in this research work are all stable and since no new real-valued numerical iteration method is developed, stability will not be further discussed. More about the stability of algorithms can be found in Dahmen & Reusken (2008, pp. 42-48). However, the condition analysis plays an important role in this thesis, because it greatly depends on the problem formulation itself. An ill-conditioned system can cause diverging iteration sequences, if for example the Newton step can not be accurately calculated. Problems are termed well-conditioned as long as the error of the output data is not much higher than the one of the input data, ideally, the error does not increase at all. Referring to linear systems of the form 2.54, the exact input data is $\boldsymbol{A}$ and $\boldsymbol{b}$, both associated with an error of $\boldsymbol{\Delta A}$ and $\boldsymbol{\Delta b}$

respectively. The output data is $\boldsymbol{x}$ with an error of $\Delta\boldsymbol{x}$. To quantify the error $\Delta\boldsymbol{x}$ that results from perturbed input data, $\Delta\boldsymbol{A}$ shall be initially neglected. Then it follows

$$\Delta\boldsymbol{x} = \boldsymbol{A}^{-1} \cdot \Delta\boldsymbol{b} \quad . \tag{2.55}$$

Using any of the p-norms presented in table 2.1 and the triangle inequality, one can also overestimate the relative error $\frac{\|\Delta\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}$ of $\boldsymbol{x}$ by

$$\frac{\|\Delta\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p} \leq \overbrace{\|\boldsymbol{A}^{-1}\|_p \cdot \|\boldsymbol{A}\|_p}^{:=\kappa_p(\boldsymbol{A})} \cdot \frac{\|\Delta\boldsymbol{b}\|_p}{\|\boldsymbol{b}\|_p} \quad . \tag{2.56}$$

In this inequality $\kappa_p(\boldsymbol{A})$ is called the relative condition number of matrix $\boldsymbol{A}$ in norm p (Dahmen and Reusken, 2008, p. 59), which is basically the enhancement factor from the relative errors of $\boldsymbol{b}$ to the one of $\boldsymbol{x}$. The relative condition number can never be lower than 1

$$1 \leq \kappa_p(\boldsymbol{A}) \quad . \tag{2.57}$$

The lower it is, the better a system is conditioned. Hence, the perfect value is 1, in which case no error enhancement occurs. The identity matrix for example has a condition number of 1. However, if $\boldsymbol{A}$ is perturbed, Eq. 2.54 turns into

$$(\boldsymbol{A} + \Delta\boldsymbol{A}) \cdot (\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b} \quad , \tag{2.58}$$

from which the following overestimation for $\frac{\|\Delta\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p}$ of $\boldsymbol{x}$ is deduced

$$\frac{\|\Delta\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p} \leq \frac{\kappa_p(\boldsymbol{A})}{1 - \kappa_p(\boldsymbol{A}) \cdot \frac{\|\Delta\boldsymbol{A}\|_p}{\|\boldsymbol{A}\|_p}} \cdot \left(\frac{\|\Delta\boldsymbol{A}\|_p}{\|\boldsymbol{A}\|_p} + \frac{\|\Delta\boldsymbol{b}\|_p}{\|\boldsymbol{b}\|_p} \right) \quad \text{for } \kappa_p(\boldsymbol{A}) \cdot \frac{\|\Delta\boldsymbol{A}\|_p}{\|\boldsymbol{A}\|_p} < 1 \quad . \tag{2.59}$$

The exact derivation of condition 2.59 from 2.58 is given in Dahmen & Reusken (2008, pp. 59-60). Note if $\frac{\|\Delta\boldsymbol{A}\|_p}{\|\boldsymbol{A}\|_p} \ll \frac{\|\Delta\boldsymbol{b}\|_p}{\|\boldsymbol{b}\|_p}$, condition 2.59 becomes equal to 2.56. To check the accuracy of a numerically determined solution $\boldsymbol{x}$ of Eq. 2.54, Dahmen & Reusken (2008, p. 61) conclude the following condition, which bounds the

Tab. 2.1: Important p-norms for real-valued n-dimensional vectors $\boldsymbol{x} := (x_1, \dots x_i, \dots, x_n)^T$ and $m \times n$-dimensional matrices $\boldsymbol{A}$ with elements $a_{j,i}$, row indices $j = 1 \dots m$ and column indices $i = 1 \dots n$.

p	Vector norm	Induced matrix norm
1	$\sum_{i=1}^{n} \lvert x_i \rvert$	$\max_{i=1\dots n} \sum_{j=1}^{m} \lvert a_{j,i} \rvert$
2	$\left(\sum_{i=1}^{n} \lvert x_i \rvert^2\right)^{0.5}$	$\max_{i=1\dots n} \sigma_i$
∞	$\max_{i=1\dots n} \lvert x_i \rvert$	$\max_{j=1\dots m} \sum_{i=1}^{n} \lvert a_{j,i} \rvert$

maximum relative error to the exact solution $\boldsymbol{x}^*$:

$$\frac{\|\boldsymbol{x} - \boldsymbol{x}^*\|_p}{\|\boldsymbol{x}\|_p} \leq \kappa_p(\boldsymbol{A}) \cdot \frac{\overbrace{\|\boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x}\|_p}^{:=\boldsymbol{r}}}{\|\boldsymbol{b}\|_p} \quad , \tag{2.60}$$

with $\boldsymbol{r}$ being the residual vector. In consequence, if the entries of the residual vector are close to 0 this does not have to apply for the distance between computed and exact solution in case of a large condition number. Hence, the computed solution can be quite inaccurate. A rule of thumb states that for $\kappa_p(A) \approx 10^k$, k digits of accuracy are lost in the computed solution (Watkins, 2002, p. 165).

Regarding the vector and matrix norms the $p = 2$-norm is used throughout this work. The related relative condition number of $\boldsymbol{A}$ according to table 2.1 becomes

$$\kappa_2(\boldsymbol{A}) := \max_{i=1\dots n} \sigma_i(\boldsymbol{A}^{-1}) \cdot \max_{i=1\dots n} \sigma_i(\boldsymbol{A}) = \frac{\max_{i=1\dots n} \sigma_i(\boldsymbol{A})}{\min_{i=1\dots n} \sigma_i(\boldsymbol{A})} \quad . \tag{2.61}$$

The singular values σ_i are determined by a singular value decomposition, which is explained in Gander et al. (2014, pp. 269-280) next to the derivation of Eq. 2.61. Note that singular matrices have singular values equal to zero and hence their condition number becomes infinite. In this work, the condition number is used to determine the condition of NLEs, or to be more precise, their linear approximations at iteration points and their solutions. In the end, this quantity is used to check whether a preconditioning strategy, such as the scaling or decomposition

methods presented in the next sections, can successfully improve the condition of an NLE.

2.4.1 Scaling

Real world problems are associated with a broad range of different scales. Especially in chemical engineering processes, variable values may range from micro scale transport phenomena such as diffusive mass transfer up to macro scale models such as energy and material balances. For example, diffusion coefficients in liquids are typically of the order 10^{-9} $m^2\,s^{-1}$, while the heat released when 1000 kg of water condenses at atmospheric pressure is about 2.35×10^9 J. Hence, a process model that relies on both, encompasses about 19 orders of magnitude in the decimal system using SI units. These different orders of magnitude often cause a broad range of orders in the non-zero entries of the Jacobian matrix as well, with the consequence that elements of low order become almost zero. When such elements lie on the diagonal or their related rows become structurally identical to another row due to their disappearance, the Jacobian matrix is singular or nearly singular and the overall system becomes ill-conditioned. Through row and column scaling of the matrix $\boldsymbol{A}$, and the vectors $\boldsymbol{x}$ and $\boldsymbol{b}$, it is possible to improve the condition. This is demonstrated with the following two-dimensional system

$$\begin{bmatrix} 15 & 1000 \\ 0 & 2 \end{bmatrix} \cdot \boldsymbol{x} = \begin{pmatrix} 100 \\ 1 \end{pmatrix} \quad , \tag{2.62}$$

which has a condition number of $\kappa_2(A) = 3.33^4$ and the actual solution $\boldsymbol{x}^* = (\frac{80}{3};\ 0.5)^T$. This rather large condition number results from the fact that the rows are structurally quite similar, because 15 is much smaller than 1000 just as 0 is much smaller than 2. Through row scaling the entries can be equilibrated. Generally, row scaling involves a matrix multiplication on each side of Eq. 2.54

$$\boldsymbol{D}_R \cdot \boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{D}_R \cdot \boldsymbol{b} \quad . \tag{2.63}$$

Exemplarily, a method is shown that Eq. 2.62 can be row-scaled with. Here, row scaling refers to the vector norm $p = 1$ from table 2.1 applied on each row vector of $\boldsymbol{A}$. The diagonal row scaling matrix $\boldsymbol{D}_R$ looks as follows

$$D_R := \begin{bmatrix} d_{R,1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & d_{R,n} \end{bmatrix} \quad \text{, with } d_{R,j} = min\{\frac{1}{\sum_{i=1}^{n} |a_{j,i}|}; \frac{1}{|b_j|}\} \quad . \tag{2.64}$$

The sum of the absolute values of all entries $a_{j,:}$ in row j is calculated and compared with the absolute value of b_j. All $a_{j,:}$ and b_j are then divided by the larger value of both so that none of them is greater than one and lower than minus one. In consequence, the scaled version of Eq. 2.62 becomes

$$\begin{bmatrix} 0.0148 & 0.9852 \\ 0 & 1 \end{bmatrix} \cdot \boldsymbol{x} = \begin{pmatrix} 9.852 \\ 0.5 \end{pmatrix} \quad \text{, with } D_R := \begin{bmatrix} \frac{1}{1015} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \tag{2.65}$$

and a condition number of $\kappa_2(D_R \cdot A) = 133.35$. Another way of scaling is column scaling, which is essentially a unit transformation of the variables. A linear system with n variables to solve is then

$$D_C := \begin{bmatrix} d_{C,1}(x_1) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & d_{C,n}(x_n) \end{bmatrix} \tag{2.66}$$

$$\boldsymbol{A} \cdot \boldsymbol{D}_C \cdot \overbrace{(\boldsymbol{D}_C^{-1} \cdot \boldsymbol{x})}^{:=\boldsymbol{y}} = \boldsymbol{b} \quad , \tag{2.67}$$

with $d_{C,i}$ being the conversion factor of the i-th variable for $i = 1 \dots n$. The vector $\boldsymbol{y}$ equals the converted variable entries, which are used in the scaled linear system.

The values in the actual units can be simply determined by $\boldsymbol{x} = \boldsymbol{D_C} \cdot \boldsymbol{y}$. Example 2.62 can also be column-scaled for instance if the order of magnitude referring to the decimal system of the solution is already known. Assuming this is here the case then the converted linear system becomes

$$\begin{bmatrix} 150 & 100 \\ 0 & 0.2 \end{bmatrix} \cdot \boldsymbol{y} = \begin{pmatrix} 100 \\ 1 \end{pmatrix} \quad \text{with } D_C := \begin{bmatrix} 10 & 0 \\ 0 & \frac{1}{10} \end{bmatrix} \quad . \tag{2.68}$$

The condition number is $\kappa_2(A \cdot D_c) = 1083.33$, hence, in this case, less well-conditioned than after row scaling. One sees that the first row's entries have been equilibrated but the second row's entry is much smaller and the entire row tends to be a row with near zero entries compared to the first one. Through additional row scaling the rows are equilibrated as well and the condition number decreases to 4.44. The corresponding system is

$$\begin{bmatrix} 0.6 & 0.4 \\ 0 & 0.4 \end{bmatrix} \cdot \boldsymbol{y} = \begin{pmatrix} 0.4 \\ 2 \end{pmatrix} \quad \text{with } \boldsymbol{y} = \begin{pmatrix} \frac{8}{3} \\ 5 \end{pmatrix} \text{ and } \boldsymbol{x} = \begin{bmatrix} 10 & 0 \\ 0 & \frac{1}{10} \end{bmatrix} \cdot \boldsymbol{y} \quad . \tag{2.69}$$

If the variables' orders of magnitudes are not known a priori, there are also automatic scaling algorithms available for the equilibration of matrix entries. Two of them can be used in the numerical iteration process of the suggested hybrid approach, namely the routines from Curtis & Reid (1972) and Knight et al. (2014). The first one minimizes the expression

$$\min_{\rho_j, \gamma_i} \sum_{a_{j,i} \neq 0} (log|a_{j,i}| - \rho_j - \gamma_i)^2 \quad \text{with } i,j = 1 \dots n \quad , \tag{2.70}$$

with $D_R = diag(e^{-\rho_j})$ and $D_C = diag(e^{-\gamma_i})$ so that all absolute values of the entries in $\boldsymbol{A}$ become close to one. The second method is iterative and uses the norm $p = \infty$ on row and column vectors in the calculation specification

$$a_{j,i}^{(k+1)} := \|a_{j,:}^{(k)}\|_\infty^{-0.5} \cdot a_{j,i}^{(k)} \cdot \|a_{:,i}^{(k)}\|_\infty^{-0.5} \quad . \tag{2.71}$$

Every row and column vector converges to one in the ∞-norm, which is proven in Ruiz (2001). Hence, all entries of $\boldsymbol{A}$ are again within the range $[-1, 1]$.

2.4.2 Decomposition

A matrix $\boldsymbol{A}$ from a linear system of 2.54 can be permuted to a block triangular form

$$\boldsymbol{P} \cdot \boldsymbol{A} \cdot \boldsymbol{Q} := \begin{bmatrix} \boldsymbol{B}_{1,1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ \boldsymbol{B}_{n,1} & \dots & \boldsymbol{B}_{n,n} \end{bmatrix} \quad . \tag{2.72}$$

with $\boldsymbol{B}_{i,i}$ being irreducible subblock matrices that either can not be further subdivided into even smaller matrices or have been permuted as well into a block triangular form by a so-called fine decomposition. Formally, the permutation equals two matrix multiplications $\boldsymbol{P} \cdot \boldsymbol{A} \cdot \boldsymbol{Q}$, where $\boldsymbol{P}$ is responsible for the row and $\boldsymbol{Q}$ for the column permutation of $\boldsymbol{A}$. The linear system then becomes

$$\underbrace{(\boldsymbol{P} \cdot \boldsymbol{A} \cdot \boldsymbol{Q})}_{:=B} \cdot \overbrace{\boldsymbol{Q}^{-1} \cdot \boldsymbol{x}}^{:=\boldsymbol{y}} = \boldsymbol{P} \cdot \boldsymbol{b} \quad . \tag{2.73}$$

Several algorithms exist to bring $\boldsymbol{A}$ into a block triangular form. Duff et al. (2017a, pp. 108-136) present a thorough overview on established methods. Among them, we shall focus on the ***D**ulmage **M**endelsohn* (DM) decomposition that was first introduced in Dulmage & Mendelsohn (1958) and comes from the theory of bipartite graphs. A bipartite graph is a mapping of two independent sets of elements known as vertices, where each vertex from one set can be assigned to at least one vertex from the other set and vice versa. Such an assignment is termed an edge. The method aims to find a maximum matching between rows and columns of $\boldsymbol{A}$. A matching is defined as an ordering of a specific row (vertex from set of rows) to a specific column (vertex from set of columns) based on non-zero elements in $\boldsymbol{A}$ (edges), i.e., the linear function related to the row can be applied to solve the corresponding variable associated with the column. A

maximum matching in this sense means, finding the maximum amount of 1-to-1 orderings between rows and columns. In well-determined, nonsingular systems of dimension n, $\boldsymbol{A}$ is square, and n matchings can be found in a maximum matching. In rectangular or singular systems not every row or column can be sorted to a corresponding column or row, some of them will remain unmatched. The number of maximum matchings is therefore lower than the dimension of $\boldsymbol{A}$. The coarse DM decomposition is especially useful for the latter type of systems. After finding the maximum matching it starts with the unmatched rows and sorts them to the upper corner of the permuted matrix. For an $m \times n$ matrix this equals the rows $(m - l + n_1)$ presented in figure 2.2. Columns that share non-zero elements with such rows are sorted to the left so that all together they build the first block $(m - l + n_1) \times n_1$. Next, unmatched columns are permuted to the right, and rows, in which they have non-zero elements, are sorted to the bottom and build the last block $m_1 \times (m_1 + n - l)$. The middle block $(l - m_1 - n_1) \times (l - m_1 - n_1)$ is a square, well-determined block with a maximum matching. In the fine DM decomposition all three blocks are individually brought into a block triangular structure. The problems considered in this work only belong to the type of well-determined, square systems so that the coarse decomposition will not be used and the fine decomposition is of greater importance. Nevertheless, if a square system is singular the number of matchings in a maximum matching will be lower than the system's dimension and the same kind of structure as in the coarse decomposition will appear. This is useful for troubleshooting of models as over-determined parts can then be easily identified in the upper left part (too many equations for too few variables) and under-determined parts in the bottom right block (too few equations for too many variables).

The fine DM decomposition produces on a well-determined equation system, i.e., as many unknown variables as linear independent equations, a block triangular form with non-zero diagonal elements as shown in figure 2.3a. Each square block on the diagonal equals an independent sub-problem. Hence, the numerical solver starts in the top left and processes successively each of such blocks. The variables associated with the leftmost block on the diagonal are solved first. Variables that have been solved already are set to their solutions and represent constants in the subsequent blocks on the diagonal. Hence, the solution space can be greatly decreased, in case some variables can be removed during this process. Solving a

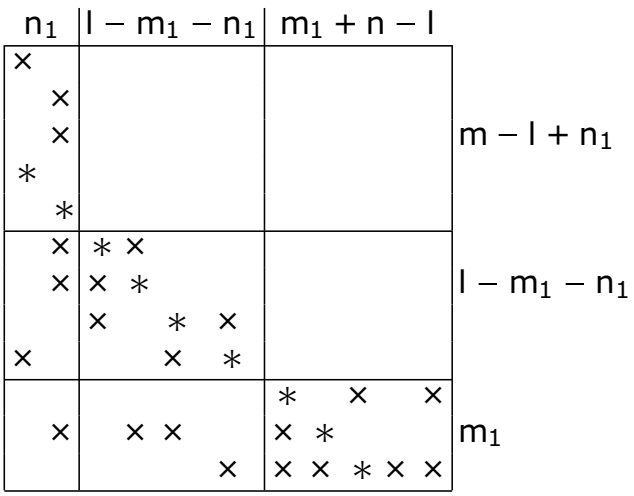
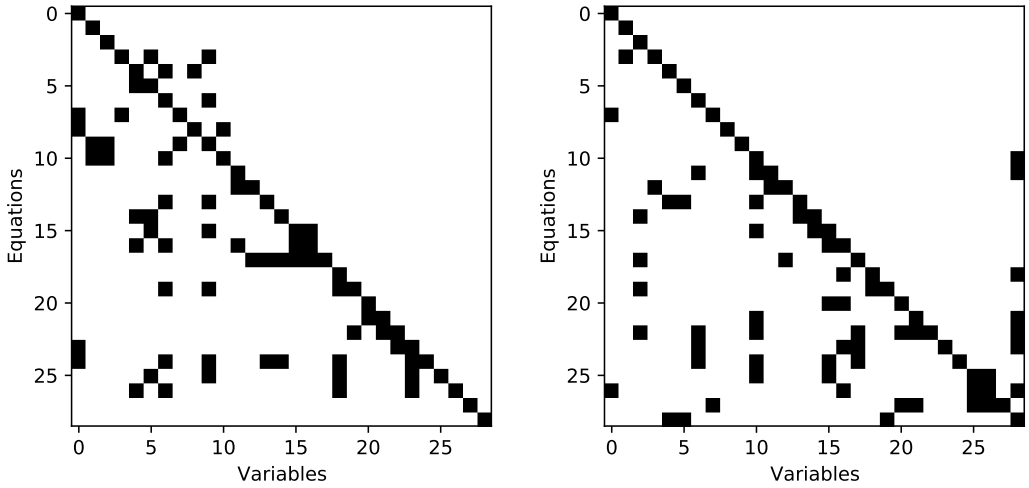


Fig. 2.2: Block triangular structure after coarse DM decomposition, from Duff et al. (2017a, p. 134).

large $n \times n$ system of linear equations by LU decomposition for example requires about $2 \cdot \frac{n^3}{3}$ floating point operations, if it is solved as a dense system (Boyd and Vandenberghe, 2013, p. 668). Assuming the system could be decomposed into two $\frac{n}{2} \times \frac{n}{2}$ subblocks and solving them sequentially reduces the computational costs down to 25 % ($4 \cdot \frac{(n/2)^3}{3} = \frac{n^3}{6} = \frac{1}{4} \cdot \left(2 \cdot \frac{n^3}{3}\right)$). In Bublitz et al. (2017a) it was demonstrated for two chemical process models, namely a flash unit and an absorption column, that solving a decomposed system decreases the computational time of the numerical iteration by a significant extend compared to the original problem. On top, the condition of the individual sub-problems improved, most likely because functions and variables of certain sub-problems rather possess similar orders of magnitude. An additional advantage of this permutation is that some subblocks might be completely independent from each other, for example subblocks 1 to 3 in figure 2.3a. They can actually be solved in parallel and further decrease the computational time. We reference in the following text always to fine Dulmage-Mendeslohn decomposition, whenever writing DM.

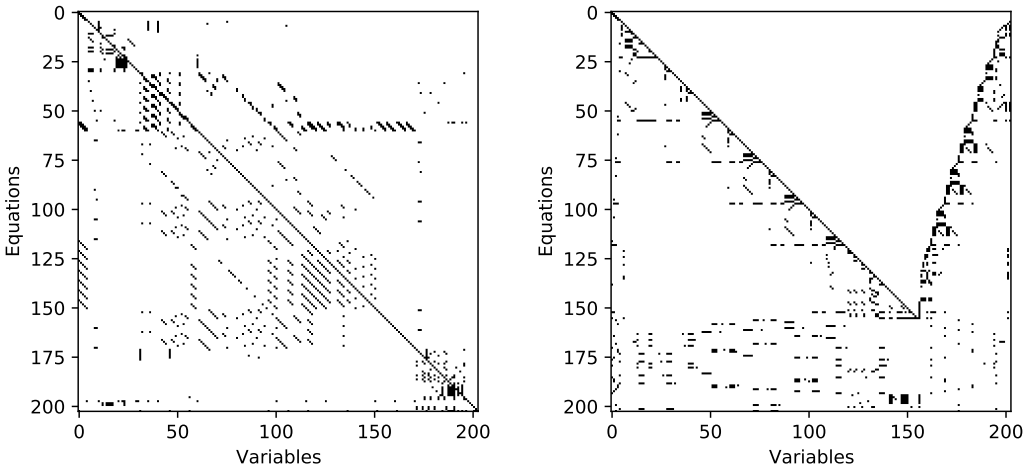
In DM, some subblocks of large dimension often persist, for example the fourth block in figure 2.3a. They may remain difficult to solve. However, the class of ***B**ordered **B**lock **T**riangular **F**orm*s (BBTFs) methods is able to decrease the size of such diagonal blocks further by moving some columns of the matrix, which hold



(a) Block triangular decomposition. **(b)** Bordered block triangular transformation.

Fig. 2.3: Permuted Jacobian incidence matrices of the flash unit model described in section 4.3. An analysis on solving these systems numerically is given in Bublitz et al. (2017a).

many row entries, to the right-hand side. In this way, it excludes such columns from the sequence of blocks on the diagonal and only small, dense blocks remain. Figure 2.3 shows the difference for the same system between its block triangular form in figure 2.3a and its bordered block triangular form in Fig 2.3b. The latter moves here just one column to the right. In this way, all diagonal blocks, except for one dense 2×2 block, reduce to 1×1 subsystems. The columns that are moved to the right together with their matched rows at the bottom build the border. Figure 2.4 shows the resulting matrix forms from DM and BBTF applied on a rather large equation system. It can be seen that BBTF keeps the dimension of the diagonal blocks small in contrast to the DM decomposed form. An efficient algorithm to get a matrix into this form is the Hellerman-Rarick algorithm that was first introduced in Hellerman & Rarick (1971) and is nowadays known under the designation $P3$. The current version used in this work is named $P5$ and refers to the advancement of Erisman et al. (1985). This method ensures that the diagonal elements are non-zero, which $P3$ could not. Hence, structural stability is given in $P5$. Details about the actual ordering algorithm can be found in Erisman et al. (1985) and Duff et al. (2017b). BBTF seems to be favorable concerning the computational work because of the low dimensional subblocks that it creates. One idea is to fix



(a) Block triangular decomposition. **(b)** Bordered block triangular transformation.

Fig. 2.4: Permuted 203 × 203 Jacobian incidence matrices of the distillation column model described in section 4.5 consisting of five separation trays and five components.

in a first step all variables associated with columns of the border and solve the diagonal's blocks sequentially in the same manner as it was suggested for block triangular matrices. In a second step, the variables of the border are solved in their corresponding border functions, at the numerically determined values of the other variables from the previous step. Next, the variables of the border are updated, and the first step is repeated. The procedure continues until the final solution is found. In Bublitz et al. (2017a) it was demonstrated that this procedure is indeed faster than solving the sequence of sub-problems resulting from a DM decomposed system, if the variables of the border are initialized with values not far from the actual solution. Nevertheless, except for these close-by initial points, the strategy suffers from numerical instabilities as was also concluded by Baharev (2016) and Duff et al. (2017b). No special strategy exists up to now to ensure that the pivot elements are far away from zero in order to achieve a better conditioned system. If this issue can be solved in future, this strategy seems to be even more suitable for parallelization than the DM decomposition. The BBTF algorithm cannot only be used for solving but also for determining the variables of the border. Hereafter they are called tearing variables, because they literally tear a large equation system apart so that a sequence of small independent subproblems can be solved. Thus, the tearing variables also represent an important connection

between the subproblems and shall be further investigated within the scope of this work.

2.5 Presolve

By presolve, an additional step prior to the numerical iteration of optimization problems of type 2.4 is meant, in which the latter are simplified with the help of preprocessing algorithms. For this purpose, software such as AMPL offers optional strategies to speed up the iteration process and/or increase its robustness. AMPL starts with the elimination of linear constraints that depend on only one variable and whose solution can thus be uniquely determined. In the entire equation system, all instances of such a variable are replaced by their solutions, i.e., the model is reduced by one dimension per variable. Secondly, linear constraints of the system are used to further tighten initial variable intervals. Finally, constraints that are always fulfilled, i.e., cannot further restrict any variable intervals are removed. More detailed explanations and examples of this methodology and its options can be found in Fourer et al. (2009, pp. 275-282).

2.6 Interval Arithmetic

As Moore et al. (2009, p. ix) wrote "with interval computation we can program a computer to find intervals that contain - with absolute certainty - the exact answer to various mathematical problems". With respect to NLEs it is possible to find all of their numerical solutions within given variable domains. This can be done by the use of contracting operators $\Gamma\left(\boldsymbol{f}(\boldsymbol{x}),\overline{\underline{\boldsymbol{x}}}\right)$ that try to remove all parts of the box where variable intervals have no feasible value in one or more other variable intervals to solve

$$\boldsymbol{f}(\boldsymbol{x}) = 0 \quad . \tag{2.74}$$

These removed (sub)boxes are termed inconsistent. In a constraint-propagation scheme, the variable intervals are successively reduced until the box is either empty, solved or consistent.Various contracting techniques exist. The focus in the following sections is put on the five applied in this thesis, which are the interval

Newton $\Gamma_{nwt}\left(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{x}_c,\underline{\overline{\boldsymbol{x}}}\right)$ that additionally requires a point of expansion $\boldsymbol{x}_c \in \underline{\overline{\boldsymbol{x}}}$, hull consistency $\Gamma_{hc}\left(\boldsymbol{f}(\boldsymbol{x}),\underline{\overline{\boldsymbol{x}}}\right)$, box consistency $\Gamma_{bc}\left(\boldsymbol{f}(\boldsymbol{x}),\underline{\overline{\boldsymbol{x}}}\right)$, cutting $\Gamma_C\left(\boldsymbol{f}(\boldsymbol{x}),\underline{\overline{\boldsymbol{x}}}\right)$, and splitting $\Gamma_S\left(\boldsymbol{f}(\boldsymbol{x}),\underline{\overline{\boldsymbol{x}}}\right)$ operators. Beforehand, a short introduction to general ***I**nterval **A**rithmetic* (IA) operations is given that are further explained in the book of Moore et al. (2009, pp. 10-50).

2.6.1 General Interval Arithmetic

Operations in real-valued algebra are also defined in IA. Indeed, IA's operations are extensions of the real-valued case. For degenerate intervals, i.e., intervals with zero width, they are equal. Generally, the elementary operations (Addition, Subtraction, Multiplication, Division) of two intervals $\underline{\overline{x}}$, $\underline{\overline{y}}$ can be expressed by

$$\underline{\overline{x}} \odot \underline{\overline{y}} := \{x \odot y : x \in \underline{\overline{x}}, y \in \underline{\overline{y}}\} \quad \odot \widehat{=} +, -, \cdot, or / \quad , \tag{2.75}$$

while for the division this holds only for $0 \notin \underline{\overline{y}}$. If $0 \in \underline{\overline{y}}$ the resulting interval will be $[-\inf, \inf]$. Nevertheless, an operator shall be introduced later to gain some tighter enclosures in this case. Table 2.2 shows the specific rules for each elementary operation. It also contains rules for the exponent, logarithm and general power functions. The latter can also be used for the n-th root $\sqrt[n]{\underline{\overline{x}}} \equiv \underline{\overline{x}}^{\frac{1}{n}}$. Ranges for such unary, monotonic functions can be easily determined by real arithmetic operations as their minimum and maximum value are related to the lower and upper bound values of $\underline{\overline{x}}$. Similar rules exist for partially monotonic functions such as the trigonometric functions (Sine, Cosine, Tangent, Cotangent, Secant, Cosecant), which can be found in mpmath's documentation (Johansson, 2010) that is the python package used for IA here.

Tab. 2.2: Elementary operations and some univariate, monotonic functions in IA.

Operation		**Resulting Interval**
$\underline{\overline{x}} + \underline{\overline{y}}$	$:=$	$[\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
$\underline{\overline{x}} - \underline{\overline{y}}$	$:=$	$[\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

$$
\begin{array}{lll}
\overline{\underline{x}} \cdot \overline{\underline{y}} & := & [\min p,\ \max p] \\
 & & p = \{\underline{x} \cdot \underline{y},\ \underline{x} \cdot \overline{y},\ \overline{x} \cdot \underline{y},\ \overline{x} \cdot \overline{y}\} \\
\overline{\underline{x}} / \overline{\underline{y}} & := & [\min q,\ \max q], \quad 0 \notin \overline{\underline{y}} \\
 & & q = \{\underline{x}/\underline{y},\ \underline{x}/\overline{y},\ \overline{x}/\underline{y},\ \overline{x}/\overline{y}\} \\
 & & [-\inf,\ \inf], \quad 0 \in \overline{\underline{y}} \\
\exp \overline{\underline{x}} & := & [\exp \underline{x},\ \exp \overline{x}] \\
\log \overline{\underline{x}} & := & [\log \underline{x},\ \log \overline{x}], \quad \underline{x} > 0 \\
\overline{\underline{x}}^n & := & [\underline{x}^n, \overline{x}^n], \quad \underline{x} > 0 \text{ or n is not even} \\
 & & [\overline{x}^n, \underline{x}^n], \quad \overline{x} < 0 \text{ and n is even} \\
 & & [0, |\overline{\underline{x}}|^n], \quad 0 \in \overline{\underline{x}} \text{ and n is even}
\end{array}
$$

While the addition of 0 and multiplication with 1 do not change an interval, the multiplication with -1 is usually not its additive inverse. Hence, as already shown in Eq. 2.2 and Eq. 2.3 the subtraction of an interval from itself equals 0 only if the interval is degenerate, when interval and real arithmetic become equivalent. IA is associative and commutative but only distributive in the real case

$$x \cdot (y + z) = x \cdot y + x \cdot z \quad . \tag{2.76}$$

For intervals $\overline{\underline{x}} = [0, 1], \overline{\underline{y}} = [1, 2], \overline{\underline{z}} = [-2, 3]$ the interval extensions of the left and right term of this equation are

$$\overline{\underline{x}} \cdot (\overline{\underline{y}} + \overline{\underline{z}}) = [0, 1] \cdot ([1, 2] + [-2, 3]) = [-1, 5] \tag{2.77}$$

$$\overline{\underline{x}} \cdot \overline{\underline{y}} + \overline{\underline{x}} \cdot \overline{\underline{z}} = [0, 1] \cdot [1, 2] + [0, 1] \cdot [-2, 3] = [-2, 5] \quad . \tag{2.78}$$

Hence, the formulation of the expression has a great influence on the tightness of the resulting interval here. The reason for this is the so-called interval dependency problem. To determine the lower bound of Eq. 2.78, the value in $\overline{\underline{x}}$ differs for both

$\overline{x}$ instances, while in real-valued arithmetic their values must be the same

$$\underline{f}(\overline{x},\overline{y},\overline{z}) = \underline{x} \cdot \underline{y} + \overline{x} \cdot \underline{z} = -2 \quad . \tag{2.79}$$

In order to obtain tight enclosures, one should try to avoid multiple variable occurrences in expressions to prevent interval dependency. This is also true for fractions where a variable is part of both nominator and denominator, here shown for $\overline{x} = [0, 1], \overline{y} = [1, 2]$

$$\frac{\overline{x}}{\overline{x} + \overline{y}} = \frac{[0, 1]}{[0, 1] + [1, 2]} = [0, 1] \tag{2.80}$$

that can be easily rearranged to

$$\frac{1}{1 + \frac{\overline{y}}{\overline{x}}} = \frac{1}{1 + \frac{[0, 1]}{[1, 2]}} = [0.5, 1] \quad . \tag{2.81}$$

The general conclusion that can be drawn from this example is that as long as all variables occur only once in a function $f(\boldsymbol{x})$

$$\overline{\underline{f}}(\overline{\boldsymbol{x}}) = \overline{\underline{f}}^{*}(\overline{\boldsymbol{x}}) \tag{2.82}$$

holds, i.e., the range of its interval extended version in box $\overline{\boldsymbol{x}}$ is as tight as its image set $\overline{\underline{f}}^{*}(\overline{\boldsymbol{x}})$. This is also true for a particular variable x_i that occurs only once in a function $f(\boldsymbol{x})$, while all other variables have multiple variable instances. If all variables but x_i are replaced by their intervals one obtains the interval extended, univariate function $\overline{\underline{g}}_i(x_i)$

$$\overline{\underline{g}}_i(x_i) := \overline{\underline{f}}(\overline{x}_1, \ldots x_i, \ldots \overline{x}_n) \quad x_i \in \overline{x}_i \quad i = 1 \ldots n \quad , \tag{2.83}$$

for which Eq. 2.82 also applies

$$\overline{\underline{g}}_i(\overline{x_i}) = \overline{\underline{g}}_i^{*}(\overline{x_i}) \quad . \tag{2.84}$$

The image set of a function with multiple instances of a particular variable can be greatly overestimated by IA due to interval dependency as shown in Eq. 2.78

and Eq. 2.80. Hence, one can only conclude that $\underline{\overline{f}}(\underline{\overline{\boldsymbol{x}}})$ contains its image set

$$\underline{\overline{f}}(\underline{\overline{\boldsymbol{x}}}) \supseteq \underline{\overline{f}}^{*}(\underline{\overline{\boldsymbol{x}}}) \quad . \tag{2.85}$$

The described fundamental operations are sufficient enough to build interval extensions for the discussed NLEs in this thesis, as even long mathematical expressions can be decomposed into these unary and binary operations. Complex solutions are out of interest of this work's real-world problems. Therefore the complex space is neglected here, although interval extensions exist, e.g., $\log \underline{\overline{x}}, \quad \overline{x} < 0$, and can be found in Petković & Petković (1998). The intersection of two boxes $\underline{\overline{\boldsymbol{x}}}$ and $\underline{\overline{\boldsymbol{y}}}$ yields the subbox that is contained in both

$$\underline{\overline{\boldsymbol{x}}} \cap \underline{\overline{\boldsymbol{y}}} := (\underline{\overline{x}}_{i=1} \cap \underline{\overline{y}}_{i=1}, \ldots, \underline{\overline{x}}_{i=n} \cap \underline{\overline{y}}_{i=n}) \quad .$$

If the two intervals of any variable do not intersect the resulting box is empty. If box $\underline{\overline{\boldsymbol{x}}}$ is a subbox of $\underline{\overline{\boldsymbol{y}}}$ all of its variable intervals lie within the intervals of $\underline{\overline{\boldsymbol{y}}}$.

$$\underline{\overline{\boldsymbol{x}}} \subseteq \underline{\overline{\boldsymbol{y}}} \quad \text{if } \underline{\overline{x}}_{i=1} \subseteq \underline{\overline{y}}_{i=1}, \ldots, \underline{\overline{x}}_{i=n} \subseteq \underline{\overline{y}}_{i=n}$$

An interval extended function is said to be inclusion isotonic, if the following property holds

$$\underline{\overline{\boldsymbol{x}}} \subseteq \underline{\overline{\boldsymbol{y}}} \implies \underline{\overline{f}}(\underline{\overline{\boldsymbol{x}}}) \subseteq \underline{\overline{f}}(\underline{\overline{\boldsymbol{y}}}) \quad . \tag{2.86}$$

This is true for every interval extension of a real function applied on a box $\underline{\overline{\boldsymbol{x}}}$ that does not contain any division by zero. Hence, every subbox within $\underline{\overline{\boldsymbol{x}}}$ contains at most the minimum and maximum values of $\underline{\overline{f}}(\underline{\overline{\boldsymbol{x}}})$. As already mentioned $\underline{\overline{f}}(\underline{\overline{\boldsymbol{x}}})$ can overestimate the actual image set $\underline{\overline{f}}^{*}(\underline{\overline{\boldsymbol{x}}})$ of $f(\boldsymbol{x})$ in $\underline{\overline{\boldsymbol{x}}}$ due to interval dependency. To achieve a tighter estimation of $\underline{\overline{f}}^{*}(\underline{\overline{\boldsymbol{x}}})$, the box $\underline{\overline{\boldsymbol{x}}}$ can be divided into subboxes, for example by splitting up the interval of a variable x_i with multiple instances in

$f(x)$ into nl subintervals $\overline{\underline{x}}_{i,l}$ of equal width

$$\overline{\underline{x}}_{i,l} := [\overline{\underline{x}}_i + \frac{l-1}{nl} \cdot w(\overline{\underline{x}}_i), \quad \overline{\underline{x}}_i + \frac{l}{nl} \cdot w(\overline{\underline{x}}_i)] \qquad l = 1 \dots nl \tag{2.87}$$

$$\{\overline{\underline{\boldsymbol{x}}}\}_{i,nl} := \{ \begin{pmatrix} \overline{\underline{x}}_1 \\ \vdots \\ \overline{\underline{x}}_{i,1} \\ \vdots \\ \overline{\underline{x}}_n \end{pmatrix} ; \dots ; \begin{pmatrix} \overline{\underline{x}}_1 \\ \vdots \\ \overline{\underline{x}}_{i,nl} \\ \vdots \\ \overline{\underline{x}}_n \end{pmatrix} \} \quad . \tag{2.88}$$

In case the set $\{\overline{\underline{\boldsymbol{x}}}\}_{i,\infty}$ was theoretically constructed, one would completely remove the interval dependency for x_i as each subinterval then equals a real value in $\overline{\underline{x}}_i$. In consequence, the image set $\overline{\underline{g}}^*(x_i)$ would be obtained based on the definition of the interval extended, univariate function $\overline{\underline{g}}_i(x_i)$ from Eq. 2.84. Doing this for all variables in $\boldsymbol{x}$ with multiple instances in $f(\boldsymbol{x})$, the exact image set $\overline{\underline{f}}^*(\boldsymbol{x})$ could be theoretically achieved. To tackle interval dependency in a function $f(x)$ at least partly, one can divide the intervals of affected variables into subintervals and evaluate $f(x)$ by IA in each subbox separately. The union of the resulting function ranges $\cup\overline{\underline{f}}(\{\overline{\underline{\boldsymbol{x}}}\}_{i,nl})$ is in the worst case identical to $\overline{\underline{f}}(\overline{\underline{\boldsymbol{x}}})$ but will be much tighter in general. The following condition holds:

$$\overline{\underline{f}}(\overline{\underline{\boldsymbol{x}}}) \supseteq \cup\overline{\underline{f}}(\{\overline{\underline{\boldsymbol{x}}}\}_{i,nl}) \supseteq \overline{\underline{f}}^*(\overline{\underline{\boldsymbol{x}}}) \quad . \tag{2.89}$$

This is the so-called "refinement" of interval extended function evaluations. More on that topic can be found in Moore et al. (2009, pp. 53-55).

2.6.2 Interval Newton

IA is capable of extending the well-known, real-valued arithmetic Newton-Raphson method used for numerical iteration of nonlinear equation systems. The advantage over the latter is that its success no longer depends only on the quality of one chosen initial point. Instead, it requires initial ranges for all unknowns. They span the initial box that is iteratively reduced to the system's solution(s). Combinations

of the ***I**nterval **N**ewton with **G**eneralized **B**isection* (INGB) methods are able to find all solutions within such an initial box or they can proof that such a box does not contain any solution (Schnepper and Stadtherr, 1996). The information that an initial box is empty can be extremely useful for model debugging. If the expected range is not consistent to the used model, one has to look for structural errors in the model's formulation rather than for numerical issues during the iteration.

An interval extension of the real-valued arithmetic Newton-Raphson method was introduced by Moore in his dissertation and is extensively discussed in Moore et al. (2009, pp. 105-127). Similar to the real-valued method, a system referring to problem 2.74 is linearized by computing its first derivatives with respect to the iteration variables. The derivatives are evaluated in the current box $\overline{\underline{\boldsymbol{x}}}^{(k)}$ of the k-th iteration step and stored in an interval extended Jacobian matrix $\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}}^{(k)})$. In order to determine the Newton step the vector of the system's functions $\overline{\underline{f}}(\boldsymbol{x}_c)$ is evaluated at any point $\boldsymbol{x}_c$ within $\overline{\underline{\boldsymbol{x}}}^{(k)}$. A common choice is the midpoint $m(\overline{\underline{\boldsymbol{x}}}^{(k)})$. Based on these quantities the interval extended Newton step is shown in Eq. 2.91.

$$\boldsymbol{x}_c := m(\overline{\underline{\boldsymbol{x}}}^{(k)}) \tag{2.90}$$

$$\overline{\underline{\boldsymbol{x}}}^{(k+1)} := \overline{\underline{\boldsymbol{x}}}^{(k)} \cap \underbrace{\left(\boldsymbol{x}_c - \overline{\underline{\boldsymbol{f}}}(\boldsymbol{x}_c) \cdot \overline{\underline{\boldsymbol{J}}}^{\,-1}(\overline{\underline{\boldsymbol{x}}}^{(k)})\right)}_{\Gamma_{nwt}\left(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{x}_c,\overline{\underline{\boldsymbol{x}}}^{(k)}\right)} \quad . \tag{2.91}$$

The interval extended version of the function vector is used as this method also works if some of the system's parameters are uncertain and can only be represented by intervals. In this case the method still converges and terminates with (an) exact solution range(s) within the chosen bounds. Figure 2.5 shows the graphical interpretation of an univariate function in $\overline{\underline{x}}^{(0)} = [0, 4]$. The Jacobian matrix equals the gradient with respect to x, which is an interval here. The roots of the linearized system for the minimum and maximum gradient value determine lower and upper bounds of the next iteration step's x-interval that is $\overline{\underline{x}}^{(k+1)}$. A gradient value of 0 results in infinite roots for $0 \in \overline{\underline{f}}(x_c)$ or no root for $0 \notin \overline{\underline{f}}(x_c)$. Both make the intersection with an initially provided interval referring to Eq. 2.91 inevitable as this is the only way to bound an iteration variable range in this case (see determination of $\underline{x}^{(1)}$ in figure 2.5). Nevertheless, the procedure still converges even in such a singular case.

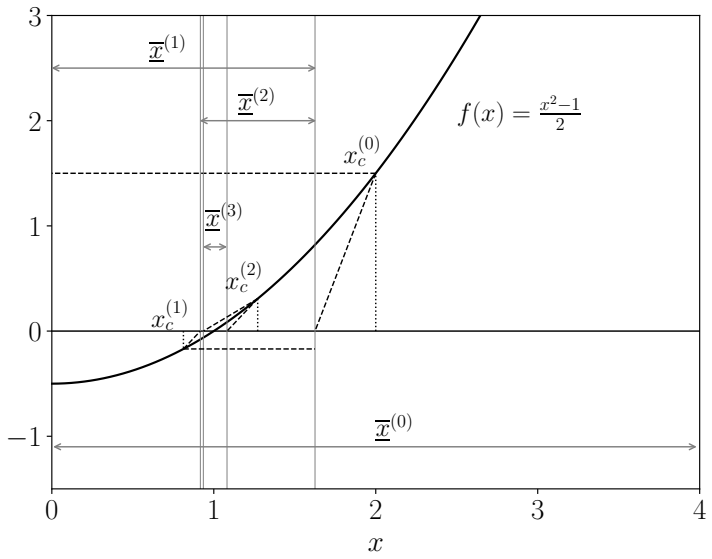


Fig. 2.5: First two iteration steps of the interval Newton method to determine the root of $f(x)$.

The interval Newton method is able to identify empty boxes and boxes with a unique solution. These properties are for example proven in Neumaier (1990, pp. 131-152). Following conclusions can be made regarding box $\overline{\underline{\boldsymbol{x}}}^{(k)}$:

1) If $\Gamma_{nwt}\left(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_c, \overline{\underline{\boldsymbol{x}}}^{(k)}\right) \subset \overline{\underline{\boldsymbol{x}}}^{(k)} \implies$ There is a unique solution $\boldsymbol{x}^* \in \overline{\underline{\boldsymbol{x}}}^{(k)}$ with $\boldsymbol{f}(\boldsymbol{x}^*) = 0$, the interval Newton method converges to it. All variable bounds of the reduced box must lie in the interior of $\overline{\underline{\boldsymbol{x}}}^{(k)}$.

2) Else if $\Gamma_{nwt}\left(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{x}_c, \overline{\underline{\boldsymbol{x}}}^{(k)}\right) \cap \overline{\underline{\boldsymbol{x}}}^{(k)} = \emptyset \implies$ There is no solution in $\overline{\underline{\boldsymbol{x}}}^{(k)}$ that satisfies $\boldsymbol{f}(\boldsymbol{x}) = 0$.

3) Else if $\Gamma_{nwt}\left(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x}_c, \overline{\underline{\boldsymbol{x}}}^{(k)}\right) \cap \overline{\underline{\boldsymbol{x}}}^{(k)} = \overline{\underline{\boldsymbol{x}}}^{(k)} \implies$ The box is consistent and $\overline{\underline{\boldsymbol{x}}}^{(k)}$ needs to be bisected into subboxes that are processed independently.

4) Else: $\overline{\underline{\boldsymbol{x}}}^{(k+1)} := \Gamma_{nwt}\left(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x}_c, \overline{\underline{\boldsymbol{x}}}^{(k)}\right) \cap \overline{\underline{\boldsymbol{x}}}^{(k)}$, no conclusion can be made at this point and the test is repeated in the next iteration step.

This so-called "root inclusion test", should be applied on a box after its Newton step reduction as long as it has not been discarded by case 2) or been proven to

contain a unique solution by case 1). In this way, all solutions to an NLE can be found, e.g., with the method suggest by Schnepper & Stadtherr (1996).

The inverse of the Jacobian matrix in Eq. 2.91 is not computed directly, as this generally leads to an overestimated box $\overline{\underline{x}}^{(k+1)}$ due to interval dependency (Hansen and Smith, 1967). Instead, other methods exist to solve the underlying linear system. A detailed overview is given by Hansen & Walster (2003, pp. 83-106). The interval extended Gauss elimination procedure as introduced by Wilkinson (1961) works well on nearly degenerate entries in $\overline{\underline{\boldsymbol{J}}}$ and $\overline{\underline{\boldsymbol{f}}}(\boldsymbol{x}_c)$ and can be used to bound errors or proof regularity of linear systems, i.e., the interval matrix contains no real-valued matrices that are singular. It is not applied in this work as the initial variable intervals are generally wide and dependency issues during the procedure coupled with propagation of round-off errors through the sequential solving procedure would greatly increase the chance of the method to fail. Another option is the hull method that traces back to Bliek (1992) and Hansen (1992) and was further developed by Rohn (1993), Ning & Kearfott (1997), Neumaier (1999), Neumaier (2000) and Hansen (2000). This method requires a regular Jacobian matrix, which cannot be guaranteed for, especially within a wide initial box. The third method is an interval extended way of the Gauss-Seidel procedure that was proposed by Alefeld & Herzberger (1970) and further extended by Hansen & Sengupta (1981), which also works on irregular matrices. Hansen's and Sengupta's version involves preconditioning of $\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}})$ and $\overline{\underline{\boldsymbol{f}}}(\boldsymbol{x}_c)$ that will be referred to at the end of this section. Without preconditioning the procedure is formulated as follows

$$\overline{\underline{x}}_i^{(k+1)} := \overline{\underline{x}}_i^{(k)} \cap x_{c,i}^{(k)} - \frac{\overline{\underline{f}}_i(\boldsymbol{x}_c) + \sum_{j=1}^{i-1} \overline{\underline{j}}_{ij} \cdot (\overline{\underline{x}}_j^{(k+1)} - x_{c,j}^{(k+1)}) + \sum_{j=i+1}^{n} \overline{\underline{j}}_{ij} \cdot (\overline{\underline{x}}_j^{(k)} - x_{c,j}^{(k)})}{\overline{\underline{j}}_{ii}} \tag{2.92}$$

for the i-th equation and variable of the equation system. This is performed successively for all n variables and equations of the square problem, and the variable intervals are updated along the way. If the Jacobian interval $\overline{\underline{j}}_{ii}$ contains zero, the Gauss-Seidel step from Eq. 2.92 would not reduce $\overline{\underline{x}}_i^{(k)}$ by applying the division rule from table 2.2. Therefore, Hansen and Sengupta defined this

operation in a new way.

$$\overline{\underline{x}}/\overline{\underline{y}} := \begin{cases} \overline{\underline{x}} \cdot [1/\overline{y}, 1/\underline{y}] & \text{if} \quad 0 \notin \overline{\underline{y}}, \\ [-\infty, +\infty] & \text{if} \quad 0 \in \overline{\underline{x}} \quad \text{and} \quad 0 \in \overline{\underline{y}} \\ [\overline{x}/\underline{y}, +\infty] & \text{if} \quad \overline{x} < 0 \quad \text{and} \quad \underline{y} < \overline{y} = 0 \\ [-\infty, \overline{x}/\overline{y}] \cup [\overline{x}/\underline{y}, +\infty] & \text{if} \quad \overline{x} < 0 \quad \text{and} \quad \underline{y} < 0 < \overline{y} \\ [-\infty, \overline{x}/\overline{y}] & \text{if} \quad \overline{x} < 0 \quad \text{and} \quad 0 = \underline{y} < \overline{y} \\ [-\infty, \underline{x}/\underline{y}] & \text{if} \quad 0 < \underline{x} \quad \text{and} \quad \underline{y} < \overline{y} = 0 \\ [-\infty, \underline{x}/\underline{y}] \cup [\underline{x}/\overline{y}, +\infty] & \text{if} \quad 0 < \underline{x} \quad \text{and} \quad \underline{y} < 0 < \overline{y} \\ [\underline{x}/\overline{y}, +\infty] & \text{if} \quad 0 < \underline{x} \quad \text{and} \quad 0 = \underline{y} < \overline{y} \\ \emptyset & \text{if} \quad 0 \notin \overline{\underline{x}} \quad \text{and} \quad 0 = \overline{\underline{y}} \end{cases} \tag{2.93}$$

Doing so, one gains the useful information about the infeasible region around zero, which can never be part of the solution set produced by applying real-valued arithmetic division on all real elements in $\overline{\underline{x}}$ and $\overline{\underline{y}}$. Consequently one excludes the singular case and the former interval splits up into two disjoint subintervals related to two subboxes. In order to use this method efficiently, there should be no structural zeros on the diagonal of the Jacobian matrix, i.e., that the i-th variable is not part of the i-th equation. This case can be avoided by applying the DM decomposition before (see section 2.4.2), given that the system of interest is well-determined.

As an alternative interval extended Newton step, Krawczyk's method should be mentioned, which was originally introduced to avoid divisions by zero containing intervals in Eq. 2.91 (Krawczyk, 1969). Neumaier showed that the preconditioned interval Newton method combined with the Gauss-Seidel procedure produces an enclosure, which is at least as tight as the one resulting from Krawczyk's method (Neumaier, 1990, p. 138). In consequence, the latter will not be further considered in this thesis.

Preconditioning is applied to achieve sharper bounds for the intervals resulting from Eq. 2.92 and to identify boxes with unique or no solution by the root inclu-

sion test as soon as possible. In fact, if the Jacobian matrix fulfills the condition

$$\min_{x \in \bar{\underline{J}}_{i,i}} |x| > \sum_{j=1, j \neq i}^{n} |\bar{\underline{j}}_{i,j}| \quad , \tag{2.94}$$

it is said to be diagonally dominant, in which case the interval Newton method converges without any bisection required (Kearfott, 1990). A common approach is to scale Eq. 2.91 by the real-valued matrix, that contains all midpoints of the interval Jacobian matrix $m\left(\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}}^{(k)})\right)^{-1}$ so that the general Newton step becomes

$$\overline{\underline{\boldsymbol{x}}}^{(k+1)} := \overline{\underline{\boldsymbol{x}}}^{(k)} \cap \left(\boldsymbol{x}_c - \underbrace{\overline{\underline{\boldsymbol{f}}}(\boldsymbol{x}_c) \cdot m\left(\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}}^{(k)})\right)^{-1}}_{\overline{\underline{\boldsymbol{k}}}} \cdot \underbrace{\overline{\underline{\boldsymbol{J}}}^{-1}(\overline{\underline{\boldsymbol{x}}}^{(k)}) \cdot m\left(\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}})^{(k)}\right)}_{\overline{\underline{\boldsymbol{G}}}^{-1}} \right) \quad , \tag{2.95}$$

with the intention that $\overline{\underline{\boldsymbol{G}}}$ is more diagonally dominant than $\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}}^{(k)})$. This is true if $\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}}^{(k)})$ is a regular matrix with rather small interval entries so that it is almost degenerate. For a degenerate Jacobian matrix, $\overline{\underline{\boldsymbol{G}}}$ equals the identity matrix, which matches condition 2.94. The Gauss-Seidel procedure becomes

$$\overline{\underline{x}}_i^{(k+1)} := \overline{\underline{x}}_i^{(k)} \cap x_{c,i}^{(k)} - \frac{\overline{\underline{k}}_i + \sum_{j=1}^{i-1} \overline{\underline{g}}_{ij} \cdot (\overline{\underline{x}}_j^{(k+1)} - x_c^{(k+1)}) + \sum_{j=i+1}^{n} \overline{\underline{g}}_{ij} \cdot (\overline{\underline{x}}_j^{(k)} - x_c^{(k)})}{\overline{\underline{g}}_{ii}} \quad . \tag{2.96}$$

Unfortunately, this preconditioning might worsen the performance for irregular Jacobian matrices. Kearfott introduced an alternative preconditioning strategy for the Gauss-Seidel procedure, which „involves optimality conditions expressible as linear programming problems“(Kearfott, 1990, p. 1) to find the minimal widths of the variable intervals. The interested reader is referred to Kearfott (1990), where the performance of multiple preconditioners for the Gauss-Seidel procedure are compared. Gau and Schnepper have suggested two alternative preconditioners. The first one is the usage of the real-valued inverse of the current box's mid points $\boldsymbol{J}(m(\overline{\underline{\boldsymbol{x}}}^{(k)}))^{-1}$ instead of $m\left(\overline{\underline{\boldsymbol{J}}}(\overline{\underline{\boldsymbol{x}}}^{(k)})\right)^{-1}$ in Eq. 2.95, so that the computation of $\overline{\underline{\boldsymbol{G}}}^{-1}$ becomes less expensive. The second option is a pivoting strategy to identify

the optimal row / column combination, i.e., in which equation a variable interval can be reduced the most or the box can be even discarded if the interval does not overlap with the original one (Schnepper and Stadtherr, 1996).

2.6.3 Consistency Techniques

In contrast to the interval Newton method, in which the initial variable space is reduced via a sequence of linear approximations of an NLE at the current iterate, consistency methods use the initial variable intervals directly and check to which extent at all, they can contain solutions in the individual equations of Eq. 2.74. For example, in a scalar function $f(x, y)$ that depends on two variables x, y with their initial intervals $\underline{\overline{x}}$, $\underline{\overline{y}}$, a subinterval $\underline{\overline{y}}_s \subset \underline{\overline{y}}$ can be removed if

$$0 \notin \underline{\overline{f}}(\underline{\overline{x}}, \underline{\overline{y}}_s) \quad .$$

In this way, the initial variable space can be successively reduced by an alternation through all equations and variables. Consistency methods mostly outperform the interval Newton method during the reduction of large boxes (Hansen and Walster, 2003, p. 205) and can, to some extent, tackle the interval dependency problem. However, there are cases in which the variable intervals of a box become mutually consistent, i.e., no further subbox can be removed, although the box is not yet degenerate according to the desired tolerances. The size of a consistent box depends on the applied method and the formulation of the equations. In this work, two consistency methods are investigated, namely: hull and box consistency. Before both methods are introduced, the differences in their achievable consistencies shall be discussed.

Lhomme (1993) defined three degrees of consistency in IA from which arc consistency is the highest one measured by the tightness of the variable intervals followed by hull and box consistency. Table 2.3 shows some 2D/3D examples with consistent variable intervals respectively. Equations with arc consistent intervals are also hull consistent and hull consistent intervals are box consistent as well. Arc consistency means no infeasible regions exist within the consistent intervals as is the case for examples (a), (c) and (e) in table 2.3. Hull consistency allows these infeasible regions, which can be seen in example (b) as there is no solution

for $\{x_1 \in \underline{\overline{x}} \mid -2 < x_1 < 2\}$. Hull consistency algorithms decompose an equation into so-called primitives, which could be for example (b) $p_1 = x_1^2$, $p_2 = 2 \cdot x_2$ and $p_1 = p_2$. The equation is said to be hull consistent, if all primitives are fulfilled using the IA operations defined in section 2.6.1. If no infeasible regions occur in the equation's related intervals, it is hull and arc consistent, which holds for example (c): One can easily see that the underlying constraint $x_1 = x_2$ needs to be met so that consistency is reached as soon as $\underline{\overline{x}}_1 = \underline{\overline{x}}_2$. Box consistency algorithms would stop with already looser bounds for x_2 as example (d) shows. They use the IA operations for all variables but one, which is evaluated by real arithmetic at its bounds instead. Example (d) would therefore be evaluated at its distinct bounds of $\underline{\overline{x}}_1$ and $\underline{\overline{x}}_2$. As both function ranges contain zero

$$0 \in [\underline{x}_1 - 2 \cdot \underline{x}_1, \overline{x}_1 - 2 \cdot \overline{x}_1] + \underline{\overline{x}}_2 = [-1, 2] \tag{2.97}$$

$$0 \in \underline{\overline{x}}_1 - 2 \cdot \underline{\overline{x}}_1 + [\underline{x}_2, \overline{x}_2] = [-2, 3] \quad , \tag{2.98}$$

the box is consistent. Due to interval dependency in Eq. 2.98, when $\underline{\overline{x}}_1$ is evaluated by IA, the bounds of $\underline{\overline{x}}_2$ can not be adequately tightened. However, the underlying constraint $x_1 = x_2$ is normally not this easily found and multiple occurrences of the same variable in single equations are generally processed as independent variables in hull consistency methods with the same initial interval. This is exemplarily shown in equation (f), which is the decomposed version of (e) that a typical hull consistency method works with. Example (f) is already hull consistent when $\underline{\overline{x}}_1 = \underline{\overline{x}}_3 = \underline{\overline{x}}_4 = [-1, 1]$, i.e., for the decomposition

$$\underline{\overline{x}}_1^3 = [-1, 1] \subseteq \underline{\overline{x}}_3^2 - \underline{\overline{x}}_4 - \underline{\overline{x}}_2 = [-4, 2] \tag{2.99}$$

$$\underline{\overline{x}}_2 = [0, 3] \subseteq \underline{\overline{x}}_3^2 - \underline{\overline{x}}_4 - \underline{\overline{x}}_1^3 = [-2, 3] \tag{2.100}$$

$$\underline{\overline{x}}_3^2 = [0, 1] \subseteq \underline{\overline{x}}_1^3 + \underline{\overline{x}}_4 + \underline{\overline{x}}_2 = [-2, 5] \tag{2.101}$$

$$\underline{\overline{x}}_4 = [-1, 1] \subseteq \underline{\overline{x}}_3^2 - \underline{\overline{x}}_1^3 - \underline{\overline{x}}_2 = [-4, 2] \quad . \tag{2.102}$$

In contrast to the latter, a box consistency method keeps the linkage between the variable instances and can further reduce $\underline{\overline{x}}_1$ to $[-1, 0]$ by equation (e), because there is no solution for $x_1 = 1$

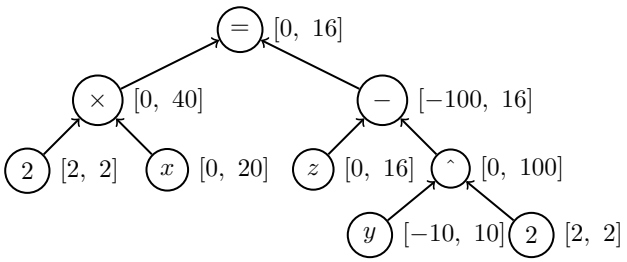
$$0 \notin (1)^3 - (1)^2 + 1 + [0, 3] = [1, 4] \quad , \tag{2.103}$$

Tab. 2.3: Partial consistencies for 2D and 3D examples.

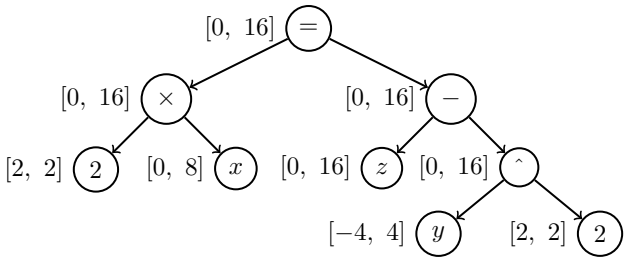
Examples	$\overline{x}_1/\underline{x}_3/\overline{x}_4\,\overline{x}_2$		**Highest consistency**	**ID**
$x_1^2 - 2 \cdot x_2 = 0$	$[2, 3]$	$[2, 4.5]$	Arc consistent	(a)
	$[-3, 3]$	$[2, 4.5]$	Hull consistent	(b)
$x_1 - 2 \cdot x_1 + x_2 = 0$	$[0, 1]$	$[0, 1]$	Arc consistent	(c)
	$[0, 1]$	$[0, 2]$	Box consistent	(d)
$x_1^3 - x_1^2 + x_1 + x_2 = 0$	$[-1, 0]$	$[0,\ 3]$	Arc consistent	(e)
			Box consistent	
$x_1^3 - x_3^2 + x_4 + x_2 = 0$	$[-1, 1]$	$[0,\ 3]$	Arc consistent	(f)

and any other distinct value $x_1 > 0$. In this case, box consistency is equivalent to the true arc consistent bounds. In the work of Collavizza et al. (1998) this rank order of consistencies is proved mathematically. Arc consistent methods do not exist, as far as we know, because it does not seem very efficient to filter out all infeasible regions within intervals at once. After all, 2^k subboxes would arise from a box if each of k of its variable intervals previously contained one infeasible region. Hull and box consistency methods, by definition, allow such infeasibile regions in intervals and remove one at a time as a box becomes consistent. Given that, as shown in the previous examples, for some equation formulations hull consistency yields sharper variable bounds and for others box consistency, both approaches are used for box reduction.

The hull consistency method applied in this thesis is the HC4revise algorithm that has been introduced by Benhamou & Puget (1999). Figure 2.6 presents the decomposed version of an equation by this algorithm where nodes equal either mathematical operations, variables or constants. Starting from the “leaves", variable intervals are propagated using the IA-based mathematical operations and both equation sides are intersected in the so-called forward evaluation step. The resulting interval at the node of the root is $[0, 16]$. Starting from this root node, the equation is now evaluated in reverse order to end at the leaves and reduce the variable intervals on the way. This is called backward evaluation. Here, the



(a) Forward evaluation.



(b) Backward evaluation.

Fig. 2.6: Working principle of HC4revise applied on equation: $2 \cdot x = z - y^2$ within the bounds: $\overline{x} = [0, 20]$, $\overline{y} = [-10, 10]$ and $\overline{z} = [0, 16]$. Example is taken from Benhamou & Puget (1999).

intervals $\overline{x} = [0, 20]$ and $\overline{y} = [-10, 10]$ can be reduced to $[0, 8]$ and $[-4, 4]$, while $\overline{z}$ remains unchanged. The equation is hull consistent in the resulting box. In order to solve an NLE of more than one equation, it is cycled through all equations. All reduced intervals of one variable are intersected along the way, because the resulting domain needs to be feasible within every equation. One pass through all equations is termed one reduction step and denoted for the hull consistency method by the use of the contractor Γ_{hc} as

$$\overline{\boldsymbol{x}}^{(k+1)} := \overline{\boldsymbol{x}}^{(k)} \cap \Gamma_{hc} \left(\boldsymbol{f}(\boldsymbol{x}), \overline{\boldsymbol{x}}^{(k)} \right) \quad . \tag{2.104}$$

The algorithm stops when all equations are hull consistent in the remaining box. To sum up, hull consistency methods have the advantage of not requiring any derivatives. However, a disadvantage is that especially in equations with several instances of a variable, intervals cannot be reduced well as the examples from table 2.3 have shown. Box-consistency methods seem to be better suited for this purpose. Box consistency methods extend a real-valued function of the type

$$f(\boldsymbol{x}) = 0, \qquad \boldsymbol{x} \in \mathbb{R}^n \tag{2.105}$$

to its IA representation by replacing all real-valued variables by their intervals except one

$$\overline{\underline{g}}(x_i) = \overline{\underline{f}}(\overline{\underline{x}}_1, \ldots, x_i, \ldots, \overline{\underline{x}}_n) = 0 \quad . \tag{2.106}$$

The interval extended, univariate function $\overline{\underline{g}}(x_i)$ is then iterated within the current interval $\overline{\underline{x}}_i$ to remove parts in $\overline{\underline{g}}(x_i)$ that do not fulfill Eq. 2.106. Commonly, the interval Newton operator from section 2.6.2 is applied on the edge intervals $\overline{\underline{y}} = [\underline{x}_i, c]$ and $\overline{\underline{z}} = [d, \overline{x}_i]$ with c,$d \in \overline{\underline{x}}_i$ to linearly approximate the lowest and highest root in $\overline{\underline{x}}_i$. An implementation can be found in Mc-Allester et al. (1995) where the mid points of $\overline{\underline{y}}$ and $\overline{\underline{z}}$ are used as point of expansion for the Newton step

$$\overline{\underline{x}}_i^{(k+1)} := \overline{\underline{x}}_i^{(k)} \cap \left[\underline{\Gamma_{nwt}\left(\overline{\underline{g}}(x_i), m(\overline{\underline{y}}), \overline{\underline{y}}\right)}, \ \overline{\Gamma_{nwt}\left(\overline{\underline{g}}(x_i), m(\overline{\underline{z}}), \overline{\underline{z}}\right)} \right] \quad . \tag{2.107}$$

The choice of c and d directly influences the widths of $\overline{\underline{y}}$ and $\overline{\underline{z}}$ and therefore the widths of the gradient intervals $\overline{\frac{\partial g}{\partial x_i}(\overline{\underline{y}})}$ and $\overline{\frac{\partial g}{\partial x_i}(\overline{\underline{z}})}$. A heuristic is given in Hansen & Walster (2003, pp. 201-203) on how to choose c and d to avoid slow progress for too small gradient intervals or no progress for too large ones with approximated roots outside of $\overline{\underline{x}}_i$. Similar to hull consistency methods, box consistency methods cycle through all equations and variable domains until box consistency is achieved. Analogously, one pass through all equations is expressed by the box consistency operator Γ_{bc}

$$\overline{\underline{\boldsymbol{x}}}^{(k+1)} := \overline{\underline{\boldsymbol{x}}}^{(k)} \cap \Gamma_{bc}\left(\boldsymbol{f}(\boldsymbol{x}), \overline{\underline{\boldsymbol{x}}}^{(k)}\right) \quad . \tag{2.108}$$

As already mentioned box consistency methods often find tighter bounds than hull consistency and interval Newton operators for a variable that occurs multiple times in a function due to the avoidance of interval dependency through its real-value evaluation. Nevertheless, it is also said to be the computationally most expensive method out of the three (Hansen and Walster, 2003, p. 227). The algorithm HC4 suggested by Benhamou & Puget (1999) combines hull and box consistency methods. It uses the aforementioned HC4revise algorithm for single and a box consistency method for multiple variable occurrences in a currently processed equation. Like the Interval Newton method, a “root inclusion test" can

also be performed for consistency methods (Hansen and Walster, 2003, p. 226), i.e., they can proof uniqueness or nonexistence of solution(s) in boxes.

2.6.4 Splitting Techniques

Whenever a problem becomes consistent without all variable intervals being degenerate the current box needs to be split into subboxes that are then processed independently. Consistency methods will reduce any initial box for the following 2D-example

$$x_1^2 + x_2^2 = 1 \tag{2.109}$$

$$x_1 - x_2 = 0 \tag{2.110}$$

to the consistent intervals $\overline{x}_1 = \overline{x}_2 = [-1, 1]$. Only by splitting one interval, the algorithm proceeds to find the two solutions

$$\{\boldsymbol{x}^*\} = \{\left(-\sqrt{0.5},\ -\sqrt{0.5}\right); \left(\sqrt{0.5},\ \sqrt{0.5}\right)\} \quad .$$

An interval may be bisected, but theoretically an interval can also be split at any other interior point. The general bisection step of a box is described by

$$\{\overline{\underline{\boldsymbol{x}}}^{(k+1)}\} := \underbrace{\{ \begin{pmatrix} \overline{\underline{x}}_{i=1}^{(k)} \\ \vdots \\ [\underline{x}_{i=s}^{(k)},\ m(\overline{\underline{x}}_{i=s}^{(k)})] \\ \vdots \\ \overline{\underline{x}}_{i=n}^{(k)} \end{pmatrix} ; \begin{pmatrix} \overline{\underline{x}}_{i=1}^{(k)} \\ \vdots \\ [m(\overline{\underline{x}}_{i=s}^{(k)}),\ \overline{x}_{i=s}^{(k)}] \\ \vdots \\ \overline{\underline{x}}_{i=n}^{(k)} \end{pmatrix} \}}_{\Gamma_S\left(\boldsymbol{f}(\boldsymbol{x}), \overline{\underline{\boldsymbol{x}}}^{(k)}\right)} \quad , \tag{2.111}$$

where $\overline{\underline{x}}_{i=s}$ denotes the variable interval that is chosen to be split. Several heuristics exist. Table 2.4 presents the most common ones among them. To determine the largest derivative one requires a merit function *obj* that is minimized. A valid

choice could be the sum of the squared function residuals

$$obj = \sum_{j=1}^{n} f_j(\boldsymbol{x})^2 \quad . \tag{2.112}$$

Uniform subdivision is not practical as it produces too many boxes, especially in large NLEs. Cyclic bisection may suffer of unsatisfying progress in the box reduction according to Asaithambi et al. (1982) and Ratschek & Rokne (1988). A good summary on this topic is given in Ratschek & Rokne (2009). Whenever a variable interval contains an infeasible region, a so-called gap, the Hansen-Sengupta operator from Eq. 2.93 could theoretically split it into two feasible subintervals. If the initial box contains many of these gaps such a reduction would cause a combinatorial explosion as k gaps produce 2^k subboxes that all need to be processed (Lhomme, 1993). In consequence, contractors usually ignore the gap and only mark intervals with a gap until the overall problem becomes consistent. Subsequently, only one of the intervals with a gap is split to produce two subboxes, a common choice is to select the largest gap (Hansen and Greenberg, 1983). Other contracting methods are then used on both subboxes until consistency is reached once again. Usually, some subboxes are proven to be empty during the procedure so that the number of them remains reasonable.

Tab. 2.4: Methods for choosing which variable's interval to split of an n-dimensional box.

Method	Selected variable interval(s)	Number of boxes	Source
Uniform subdivision	all n intervals	2^n	(Moore et al., 2009, pp. 55-57)
Largest width	$\max_{i=1\ldots n} w(\overline{x}_i)$	2	(Hansen and Greenberg, 1983)
Cyclic bisection	cycling all n intervals	2	(Moore, 1979, pp. 49-50)
Largest derivative	$\max_{i=1\ldots n} w\left(\overline{\frac{\partial\ obj}{\partial x_i}(\overline{\boldsymbol{x}})} \cdot (\overline{x}_i - m(\overline{x}_i))\right)$	2	(Ratz and Csendes, 1995)

2.6.5 Cutting

An alternative way to splitting is the usage of cutting that can be seen as a stronger consistency than the aforementioned hull consistency (Lhomme, 1993). Instead of removing infeasible gaps in the interior of a variable interval, one tries to cut off infeasible subintervals at its lower and/or upper bound by proving the related subbox to be empty for the given problem. Figure 2.7 shows this principle. Through cutting one can for example further reduce the hull consistent bounds $\overline{x}_1 = \overline{x}_2 = [0, 1]$ of the equation system given by Eq. 2.109 and Eq. 2.110 without generating new subboxes by splitting. Applying a hull consistency method on Eq. 2.109 in the subinterval $\overline{x}_1^u = [0.8, 1]$ of $\overline{x}_1$ results in $\overline{x}_2 = [0, 0.6]$. However, this contradicts Eq. 2.110, so that the subbox is empty. Accordingly, this can be proven for the subinterval $\overline{x}_1^l = [0, 0.6]$ and as the problem is symmetrical, both intervals $\overline{x}_1$ and $\overline{x}_2$ can be reduced to $[0.6, 0.8]$. In this way, the risk of exponential growth of the computational effort caused by too many subboxes can be partially avoided. Nevertheless, cutting cannot completely replace splitting as the intervals can also become consistent without being degenerate. The reason is again interval dependency, which also occurs in the considered example, i.e., for

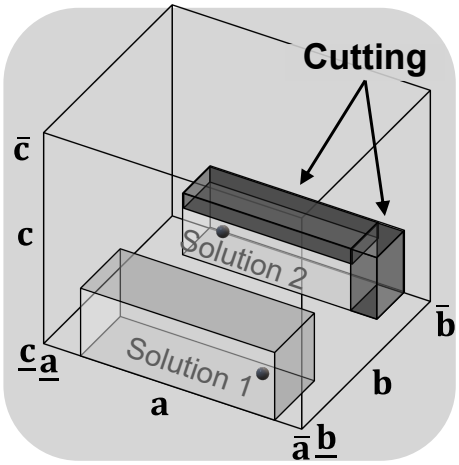


Fig. 2.7: Sketch of box cutting. The dark subboxes at the edges can be removed if they are proven to be empty.

a certain $x_1 \in [0.6, 0.8]$ x_2 has different values to solve Eq. 2.109 and Eq. 2.110 in real-valued arithmetic.

3 Novel Hybrid Approach

The proposed hybrid approach is an automatic initialization and solution strategy to efficiently solve NLEs of the form $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ in case no converging initial point or tight bounds are at hand. The user must provide an initial box $\overline{\underline{\boldsymbol{x}}}^{(0)}$. The hybrid approach then follows the interval and real-valued arithmetic containing steps shown in figure 3.1 from the perspective of one box.

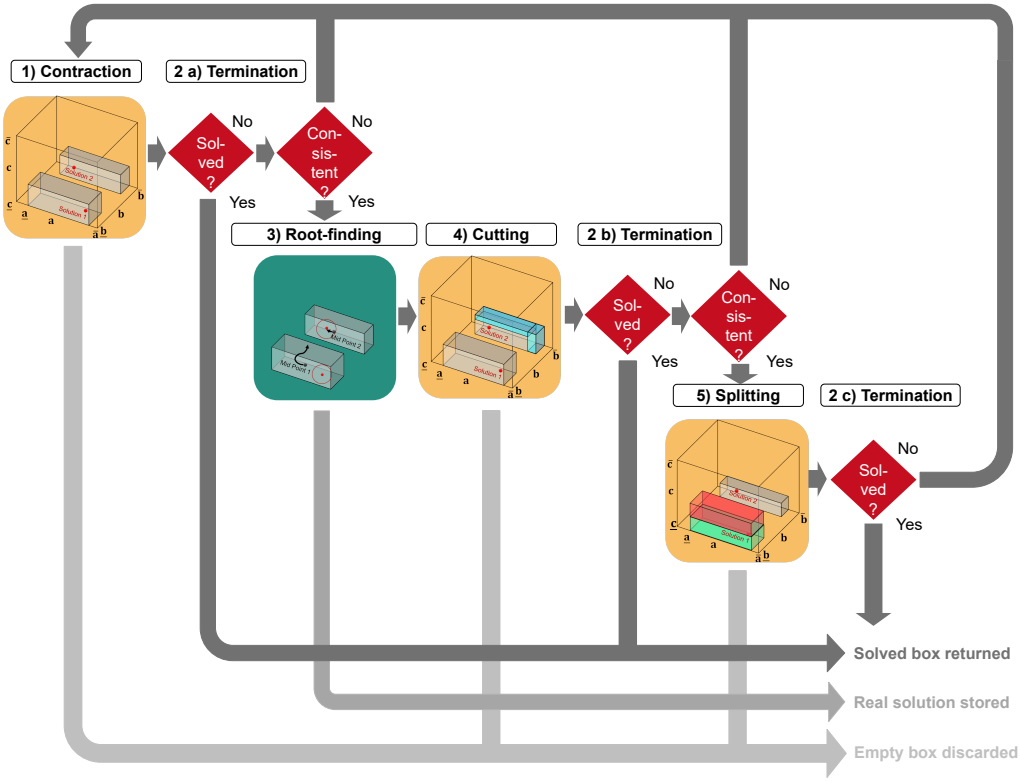


Fig. 3.1: Five steps of the hybrid approach: IA steps (yellow shaded), termination steps (red shaded), real arithmetic root-finding step (green shaded).

Sections 3.1 to 3.5 explain these steps in detail. It is essential that as soon as a box is solved or identified as being empty, it is marked accordingly and not reduced any further. A box is labeled as "solved" when it is degenerate, i.e., the width of each interval falls below a minimum value determined by a relative and absolute tolerance (explained in section 3.2). If a solution is found in the root-finding step, it is immediately reported to the user in a text file. If the key "termination" is set to `one_solution` in the solver's dictionary, the algorithm acts as a local solver and terminates the hybrid approach when a real-valued solution, i.e., a single root, is found. Otherwise, the solution is stored and used in the unique solution test, i.e., if a box satisfies the condition and the solution lies inside the box, it does not need to be reduced further and is also considered solved. A reduction step involves that all currently unsolved boxes are processed once by the hybrid approach. How such a step looks like for an individual box is explained in section 3.6 and how it does for multiple boxes is topic of section 3.7. Subsequently, section 3.8 discusses parallelized processes in the algorithm, and section 3.9 presents briefly the implementation of the hybrid approach in Python. Section 3.10 closes this chapter with the definition of some analysis parameters to evaluate the performance of the hybrid approach in the upcoming chapters.

3.1 Contraction

Three different methods can be used for the contraction step in the hybrid approach, namely

- The Interval Newton (see section 2.6.2)
- The hull consistency method HC4revise from Benhamou & Puget (1999) (see section 2.6.3)
- A new box consistency method, called Bnormal (own development, explained in this section)

The Interval Newton has been implemented in form of the Gauss-Seidel step referring to Eq. 2.92. It is iterated variable-wise, meaning that the variables are successively reduced and each newly reduced variable interval is subsequently

used in the next variable's reduction step. Gaps during the reduction are ignored and only the outer bounds of the interval are used. Nevertheless, such an interval is marked as discontinuous and can be used for splitting later. Except for the preconditioner `pivotAll`, the i-th variable is only reduced in the i-th equation. The DM decomposition (see section 2.4.2) ensures that there is no structural zero on the diagonal of the Jacobian matrix, i.e., the i-th equation depends always on the i-th variable for a well-posed equation system. The general form of the Newton contractor applied on the box $\overline{\underline{\boldsymbol{x}}}^{(k)}$ in the k-th reduction step is

$$\overline{\underline{\boldsymbol{x}}}^{(k+1)} := \Gamma_{nwt}\left(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x}_c, \overline{\underline{\boldsymbol{x}}}^{(k)}\right) \quad , \tag{3.1}$$

with the point of expansion chosen to be the intervals mid point

$$\boldsymbol{x}_c := m(\overline{\underline{\boldsymbol{x}}}) \quad . \tag{3.2}$$

The Jacobian matrix, or to be more precise, the Gauss-Seidel step can be preconditioned in a variety of ways. The Jacobian matrix and the vector of the functions' residuals are multiplied by a real-valued, preconditioning matrix $\boldsymbol{P}$ to improve the condition of the resulting interval matrix $\overline{\underline{\boldsymbol{G}}}$, i.e., having interval entries of similar orders of magnitude.

$$\overline{\underline{\boldsymbol{G}}} := \overline{\underline{\boldsymbol{J}}} \cdot \boldsymbol{P} \qquad \boldsymbol{P} \in \mathbb{R}^n \times \mathbb{R}^n \tag{3.3}$$

$$\overline{\underline{\boldsymbol{k}}}(\boldsymbol{x}_c) := \boldsymbol{P} \cdot \overline{\underline{\boldsymbol{f}}}(\boldsymbol{x}_c) \tag{3.4}$$

$$\boldsymbol{P} := \begin{bmatrix} p_{1,1} & \cdots & p_{1,n} \\ \vdots & \ddots & \vdots \\ p_{m,1} & \cdots & p_{m,n} \end{bmatrix} \tag{3.5}$$

Entries of $\overline{\underline{\boldsymbol{G}}}$ and $\overline{\underline{\boldsymbol{k}}}$ are then used in the preconditioned Gauss-Seidel step referring to Eq. 2.96. The option used in the scope of this thesis will be the preconditioning strategy `pivotAll`, where each element $p_{j,i}$ of $\boldsymbol{P}$ is defined by

$$p_{j,i} := \begin{cases} 0 & \text{if } \overline{\underline{j}}_{j,i} = 0 \\ 1 & \text{if } \overline{\underline{j}}_{j,i} \neq 0 \end{cases} \quad , \tag{3.6}$$

with $\bar{\bar{j}}_{j,i}$ being the corresponding entry in the interval Jacobian matrix. Each variable interval is thus reduced in each equation depending on this variable. Three other preconditioning options have been implemented, but could not achieve any increase in efficiency compared to `pivotAll` in the test runs of this work. For the sake of completeness, they are presented in section 6.8.

Another contraction method that can be used in combination with Interval Newton or individually is HC4revise. The method reduces the variable intervals equation-wise, i.e., all variable intervals of one equation are led to consistency. Afterwards, the method continues with the next equation and the reduced variable intervals are directly employed. One contraction step involves one run through all equations. Note that the overall problem does not have to be consistent after a single contraction step as the newly reduced intervals might be inconsistent in the first equations. According to the definition of hull consistency, gaps are ignored and there is no return value of the external routine that allows for recognizing them. The python package that contains HC4revise is called pyibex developed by Desrochers (2015) and used in the novel hybrid approach. Both, Interval Newton and HC4revise can face issues to reduce intervals of variables that occur multiple times in an equation. Mathematical models of chemical engineering processes frequently contain such type of equations for example in reaction kinetics, thermodynamic properties, fugacity or activity coefficient calculations. If variables can not be further reduced by contraction, the current box will be split, and the iteration is quickly dominated by extensive splitting, becoming almost as slow as a classical branch and bound algorithm. One aim of this thesis is to develop a method that works well on this type of equations with a priority given to the reduction efficiency rather than the computational time of one contraction step, because extensive splitting might slow down the algorithm even more than a slightly more expensive but efficient contraction step.

Before the newly developed method Bnormal is introduced, a small example typical for chemical engineering problems shall point out this issue. The mole fraction y of a binary mixture's low-boiling component in the vapor phase that is in phase equilibrium with a liquid phase can be calculated by

$$y = \frac{\alpha \cdot z}{1 + (\alpha - 1) \cdot z} \quad , \tag{3.7}$$

where z denotes the low-boiler's mole fraction in the liquid phase, and the relative volatility α is assumed to be constant. For the variables y, z, α the following intervals are known $\overline{\underline{y}} = [0.1, 0.2]$, $\overline{\underline{z}} = [0, 1]$ and $\overline{\underline{\alpha}} = [1, 2]$. How much can Interval Newton and HC4revise reduce these intervals? First, the Interval Newton is examined. Therefore, Eq.3.7 is brought into the zero-form

$$f(y,z,\alpha) = y - \frac{\alpha \cdot z}{1 + (\alpha - 1) \cdot z} = 0 \quad . \tag{3.8}$$

No preconditioning shall be used and $\boldsymbol{x}_c$ equals the variable intervals' mid points so that

$$x_c = \begin{pmatrix} 0.15 \\ 0.5 \\ 1.5 \end{pmatrix}$$

$$f(\boldsymbol{x}_c) = -0.45$$

$$\frac{\partial f}{\partial y} = j_{1,1} = 1 \quad \frac{\partial f}{\partial z} = j_{1,2} = -\frac{\alpha}{(1 + (\alpha - 1) \cdot z)^2} \quad \frac{\partial f}{\partial \alpha} = j_{1,3} = \frac{z^2 - z}{(1 + (\alpha - 1) \cdot z)^2} \quad .$$

Now, the Gauss-Seidel step

$$\overline{\underline{x}}_i^{(k+1)} := \overline{\underline{x}}_i^{(k)} \cap x_{c,i}^{(k)} - \frac{\overline{\underline{f}}_i(\boldsymbol{x}_c) + \sum_{j=1}^{i-1} \overline{\underline{j}}_{ij} \cdot (\overline{\underline{x}}_j^{(k+1)} - x_{c,j}^{(k+1)}) + \sum_{j=i+1}^{n} \overline{\underline{j}}_{ij} \cdot (\overline{\underline{x}}_j^{(k)} - x_{c,j}^{(k)})}{\overline{\underline{j}}_{ii}}$$

is exemplary applied for the first contraction step of $\overline{\underline{z}}$

$$\overline{\underline{z}}^{(1)} = [0, 1] \cap 0.5 - \frac{-0.45 + 1 \cdot ([0.1, 0.2] - 0.15) + [-1, 1] \cdot ([1, 2] - 1.5)}{[-2, \ -0.25]} \tag{3.9}$$

$$\overline{\underline{z}}^{(1)} = [0, 1] \cap 0.5 - [-0.4, \ 4] = [0, 0.9] \quad . \tag{3.10}$$

The other intervals $\overline{\underline{y}}$ and $\overline{\underline{\alpha}}$ can not be reduced at all. In fact, without bisection none of the tested contraction methods are able to reduce $\overline{\underline{y}}$ and $\overline{\underline{\alpha}}$ any further so that the analysis focuses on the reduction of $\overline{\underline{z}}$. After the first Newton step $\overline{\underline{z}}$ is reduced to $[0, 0.9]$, and the method needs in total 16 steps to finish with a consistent interval of $\overline{\underline{z}}^{(15)} = [0, 0.54]$. Why $\overline{\underline{z}}^{(16)}$ can not be further tightened? Some of the intervals in Eq. 3.9 result from the interval extended versions of the

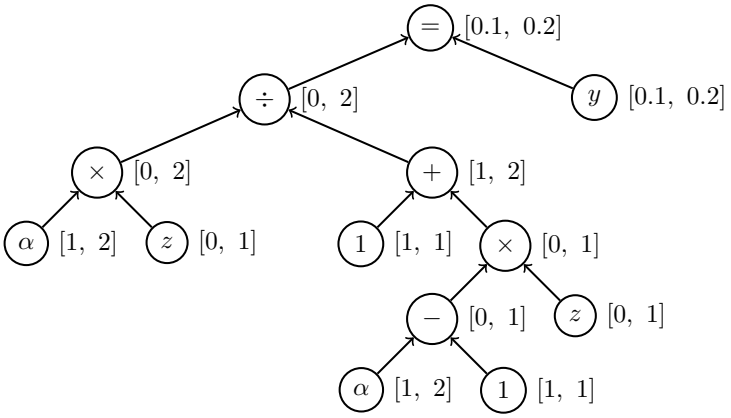
variables' derivatives. Evaluated with the usage of natural IA (see section 2.6.1), they greatly overestimate the actual range of validity due to interval dependency as can be seen in Eq. 3.11 and Eq. 3.12.

$$\overline{\partial f/\partial z} = \frac{\overline{\alpha}}{(1 + (\underline{\alpha} - 1) \cdot \underline{z})^2} \tag{3.11}$$

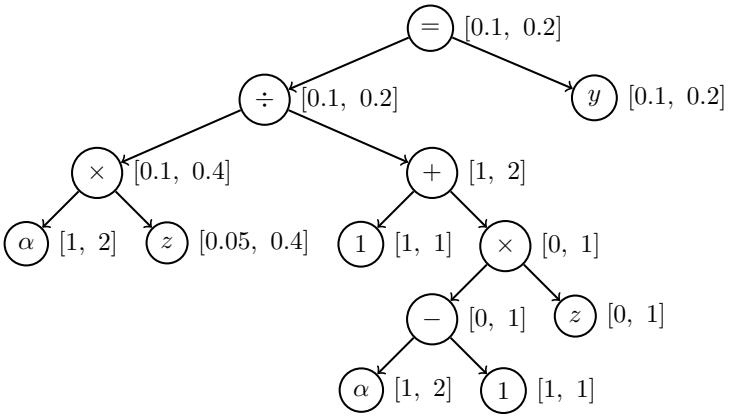
$$\underline{\partial f/\partial z} = \frac{\underline{\alpha}}{(1 + (\overline{\alpha} - 1) \cdot \overline{z})^2} \tag{3.12}$$

To determine the maximum value of this derivative by IA, the upper bound of $\overline{\alpha}$ is used in the numerator of Eq. 3.11 and its lower bound in the denominator. In real arithmetic both instances of the same variable must have the identical value. Hence, in real arithmetic $\overline{\alpha}$ and $\underline{\alpha}$ are out of scope. Interval dependency also occurs in the determination of $\underline{\partial f/\partial \alpha}$. In consequence, these coarse intervals result in a wide interval extended Gauss-Seidel step taken from x_c. Regarding the current example, the width of the Gauss-Seidel step in the 16th contraction step is at least as wide as the width of the 15th contraction step and no tightening can be obtained.

Next, HC4revise is applied on the given problem. Figure 3.2 shows the first contraction step, which covers forward and backward evaluation. The reduced interval $\overline{z}$ is $[0.05, 0.4]$ and therefore already essentially smaller than the consistent interval resulting from Interval Newton. The method reaches consistency after the sixth contraction step at $\overline{z}^{(6)} = [0.05, 0.25]$. This method works better than the Interval Newton, because no IA-based linear approximation is used anymore. Nevertheless, one can see in the first contraction step that in backward evaluation the intervals of both z instances differ as shown in figure 3.2. The one in the numerator changes to $[0.05, 0.4]$, while the one in the denominator can not be reduced. Hence, before the second contraction step the latter is updated to $[0.05, 0.4]$. Nevertheless, its reduction potential is not directly used in the first contraction step and makes more reductions steps inevitable. A problem of both methods is the missing linkage between multiple instances of the same variable. For this reason, a method was developed that attempts to preserve precisely this linkage and thus reduce the original box within a contraction step further compared to Interval Newton and HC4revise. This self-developed method is called Bnormal and belongs to the group of box consistency methods, explained in sec-



(a) Forward evaluation.



(b) Backward evaluation.

Fig. 3.2: HC4revise applied on the equation $y = \frac{\alpha \cdot z}{1+(\alpha-1)\cdot z}$ in the box $\overline{z} \times \overline{y} \times \overline{\alpha} = [0, 1] \times [0.1, 0.2] \times [1, 2]$.

tion 2.6.3. Let us focus on the reduction of $\overline{z}$ again. Eq. 3.7 can be divided into a z-dependent part $g(z)$ and a z-independent part b

$$\underbrace{y}_{b} = \underbrace{\frac{\alpha \cdot z}{1 + (\alpha - 1) \cdot z}}_{g(z)} \quad . \tag{3.13}$$

If b and $g(z)$ are evaluated by IA in $\overline{\boldsymbol{x}}$, three cases can occur:

$$\overline{b} \begin{cases} \subset \overline{g}(\overline{z}) & \overline{z} \text{ can be reduced until } \overline{b} = \overline{g}(\overline{z}) \\ \supseteq \overline{g}(\overline{z}) & \text{No reduction of } \overline{z} \text{ is possible} \\ \cap \overline{g}(\overline{z}) = \emptyset & \text{There is no solution in } \overline{\boldsymbol{x}} \quad . \end{cases} \tag{3.14}$$

In case two and three the box can not be further reduced, because $\overline{g}(\overline{z})$ is tighter or as tight as $\overline{b}$ (case two) or $\overline{g}(\overline{z})$ and $\overline{b}$ do not intersect at all (case three) so that $\overline{\boldsymbol{x}}$ has no solution and can be discarded. However, case one provides the opportunity to tighten $\overline{z}$ and $\overline{g}(\overline{z})$ further until the condition $\overline{b} = \overline{g}(\overline{z})$ is met. If case one holds and $g(z)$ is a continuously differentiable, monotonously increasing or decreasing function in $\overline{z}$, then there is exactly one convex subset in $\overline{z}$ where the condition $\overline{b} = \overline{g}(\overline{z})$ is fulfilled. In the current example, case one is met and $g(z)$ is both differentiable and monotonously increasing in $\overline{z}$, since

$$\overline{\partial g / \partial z(\overline{\boldsymbol{x}})} = \frac{\overline{\alpha}}{(1 + (\overline{\alpha} - 1) \cdot \overline{z})^2} = [0.25, 2] \geq 0 \quad . \tag{3.15}$$

If all other variables in $g(z)$ are subsequently replaced by their intervals and only z is evaluated by real arithmetic, the maximum function value of this interval extended, univariate function $\overline{g}(z)$ can be found at $\overline{z}$, while its minimum value lies at $\underline{z}$. Figure 3.3 shows the chart of $\overline{g}(z)$ in $\overline{z}$. Hence, the upper bound of $\overline{z}$ can be decreased until the condition $\overline{b} = \underline{g}(\overline{z})$ is met (highlighted by the upper dot in figure 3.3) and the lower bound of $\overline{z}$ can be increased until $\underline{b} = \overline{g}(\underline{z})$ holds (highlighted by the lower dot in figure 3.3). This is the basic idea of Bnormal. Lower and upper bound are iteratively increased and decreased, respectively, by a univariate version of the Interval Newton method as described in section 2.6.3 until the conditions are met. The interval extended Newton steps of the upper

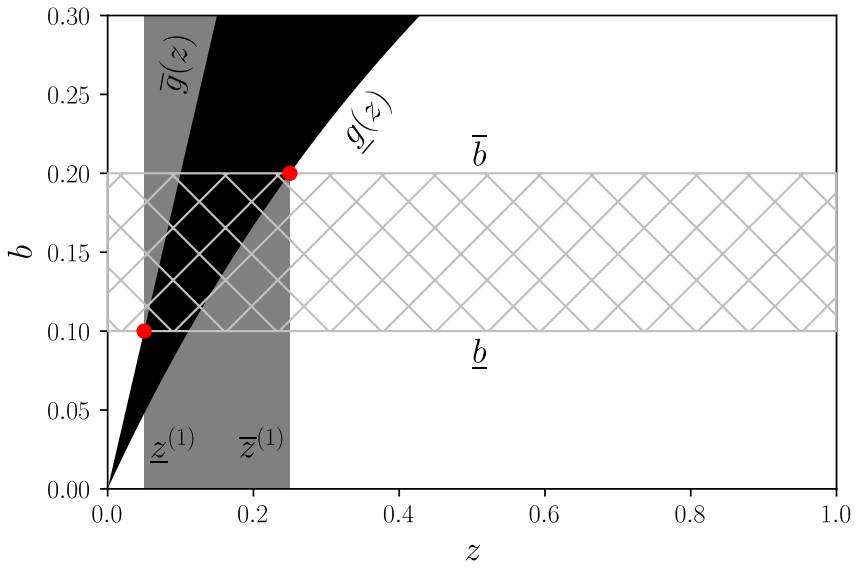


Fig. 3.3: Reduction of continuously differentiable, monotonously increasing, interval extended function $\overline{\underline{g}}(z) = \frac{[1,\,2]\cdot z}{1+([1,\,2]-1)\cdot z}$ in the initial range $\overline{\underline{z}}^{(0)} = [0,\,1]$ by Bnormal.

bound iteration $\overline{\underline{z}}_{ub}$ and the lower bound iteration $\overline{\underline{z}}_{lb}$ are

$$\overline{\underline{z}}_{ub}^{(0)} := \overline{\underline{z}} \tag{3.16}$$

$$\overline{\underline{z}}_{ub}^{(k+1)} := m(\overline{\underline{z}}_{ub}^{(k)}) - \frac{\underline{g}(m(\overline{\underline{z}}_{ub}^{(k)})) - \overline{b}}{\partial\overline{\underline{g}}/\partial z(\overline{\underline{z}}_{ub}^{(k)})} \cap \overline{\underline{z}}_{ub}^{(k)} \tag{3.17}$$

$$\overline{\underline{z}}_{lb}^{(0)} := \overline{\underline{z}} \tag{3.18}$$

$$\overline{\underline{z}}_{lb}^{(k+1)} := m(\overline{\underline{z}}_{lb}^{(k)}) - \frac{\overline{\underline{g}}(m(\overline{\underline{z}}_{lb}^{(k)})) - \underline{b}}{\partial\overline{\underline{g}}/\partial z(\overline{\underline{z}}_{lb}^{(k)})} \cap \overline{\underline{z}}_{lb}^{(k)} \quad . \tag{3.19}$$

Both iterations are independent of one another and start from the initial interval $\overline{\underline{z}}^{(0)}$. They terminate when both $\overline{\underline{z}}_{ub}^{(k)}$ and $\overline{\underline{z}}_{lb}^{(k)}$ are degenerate according to the user-specified tolerance. Then the reduced interval $\overline{\underline{z}}^{(k+1)}$ is

$$\overline{\underline{z}}^{(k+1)} := [\underline{z}_{lb}^{(k)}, \overline{z}_{ub}^{(k)}] \quad . \tag{3.20}$$

Tab. 3.1: Contraction steps k and CPU time required to reduce $\overline{z} = [0, 1]$ until consistency is achieved in the equation $0 = y - \frac{\alpha \cdot z}{1+(\alpha-1)\cdot z}$ with $\underline{y} = [0.1, 0.2]$ and $\underline{\alpha} = [1, 2]$. Tested on a Notebook Intel i7 Processor (8x 1.8 GHz, 8. Generation) and 16 GB RAM with a Linux operating system.

Interval Newton			HC4revise			Bnormal		
k	$\overline{z}$	CPU (ms)	k	$\overline{z}$	CPU (ms)	k	$\overline{z}$	CPU (ms)
1	[0, 0.81]	0.66	1	[0.05, 0.40]	0.27	1	[0.05, 0.25]	3.44
2	[0, 0.73]	1.17	2	[0.05, 0.28]	0.53	2	[0.05, 0.25]	6.73
	⋮			⋮				
16	[0, 0.54]	8.90	6	[0.05, 0.25]	1.56			

For the current example $\underline{z}$ and $\overline{z}$ can even be derived analytically

$$\overline{b} = \overbrace{\frac{\underline{\alpha} \cdot \overline{z}}{1 + (\overline{\alpha} - 1) \cdot \overline{z}}}^{\underline{g}(\overline{z})} \tag{3.21}$$

$$0.2 = \frac{1 \cdot \overline{z}}{1 + (2 - 1) \cdot \overline{z}} \tag{3.22}$$

$$\overline{z} = \frac{0.2}{1 - 0.2} = 0.25 \tag{3.23}$$

$$\underline{b} = \overbrace{\frac{\overline{\alpha} \cdot \underline{z}}{1 + (\underline{\alpha} - 1) \cdot \underline{z}}}^{\overline{g}(\underline{z})} \tag{3.24}$$

$$0.1 = \frac{2 \cdot \underline{z}}{1 + (1 - 1) \cdot \underline{z}} \tag{3.25}$$

$$\underline{z} = 0.05 \quad . \tag{3.26}$$

Bnormal needs only one contraction step to reduce $\overline{z}$ to $[0.05,\ 0.25]$, in contrast to the other two methods. A second contraction step is necessary to ensure consistence (like the last steps of the other contraction methods). Table 3.1 summarizes the results of all three contraction methods applied on this example. It can be seen that although HC4revise requires more contraction steps, overall less time is

needed to obtain the identical consitent intervals as Bnormal. Hence, a contraction step using Bnormal is much more expensive than one using HC4revise and even Interval Newton. However, it is not always the case that the consistency methods become consistent at the same intervals, as has already been illustrated by the 2D/3D examples in section 2.6.3. The extra effort may therefore be justified if it allows the box to be reduced even further. As a consequence, combinations of the contraction methods will also be examined later in this work.

Concerning Bnormal, we have not yet discussed the cases where $g(z)$ in $\overline{\underline{z}}$ is continuously differentiable and monotone decreasing, non-smooth or non-monotone. For the reduction of $\overline{\underline{z}}$ in continuously differentiable, monotone decreasing functions $g(z)$, the following two conditions are true: $\underline{b} = \overline{g}(\overline{z})$ and $\overline{b} = \underline{g}(\underline{z})$. Alternatively, they can be reformulated as monotonously increasing functions $g^*(z)$:

$$-b = \overbrace{-g(z)}^{g^*(z)} \quad . \tag{3.27}$$

For non-smooth and non-monotone functions a filter algorithm has been developed. Firstly, regions, where $\overline{\underline{g}}(z)$ is non-smooth, are filtered out of the domain by a bisecting algorithm that splits up $\overline{\underline{z}}$ into subintervals and keeps – for the moment – only those with finite derivative intervals $\partial\overline{\underline{g}}/\partial z$. Secondly, the smooth subintervals are bisectioned analogously until only monotone subintervals are left. This is done by evaluating $\partial\overline{\underline{g}}/\partial z$ in the current subinterval, if the derivative values are all greater or equal to zero it is a monotonously increasing function and if it is lower or equal to zero it is a monotone decreasing function. Constant functions with a derivative value of zero are counted to the monotonously increasing functions. Finally, the criteria for the already described reduction procedure are fulfilled and each subinterval is individually reduced. Multiple subintervals can result from the filtering process that do not share any bound with another. Some of them might be removed, because there is no intersection of $\overline{\underline{g}}(z)$ with $\overline{\underline{b}}$ in them. Others might encompass an infeasible gap. The number of boxes is limited by the hyperparameter "maxBoxNo". If there is enough capacity, an infeasible gap is removed from the original interval and the set of both subintervals is returned, resulting in two subboxes. Otherwise, Bnormal returns only the hull consistent box similar to Interval Newton and HC4revise. Nevertheless, such intervals are labeled for later

use in splitting. Intervals, where $\overline{\underline{g}}(z)$ is continuous but non-monotone can also occur for example in

$$\overline{\underline{g}}(z) = [-1, 1] \cdot z \quad . \tag{3.28}$$

They are sorted out by the filtering algorithm and separately tightened. The intervals are simply divided into a certain number of subintervals that the user has to set by the hyperparameter "resolution". The function $\overline{\underline{g}}(z)$ is then evaluated in each subinterval and checked for an intersection with $\overline{\underline{b}}$. For efficiency, it is started from the lowest subinterval in increasing order and the method stops when a subinterval $\overline{\underline{z}}_{lb}$ is found where $\overline{\underline{g}}(\overline{\underline{z}}_{lb}) \cap \overline{\underline{b}} \neq \emptyset$. The lowest bound of $\overline{\underline{z}}_{lb}$ is kept. Next, it checks the highest subinterval and moves downwards until it finds the first subinterval $\overline{\underline{z}}_{ub}$ where $\overline{\underline{g}}(\overline{\underline{z}}_{ub}) \cap \overline{\underline{b}} \neq \emptyset$. Intervals between $\overline{\underline{z}}_{lb}$ and $\overline{\underline{z}}_{ub}$ are not further checked to save time and the interval $[\underline{z}_{lb}, \overline{z}_{ub}]$ is returned. The implementation and further details on the method Bnormal have been published in detail in Bublitz et al. (2021b). The most relevant change to the current version is that the univariate Interval Newton method replaces the former bisecting algorithm for the reduction of intervals in continuously differentiable monotone functions because of computational efficiency. The algorithm is part of the Appendix figure A.3.

Let us turn once again to the ***V**apor **L**iquid **E**quilibrium* (VLE) example of Eq. 3.7. In Bnormal, the variable z has been evaluated by real-valued arithmetic in the interval extended function $\overline{\underline{g}}(z)$, which preserved the fact that both of its instances must have the same value in $\overline{\underline{g}}(z)$. Hence, only one contraction step was necessary to achieve the exact image set of $\overline{\underline{g}}(z)$ so that

$$\overline{\underline{b}} = \overline{\underline{g}}(\overline{\underline{z}}^{(1)}) = \overline{\underline{g}}^{*}(\overline{\underline{z}}^{(1)}) \quad . \tag{3.29}$$

Further reduction of $\overline{\underline{z}}^{(1)}$ through $\overline{\underline{g}}(z)$ without splitting is impossible, unless $\overline{\underline{b}}$ or $\overline{\underline{g}}(z)$ are modified in some way. Going back to Eq. 3.21 and 3.24 one can see that α is also affected by interval dependency. In section 2.6.1 an effective method called refinement has been introduced, which can further tighten the range of interval extended function evaluations, when interval dependency is present. For this purpose, a method termed tighten_bounds has been implemented as part of the hybrid approach that makes use of these refinements. Whenever a function needs to be evaluated by IA, tighten_bounds checks variable by variable, how many

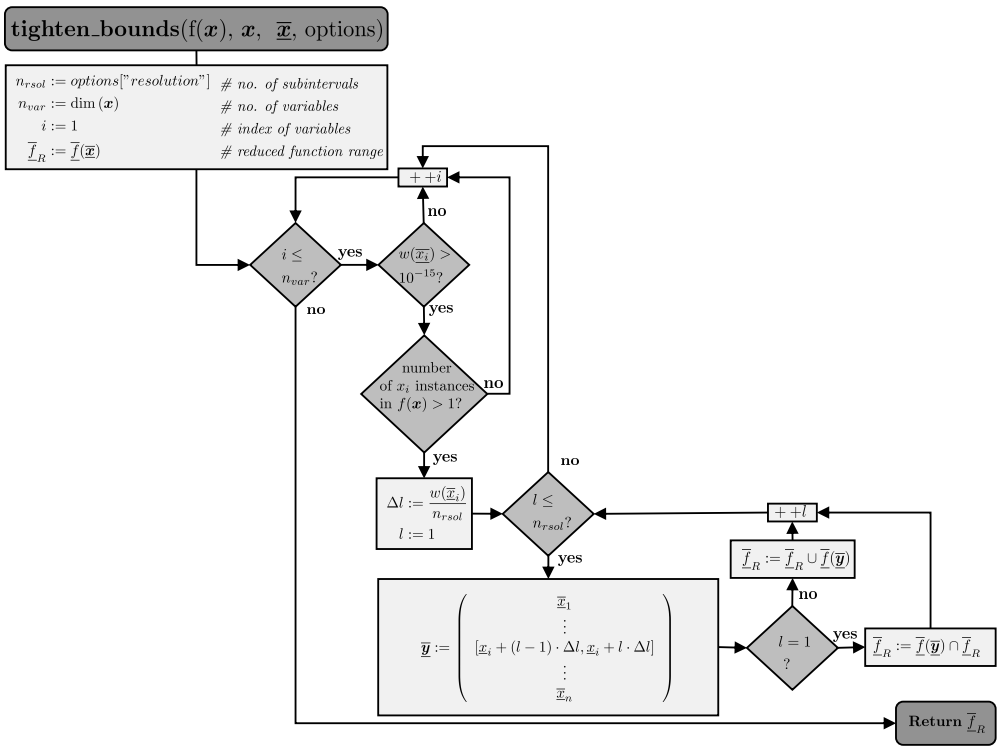


Fig. 3.4: The algorithm of tighten_bounds.

occurrences a variable has and whether its interval is not degenerate yet. Only if a variable has multiple occurrences and is not solved, refinements are useful. When such a variable has been found its interval is subdivided into a certain number of subintervals specified again by the hyperparameter "resolution". The higher the "resolution" is set the closer comes the function value range from the interval extended function to the actual image set of the function. Figure 3.4 shows the algorithm of tighten_bounds. The evaluation of $\underline{\overline{b}}$ and $\underline{\overline{g}}(z)$ can be done by tighten_bounds. The interval $\underline{\overline{b}}$ is not affected since b only consists of y, which has only one occurrence. However, $\underline{\overline{g}}(z)$ can be tightened. Is the "resolution" parameter set to 5, $\underline{\overline{z}}$ can be further narrowed to $\underline{\overline{z}}^{(1)} = [0.052, 0.208]$ after the first contraction step, which is proofed to be consistent in the second. Table 3.2 presents the results of this refinement. For completion this refinement can even be solved analytically. Eq. 3.21 and 3.24 are reformulated and evaluated for each

Tab. 3.2: Resulting subintervals $\overline{\underline{z}}_l$ from refinement of $\overline{\underline{\alpha}}$ with five subintervals, analytically solved by Eq. 3.30 and 3.31.

l	$\overline{\underline{\alpha}}_l$	$\underline{z}_l$	$\overline{z}_l$
1	[1.0, 1.2]	0.083	0.208
2	[1.2, 1.4]	0.072	0.179
3	[1.4, 1.6]	0.066	0.156
4	[1.6, 1.8]	0.057	0.139
5	[1.8, 2.0]	0.052	0.125

subinterval α_l

$$\overline{z}_l = \frac{\overline{b}}{\underline{\alpha}_l - \overline{b} \cdot (\overline{\alpha}_l - 1.0)} \quad l = 1 \ldots 5 \tag{3.30}$$

$$\underline{z}_l = \frac{\underline{b}}{\overline{\alpha}_l - \underline{b} \cdot (\underline{\alpha}_l - 1.0)} \tag{3.31}$$

The reduced interval $\overline{\underline{z}}$ equals

$$\overline{\underline{z}} = [\min(\{\underline{z}_{l=1}, \ldots, \underline{z}_{l=5}\}),\ \max(\{\overline{z}_{l=1}, \ldots, \underline{z}_{l=5}\})] = [0.052, 0.208] \quad , \tag{3.32}$$

which agrees with the algorithm's results. However, this additional work results in a CPU time of 16.25 ms that the algorithm (Bnormal combined with tighten_bounds) needs to process the two contraction steps. The test was absolved on the same computer as the ones from the pure contraction methods shown in table 3.1. On the one hand, Bnormal is significantly slower in case tighten_bounds is applied, on the other hand some extensive box splitting might be prevented in later stages of the hybrid approach.

A full reduction step consists of multiple contraction steps from either one contraction method on its own or a combination of all three methods HC4revise, Interval Newton and Bnormal. The reduction step is finished when a box is either solved, proved to be empty, or consistent. The algorithm used for the contraction that integrates the three methods above, is called contract_box and shown in figure 3.5.

As input arguments it takes one box $\underline{\overline{\boldsymbol{x}}}$, the functions $\boldsymbol{f}$ and variables $\boldsymbol{x}$ related to the problem $\boldsymbol{f}(\boldsymbol{x}) = 0$, a dictionary *options* with iteration specific data and two parameters p and l. Parameter p is a tuple that contains information about the so-called parent of the current box. The parent box of an unsplit box is the initial box. Whenever a box is split the reduction step before the split and its index regarding the current box order are stored so that it can always be reloaded in upcoming reduction steps. Former boxes will be needed for a certain splitting technique explained in section 3.5. The parameter l is the index of the box itself in the current order of boxes. There are further properties of a box, which describe its current state that are in contract_box abbreviated by

- s := boolean, `true` if box is solved
- d := boolean, `true` if box is discontinuous
- c := boolean, `true` if box is consistent to contraction
- $unique_{nwt}$:= boolean, `true` if box has unique solution based on the interval newton contractor
- $unique_{bc}$:= boolean, `true` if box has unique solution based on the box consistency contractor .

A solved box is not further processed in any of the hybrid approach's reduction steps. A discontinuous box contains at least one gap that can be treated by splitting later. A consistent but not solved box can not be further reduced by contraction and it is continued with the next steps of the hybrid approach. The two parameters $unique_{nwt}$ and $unique_{bc}$ are necessary for the unique solution test based on Interval Newton or Bnormal. This test was already introduced in section 2.6.2. Its usage here as a termination criterion will be explained in section 3.2. Initially, s, d and c are set to `false` and turned to `true` if a box is solved, proved to contain a gap or can not be further reduced by contraction, respectively.

For clarity and comprehensibility the methods get_allowed_boxes, consistent, root_inclusion, solved and unique_solution_test, which contract_box invokes internally, are only briefly explained. Their comprehensive algorithms are part of Appendix A. As test results of the thesis of Ebert (2021, pp. 34-39) showed,

HC4revise seems to be the fastest out of the three methods regarding CPU time per reduction step. This agrees with the already shown measurements from table 3.1, although it does not state anything about the reduction efficiency. However, when HC4revise is selected, contract_box starts with it and cycles through all equations until either hull consistency of the complete problem is reached, the box is solved, or proven to be empty. Empty and solved boxes are directly returned from contract_box, while a hull consistent box $\overline{\underline{\boldsymbol{y}}}$ is further processed by the other contractors, when they are selected. If this is the case, the algorithm iterates through all variables, for each one contraction step by Interval Newton and one by Bnormal is performed. Solved intervals according to the selected relative and absolute tolerances are skipped and if an interval is empty, the box has no solution and contract_box returns an empty set. Interval Newton also contains the described preconditioning and Bnormal the additional option tighten_bounds. They consider gaps so that multiple subintervals can be returned in the set $\{\overline{\underline{z}}_{v_i}\}$ and $\{\overline{\underline{a}}\}_{v_i}$ of the current variable v_i. To keep the number of subintervals low, root_inclusion applied on the whole system in the respective subboxes follows the contraction step. In case $\{\overline{\underline{z}}_{v_i}\}$ contains a single interval, the processed box $\overline{\underline{\boldsymbol{y}}}$ is directly updated to improve the reduction efficiency of the subsequent variable intervals. Otherwise the hull that encompasses all subintervals is determined and the related interval is updated to $\overline{\underline{y}}_{v_i}$. After all variables have been contracted in one cycle there are n_{var} sets $\{\overline{\underline{z}}_{v_i}\}$ containing all their reduced subintervals. Through the cross product of all these sets a set of reduced boxes $\{\overline{\underline{\boldsymbol{z}}}\}$ is obtained. If each $\{\overline{\underline{z}}_{v_i}\}$ consists of only one reduced interval there will be only one reduced box in $\{\overline{\underline{\boldsymbol{z}}}\}$. This one is then applicable for a check of being solved by the two methods solved and unique_solution_test.

A solved box is returned by contract_box, while an unsolved box and multiple boxes in $\{\overline{\underline{\boldsymbol{z}}}\}$ are checked for consistency next. In consistent the reduced box $\overline{\underline{\boldsymbol{y}}}$ that corresponds to the hull of all current subboxes is compared to the initial box $\overline{\underline{\boldsymbol{x}}}$. If all intervals remain constant regarding the selected relative and absolute tolerances, $\overline{\underline{\boldsymbol{y}}}$ is termed consistent, c is set to `true` and the reduction step is almost done. In the present implementation $\{\overline{\underline{\boldsymbol{z}}}\}$ is not directly computed, because if many discontinuities are present in $\overline{\underline{\boldsymbol{x}}}$, this can cause a high number of boxes. In the complete procedure the number of boxes is always restricted by the parameter "maxBoxNo" that is part of the dictionary *options*. The method get_allowed_boxes

finally checks how many subboxes can be built without exceeding "maxBoxNo". Hence, the gap of some intervals might be considered and they split into their feasible parts until "maxBoxNo" is reached, while others are not split and set to their intervals from $\overline{\underline{y}}$. If gaps can not be used for interval reduction, the boxes from $\{\overline{\underline{z}}\}$ are labeled as discontinuous and may be split later, whenever capacity becomes available. When the allowed $\{\overline{\underline{z}}\}$ has been generated, it is returned together with a set of the boxes' properties. Consistent boxes will be tested by the root-finding algorithm.

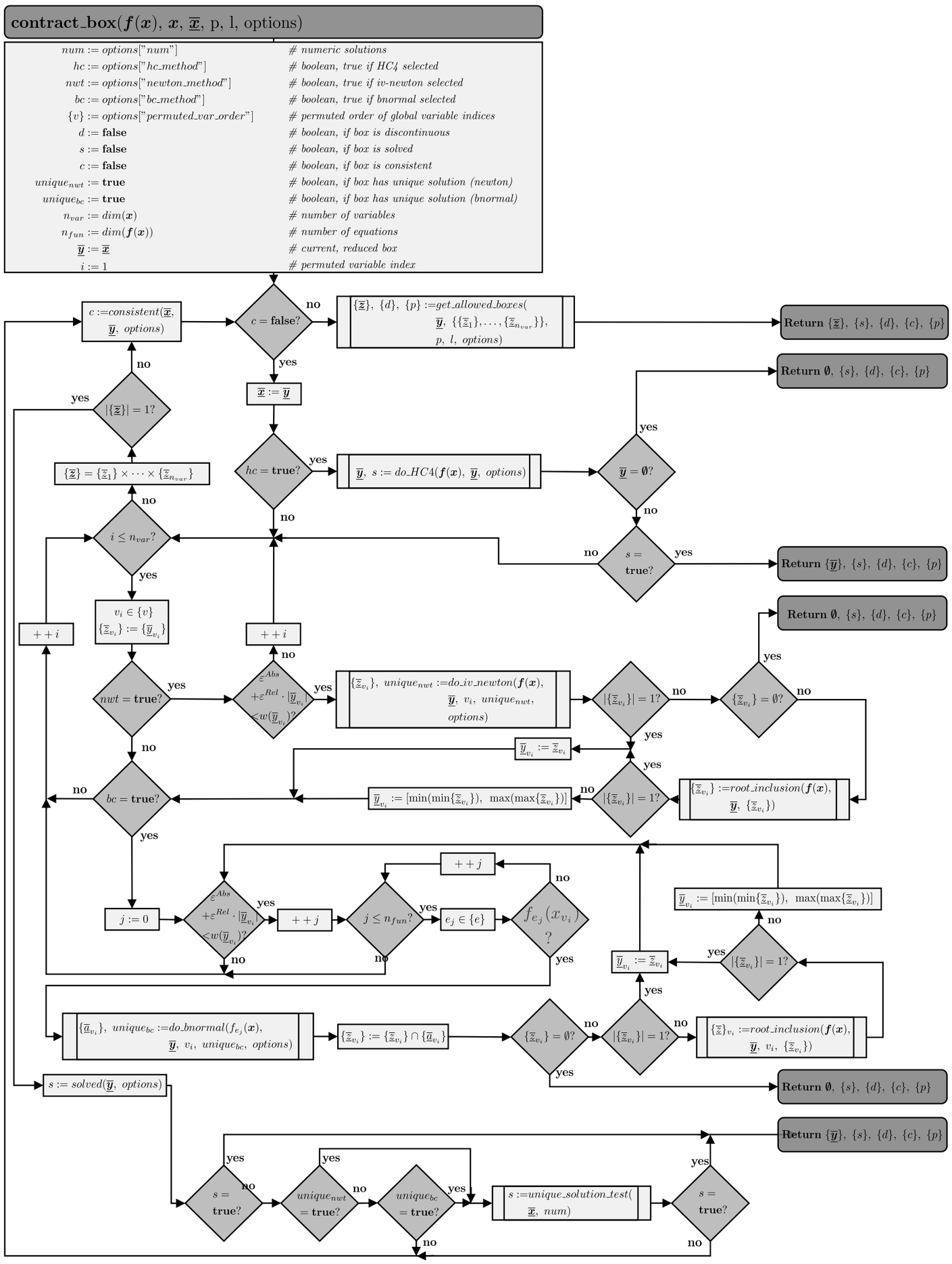


Fig. 3.5: The algorithm of contract_box.

3.2 Termination

The contraction of a box continues until either consistency is achieved, the maximum number of contraction steps selected by the hyperparameter "redStepMax" is reached, it has been proven that the box contains a unique solution and this solution has already been determined numerically, or the box has been narrowed so much that all variable intervals are degenerate according to the set tolerances, i.e., it contains a solution and is sufficiently close to it. Recommendations for the settings of the algorithm's hyperparameters for a new system will be given in Appendix A.3.1. Numerical iteration methods mostly use relative tolerances ε^{Rel} as stopping criteria, because they consider the iteration variables' order of magnitude. Using an absolute tolerance ε^{Abs} instead bears the risk of reducing variables with a low order of magnitude to a lesser degree as their value may be lower than the absolute tolerance value. Hansen & Walster (2003, pp. 176-177) show that a relative stopping criterion, such as

$$\varepsilon^{Rel} \geq \frac{w(\overline{\underline{x}})}{|\overline{\underline{x}}|} \quad , \tag{3.33}$$

should never be applied as a sole criterion for IA since for a typical choice of $\varepsilon^{Rel} \ll 1$, the constraint 3.33 will never be met by a variable interval $\overline{\underline{x}}$ that contains zero. For such an interval the condition

$$1 \leq \frac{w(\overline{\underline{x}})}{|\overline{\underline{x}}|} \leq 2 \tag{3.34}$$

always holds. Hence, this hybrid approach employs the formula implemented in the method *numpy.isclose()* of the python package numpy (Harris et al., 2020):

$$(\varepsilon^{Abs} + \varepsilon^{Rel} \cdot |\overline{\underline{x}}|) \geq w(\overline{\underline{x}}) \quad . \tag{3.35}$$

Condition 3.35 needs to be met by all variable intervals $\overline{\underline{x}}$ of a box $\overline{\underline{\boldsymbol{x}}}$ before it is tagged as "degenerate". Nevertheless, a degenerate interval $\overline{\underline{x}}$ is still further tightened by decreasing the values of ε^{Abs} and ε^{Rel} for this specific variable i by an order of magnitude ($\varepsilon_i^{Abs} = 0.1 \cdot \varepsilon_i^{Abs}$, $\varepsilon_i^{Rel} = 0.1 \cdot \varepsilon_i^{Rel}$) whenever it reaches degeneration because the reduction of other variable intervals might strongly depend on it and would never fulfill condition 3.35 otherwise. Let us look at an

example

$$a = 10^7 \cdot b \quad , \tag{3.36}$$

with $\underline{\overline{b}} = [1.9, 2.0] \times 10^{-6}$. Assuming absolute and relative tolerances of $\varepsilon^{Abs} = \varepsilon^{Rel} = 10^{-6}$, the interval $\underline{\overline{b}}$ is already degenerate. However, in case variable a only occurs in Eq. 3.36, it can not be reduced any further than $\underline{\overline{a}} = 10^7 \cdot \underline{\overline{b}} = [1.9, 2]$ and would thus never fulfill the required tolerances. Instead, the algorithm would start to split $\underline{\overline{a}}$ in many subintervals associated to many subboxes until the width of each subinterval satisfies condition 3.35. However, successively decreasing ε^{Rel} and ε^{Abs} for degenerate intervals such as $\underline{\overline{b}}$ prevents this exhaustive splitting. To ensure that the successive reduction of the tolerance values does not unnecessarily slow down the methodology though, it is only performed until both tolerances drop below a value of 10^{-15}. Of course, in the worst case, increased splitting can occur even beyond this limit resulting in many solved neighbour boxes fulfilling condition 3.35. However, such boxes are then merged in a suitable way to keep the number of boxes small. Later in this section this will be explained in detail. For intervals containing zero and those close to zero, the absolute tolerance ε^{Abs} dominates. For higher orders of magnitude the relative tolerance ε^{Rel} becomes restrictive instead. The consistency check relies on analogue conditions 3.37 and 3.38, for the lower and upper bounds of the intervals

$$(\varepsilon^{Abs} + \varepsilon^{Rel} \cdot \max\{|\underline{x}^{(k+1)}|, |\underline{x}^{(k)}|\}) \geq w(|\underline{x}^{(k+1)} - \underline{x}^{(k)}|) \tag{3.37}$$

$$(\varepsilon^{Abs} + \varepsilon^{Rel} \cdot \max\{|\overline{x}^{(k+1)}|, |\overline{x}^{(k)}|\}) \geq w(|\overline{x}^{(k+1)} - \overline{x}^{(k)}|) \quad . \tag{3.38}$$

A box is consistent if both conditions hold for all intervals. To filter singular or discontinuous points x_{sd} of a function $f(x)$ in $\underline{\overline{x}}$ by the contraction method Bnormal, the filter algorithm stops bisecting when condition 3.39 is fulfilled by the interval $\underline{\overline{x}}_{sd}$ with $x_{sd} \in \underline{\overline{x}}_{sd}$.

$$(0.1 \cdot \varepsilon^{Abs} + 0.1 \cdot \varepsilon^{Rel} \cdot |\underline{\overline{x}}_{sd}|) \geq w(\underline{\overline{x}}_{sd}) \tag{3.39}$$

Absolute and relative tolerances are multiplied by 0.1 so that condition 3.39 is more restrictive than condition 3.35 to prevent cutting away a potential solution, close to x_{sd}. Finally, Bnormal's univariate Interval Newton terminates when either the currently processed interval $\underline{\overline{z}}$ is degenerate, or the relevant bounds of $\underline{\overline{g}}(\underline{\overline{z}})$ and $\underline{\overline{b}}$ are almost equal, e.g., the interval associated with their distance is degenerate.

In case of a univariate, monotone increasing function $g(z)$ these distances are $|\overline{g}(\underline{z}) - \underline{b}|$ and $|\underline{g}(\overline{z}) - \overline{b}|$ for lower and upper bound iteration of $\underline{z}$ respectively (see Eq. 3.17 to 3.20). For both, condition 3.35 is applied.

Solving a problem $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ on an initial box $\underline{\overline{\boldsymbol{x}}}^{(0)}$, a reduced box $\underline{\overline{\boldsymbol{x}}}$ with $\underline{\overline{\boldsymbol{x}}} \subset \underline{\overline{\boldsymbol{x}}}^{(0)}$ is tagged as "solved" when one of the two conditions holds:

1) The box $\underline{\overline{\boldsymbol{x}}}$ is degenerate according to condition 3.35 and contains a solution $\boldsymbol{x}^*$ to $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, i.e.,
$$\boldsymbol{0} \in \underline{\overline{\boldsymbol{f}}}(\underline{\overline{\boldsymbol{x}}}) \quad . \tag{3.40}$$

2) The box fulfills a unique solution test and a solution $\boldsymbol{x}^*$ with $\boldsymbol{x}^* \in \underline{\overline{\boldsymbol{x}}}$ has been determined numerically beforehand by the root-finding algorithm.

Hence, if a box is degenerate, the condition 3.40 is checked. However, if the tolerances ε^{Abs} and ε^{Rel} are chosen too small, a box might be extensively split before it fulfills condition 3.35. On one hand interval dependency can cause relatively wide intervals $\underline{\overline{\boldsymbol{f}}}(\underline{\overline{\boldsymbol{x}}})$ so that criterion 3.40 can also be falsely met by empty boxes close to the real solution. On the other hand, rounding errors in badly conditioned systems can cause very flat functions so that there is rather a solution interval than a single real-valued solution. To prevent a worst case, in which hundreds of boxes are returned as solved, a method has been developed to unify all neighbour boxes, i.e., boxes that differ only in one dimension and the according variable intervals share exactly one bound. A solved, unified box is marked as such in the result file and an alternative tolerance value ε^{uni} is returned
$$\varepsilon^{uni} = \max_{\underline{\overline{x}}_i \in \underline{\overline{\boldsymbol{x}}}} \frac{w(\underline{\overline{x}}_i)}{1 + |\underline{\overline{x}}_i|} \quad . \tag{3.41}$$
Based on condition 3.40, Eq.3.41 assumes identical absolute and relative tolerance values that equal ε^{uni}. On these terms, ε^{uni} is the actual tolerance that the unified box fulfills. Besides, the result file also states the variable, which requires this tolerance. The second case, in which a box is tagged as "solved", encompasses a successful unique solution test and a numerically determined real-valued solution situated within the box. For each contraction method a “specific unique solution

test" exists. In Interval Newton and HC4revise the box has a unique solution if

$$\overbrace{\Gamma_{nwt}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x}_c^{(k)}, \overline{\underline{\boldsymbol{x}}}^{(k)})}^{=\overline{\underline{\boldsymbol{x}}}^{(k+1)}} \subset \overline{\underline{\boldsymbol{x}}}^{(k)} \quad \text{or} \tag{3.42}$$

$$\overbrace{\Gamma_{hc}(\boldsymbol{f}(\boldsymbol{x}), \overline{\underline{\boldsymbol{x}}}^{(k)})}^{=\overline{\underline{\boldsymbol{x}}}^{(k+1)}} \subset \overline{\underline{\boldsymbol{x}}}^{(k)} \quad . \tag{3.43}$$

These cases have been introduced in see section 2.6.2 and 2.6.3. Whenever the contraction operator produces a box $\overline{\underline{\boldsymbol{y}}}$ that is completely in the interior of $\overline{\underline{\boldsymbol{x}}}^{(k)}$ this iteration has a fixed point and the sequence of contracted intervals will converge to it (Hansen and Walster, 2003, pp. 182-188). Such a box $\overline{\underline{\boldsymbol{y}}}$ equals $\overline{\underline{\boldsymbol{x}}}^{(k+1)}$ and makes the intersection with $\overline{\underline{\boldsymbol{x}}}^{(k)}$ in $\overline{\underline{\boldsymbol{x}}}^{(k+1)} := \overline{\underline{\boldsymbol{x}}}^{(k)} \cap \overline{\underline{\boldsymbol{y}}}$ redundant. In Interval Newton and HC4revise the complete problem $\boldsymbol{f}(\boldsymbol{x})$ is not reduced all at once. Instead, in Interval Newton the Gauss-Seidel step is performed variable-wise and in HC4revise the contractor is applied equation-wise so that the box is successively reduced and updated before the next Gauss-Seidel step or contraction in the subsequent equation. Hence, the relevant conditions that are checked in Interval Newton and HC4revise are

$$\Gamma_{nwt,j,i}(f_j(\boldsymbol{x}), \boldsymbol{x}_c^{(k)}, \overline{\underline{\boldsymbol{y}}}) \subset \overline{\underline{x}}_i^{(k)} \quad \text{and} \tag{3.44}$$

$$\Gamma_{hc,j,i}(f_j(\boldsymbol{x}), \overline{\underline{\boldsymbol{y}}}) \subset \overline{\underline{x}}_i^{(k)} \quad , \tag{3.45}$$

where i denotes the currently reduced variable by function j and $\overline{\underline{\boldsymbol{y}}}$ is the already updated box within one contraction step. When the Interval Newton is not preconditioned by `pivotAll`, only the case $j = i$ is checked, i.e., variable x_i is only reduced by $f_i(x_i)$, the corresponding function, which determines the diagonal element of the Jacobian matrix. If `pivotAll` is chosen, it suffices to find any x_i-dependent function $f_j(\boldsymbol{x})$ from $\boldsymbol{f}(\boldsymbol{x}) = 0$ that fulfills Eq. 3.44 in the current contraction step. In HC4revise the functions are iterated successively. Hence, as soon as Eq. 3.45 is true for one specific variable x_i this condition has not to be checked for any other x_i-dependent function anymore. Finally, a successful unique solution test is achieved if Eq. 3.44 or Eq. 3.45 hold for all variables x_i in the respective contraction step. Note that $\overline{\underline{x}}_i^{(k)}$ is always the interval from the last contraction step not the already updated interval $\overline{\underline{y}}_i$ that is directly used in the next Gauss-Seidel step or subsequent equation of HC4revise's contraction step.

In Bnormal the univariate version of the Interval Newton is used. Hence, Eq. 3.44 also applies here, where f_j is reformulated to

$$f_j(\boldsymbol{x}) = g_{j,i}(x_i) - b_{j,i} \quad . \tag{3.46}$$

Note that there exist as many reformulations of $f_j(\boldsymbol{x})$ as the number of variables the function depends on. The method Bnormal reduces variable by variable and uses all functions that a variable occurs in. Hence, a variable x_i must fulfill condition

$$\Gamma_{bc,j,i}(f_j(\boldsymbol{x}), \underline{\overline{\boldsymbol{y}}}) \subset \underline{\overline{x}}_i^{(k)} \tag{3.47}$$

in any x_i-dependent function $f_j(\boldsymbol{x})$ from $\boldsymbol{f}(\boldsymbol{x}) = 0$. Such a function needs to be found for all variables so that the condition for the contraction step is

$$\overbrace{\Gamma_{bc}(\boldsymbol{f}(\boldsymbol{x}), \underline{\overline{\boldsymbol{x}}}^{(k)})}^{=\underline{\overline{\boldsymbol{x}}}^{(k+1)}} \subset \underline{\overline{\boldsymbol{x}}}^{(k)} \quad . \tag{3.48}$$

The unique solution test is applied after each contraction step. Variables with already degenerate intervals are excluded from it. Whenever it has been proven that a box has a unique solution, it is checked if it contains one of the already numerically found solutions from step 3 of the hybrid approach. If this is the case, the box is tagged as "solved" and will not be further processed in the hybrid approach. When a box has not been tagged as "solved" after contraction, a consistency check is executed regarding the conditions 3.38 and 3.37 for lower and upper bound that both need to be fulfilled to name a box "consistent". Inconsistent boxes are further contracted, while consistent boxes are subsequently passed to the real-valued arithmetic solver as described in the next section. After cutting and splitting (step 4 and 5), the boxes are also checked for degeneracy. The already discussed criteria apply except that no “unique solution test" is applied. The consistency check after cutting is the same.

Tab. 3.3: Applied state-of-the-art solvers and their settings.

Solver	Options and Settings	Implementation
IPOPT	"linear_solver": "MA27"	casadi
	"linear_system_scaling": "MC19"	
	"mu_init": 10^{-1}	
	"warm_start_init_point": "yes"	
SLSQP	Default	scipy
Fsolve	Default	scipy
Newton	"scaling": "None"	Own Implementation
	"scaling": "MC29"	
	"scaling": "MC77"	

3.3 Root-finding Algorithm

IA-based contraction methods can reduce boxes to such an extent that they are either degenerate or consistent. A consistent box is generally bisected and the resulting subboxes are further contracted. In the hybrid approach a root-finding solver is started from the midpoint of a consistent box, before the bisection is started given, that the reduction may suffice already for a root-finding solver to converge. However, the suggested procedure can also be used as classical IA-based bisecting method. Setting the algorithm's hyperparameter "hybridApproach" to `false` simply skips the root-finding step. The available root-finding solvers and their applied settings in the upcoming sections are listed in table 3.3. Not explicitly stated settings are at their default values. The scaling procedures "MC77", "MC29" and "MC19" as well as the linear solver "MA27" are obtained from HSL and were briefly discussed in section 2.3 and 2.4.1. The user can specify the maximum number of iteration steps to be applied and the function tolerance ε^{fTol} for the solver's termination. An NLE is not solved as one large system, instead the NLE solver is applied on all the subsystems identified by the DM

decomposition (see section 2.4.2). The implementation from the python package casadi is used (Andersson et al., 2019). If the current box of a subsystem is already degenerate according to the values of ε^{Abs} and ε^{Rel}, it is skipped. Otherwise, the selected root-finding solver is started from the midpoint of the box by default. The available state-of-the-art root-finding solvers are constrained optimization methods such as scipy's wrapper to SLSQP, an SQP routine originally introduced by Kraft (1988) and casadi's wrapper to IPOPT, an interior-point filter line-search algorithm, which is introduced in Wächter & Biegler (2006). The objective function applied to solve the associated NLE as residual minimization problems is

$$\min_{\boldsymbol{x}} \frac{0.5 \cdot \sum_{i=1}^{n} f_i(\boldsymbol{x})^2}{\varepsilon^{fTol}} \quad . \tag{3.49}$$

Dividing the least-squares problem by ε^{fTol} appears to be sensible to force the solver to continue with the iteration, although the objective value is already below the required tolerance. Sometimes the minimizer does not yet fulfill the same tolerances in the corresponding root-finding problem and leads to strong deviations from the actual solution, especially in ill-conditioned systems. For this reason, it is always checked whether such a point is a solution to the root-finding problem for the required tolerance. If this is not the case, the Newton solver is started from the point and should converge quickly to the actual solution if that exists in its vicinity. In addition to the optimization algorithms, the state-of-the-art Fsolve method of scipy expects a well-determined $n \times n$ root-finding problem ($\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$), i.e., as many unknown variables as linear independent equations, and solves it internally as an unconstrained least-squares minimization problem of its first-order Taylor approximation via a modification of Powell's trust-region-dogleg algorithm (Powell, 1970). Hence, each iteration step $\boldsymbol{d}^{(k+1)}$ is a minimizer to

$$\begin{aligned} \min_{\boldsymbol{d}} \quad & \boldsymbol{J}(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d} + \frac{1}{2} \cdot \boldsymbol{d}^T \cdot \boldsymbol{H}(\boldsymbol{x}^{(k)}) \cdot \boldsymbol{d} \\ s.t. \quad & \|\boldsymbol{d}\| \leq \Delta \\ & \boldsymbol{d} = \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)} \quad . \end{aligned} \tag{3.50}$$

Δ is the trust region's radius that has been explained in section 2.3. The Jacobian matrix $\boldsymbol{J}$ is generally determined by Broyden's rank-1 update referring to Eq. 2.27, whenever progress is achieved. Otherwise and at the initial point, a forward

difference approximation is applied. The Hessian matrix $\boldsymbol{H}$ is approximated by

$$\boldsymbol{H} \approx \boldsymbol{J}^T \cdot \boldsymbol{J} \quad . \tag{3.51}$$

3.4 Cutting

Whenever an unsolved box is consistent according to the applied contraction methods, an attempt is started to identify empty edge boxes, which can be cut off. A small example was given in section 2.6.5 to illustrate the general principle of cutting. In this section the focus is put on the actual implementation. The method originates from Ebert (2021, pp. 46-50). The basic idea is to select one specific variable interval $\overline{\underline{x}}$, generate an edge interval $\overline{\underline{y}} \subset \overline{\underline{x}}$ by taking a certain step Δx into the box starting from the currently processed lower or upper bound, and then evaluating the current system $\boldsymbol{f}(\boldsymbol{x})$ in the related edge box $\overline{\underline{\boldsymbol{y}}}$. If $\boldsymbol{0} \notin \overline{\underline{\boldsymbol{f}}}(\overline{\underline{\boldsymbol{y}}})$ there is no real-valued solution for $\boldsymbol{f}(\boldsymbol{x}) = 0$ in the edge box, and it can be removed. In Ebert's original implementation all variables were successively cut for at most nine steps taken from lower and upper bound into the box. The step size was initially chosen to 1 % of the variable's interval width and in each step increased by another 1 % of the current width. While this method could indeed prevent splitting, its drawback is, the long extra computational time compared to the conventional approach consisting of contraction and bisection only. Therefore, the method has been further developed regarding two aspects:

1) Which variable intervals shall be cut?

2) How to select a more efficient step size for cutting?

Figure 3.6 shows the actual implementation of the method, called cut_box. Regarding the first aspect, the former algorithm has been generalized for the preselection of n_t certain cut variables. The function input requires a set of the cut variable's global indices $\{\boldsymbol{t}\} := \{t_1; \ldots;\ t_i; \ldots;\ t_{n_t}\}$ with i being the index of the variable in the preselected set. Previous computational experiments from Bublitz et al. (2017a) showed strong sensitivity of NLEs regarding the initial values of their tearing variables. In case they were chosen close to the solution, the initialization of all other variables was almost irrelevant for a successful numerical solution in a

system decomposed by BBTF. Hence, if only the tearing variables are preselected for cutting, the reduction of their intervals might be sufficient enough to tighten the solution space of a problem significantly and create further potential for other variable intervals to be contracted in the first step of the hybrid approach. The tearing variables are determined by the method BBTF introduced in section 2.4.2. The applied algorithm is the MC33 from the Harwell Subroutine Library (HSL, 2022). Should the user desire to only cut the tearing variables, the hyperparameter "cutBox" needs to be set to `tear`. However, when it is set to `all` the preselection encompasses all variables with the difference to former versions that the variables are sorted by their permutation index resulting from the DM decomposition.

The second aspect concerning the step size selection has been slightly modified compared the original implementation by Ebert (2021, pp. 46-47). Instead of increasing the step size Δx by 1 % in each cut and allowing a maximum of nine steps per bound, the step size increases quadratically by the formula

$$r_{step} := \frac{(\sqrt{r_{step} \cdot 100} + 1)^2}{100} \tag{3.52}$$

$$\Delta x := r_{step} \cdot w(\underline{\overline{x}}_{t_i}) \quad , \tag{3.53}$$

starting with a minimum relative step size r_{min} of 1 %. Initially, the low step size is preserved to exclude just those variable intervals from cutting that really have no empty edge boxes. The step size increases then up to a maximum of 100 %, which is equivalent to ten cutting steps with respect to Eq. 3.52 and 3.53. Thus, the complete remaining interval is checked for root inclusion in step 10 allowing for the potential identification of an empty box and terminating the process before further cutting is done. Especially for wide intervals this method appears advantageous, as they can be further tightened in fewer cutting steps.

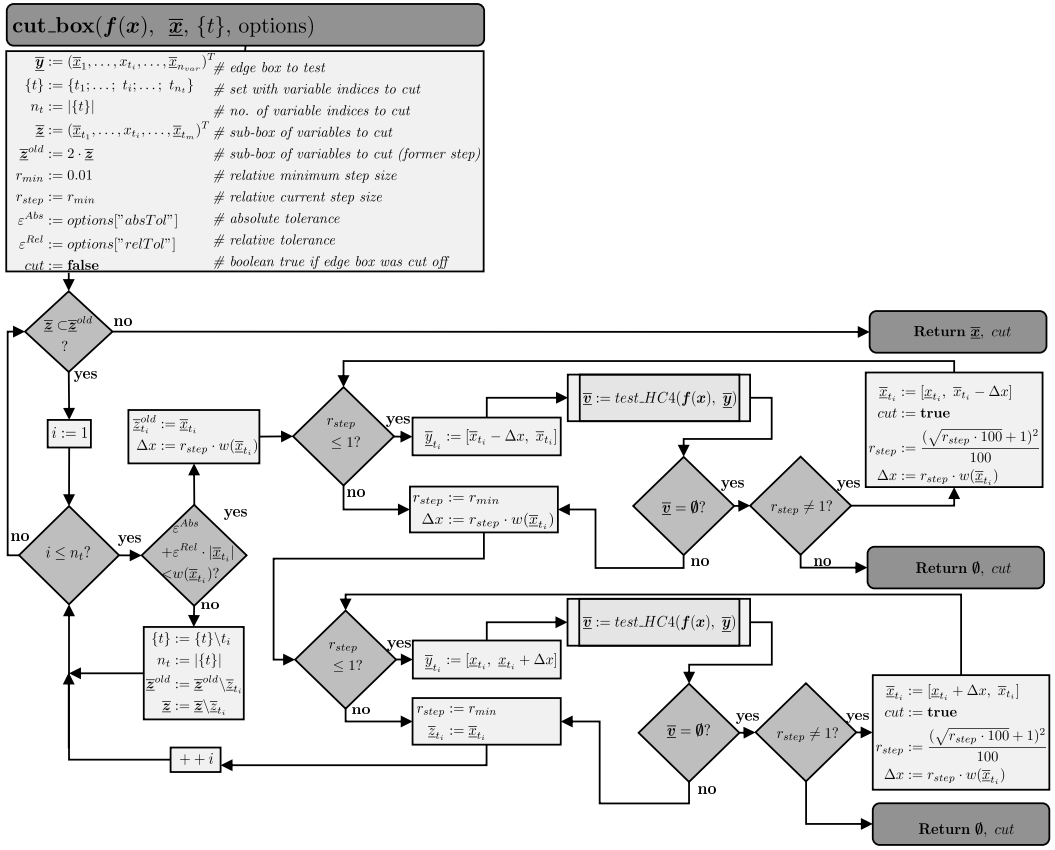


Fig. 3.6: The algorithm of cut_box.

After each individual bound reduction the step size is set back to r_{min}. Another advancement is that, after all cut variables have been reduced once, the cycle restarts to cut away further edge boxes based on the newly tightened box. So when the step size from the last cycle has been too large, some edge boxes might disappear in the new cycle given that it is again initiated with r_{min}. For root inclusion the algorithm test_HC4 is invoked, which applies three steps of the HC4revise method as described in Ebert (2021, pp. 46-47). The HC4revise method is quite fast and if it is applied multiple times, more empty edge boxes can be identified and removed. The cutting method cycles through all cut variables until consistency is reached exactly as in the contraction methods. Cut variables, which are already degenerate with respect to the set absolute tolerance ε^{Abs} and relative tolerance ε^{Rel}, are removed from the preselected set and therefore not cut. If all cut variables are degenerate or no edge boxes can be removed the variable *cut* retains its initial value `false` and is returned together with the unchanged box. If the variable *cut* is `false`, when the method returns, the next step for the currently processed box will be splitting following the hybrid approach.

3.5 Splitting

If a box is consistent to cutting and contraction, it is marked as eligible for splitting. Whenever the current number of processed boxes is lower than the maximum number of processed boxes permitted, an eligible box is split. The selection process, which box to split first, is explained in section 3.7. Concerning the question which variable to split, the techniques presented in section 2.6.4 are all heuristic. So far, there is no analytic solution to the problem which variable split leads to the best performance. The objective that has been focused on in this work is to:

Enable a maximum reduction of both boxes resulting from the split in the upcoming contraction.

Selecting only one specific variable for splitting, for example the one with the largest interval width, does not have to match this objective. Hence, the algorithm split_box has been developed to select not just one variable but rather a set of

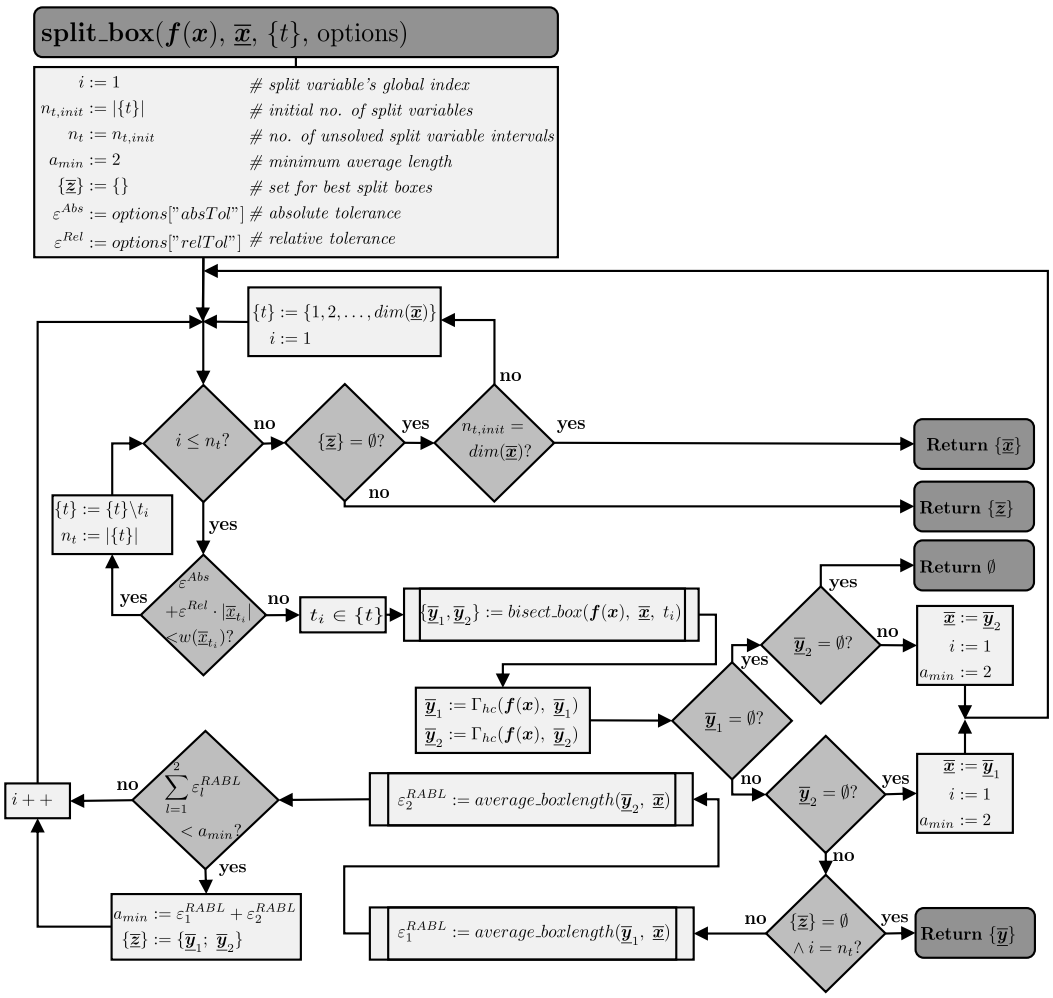


Fig. 3.7: The algorithm of split_box.

potential split variables and determine the best split out of those regarding the objective. Figure 3.7 introduces this algorithm. Before discussing the preselection of potential split variables, it shall be assumed for now this has already been accomplished: All preselection methods return a set with the global indices of the potential split variables denoted by $\{t\}$. Such a set is an input argument of split_box. Among these variables the algorithm skips all already solved variable intervals according to the absolute and relative tolerances ε^{Abs} and ε^{Rel} and continues with bisecting the box successively at each remaining potential split variable. To test the success of a split, a single contraction step by the HC4revise method

is applied on each of the resulting subboxes ($\Gamma_{hc}(\boldsymbol{f}(\boldsymbol{x}), \overline{\underline{\boldsymbol{y}}}_1)$ and $\Gamma_{hc}(\boldsymbol{f}(\boldsymbol{x}), \overline{\underline{\boldsymbol{y}}}_2)$). If this results in one box to be empty, the algorithm updates $\overline{\underline{\boldsymbol{x}}}$ to the non-empty subbox and restarts the procedure. If both boxes are empty, the algorithm simply returns an empty set. In case both boxes are non-empty, the algorithm continues trying to fulfill the objective. Therefore the success of a box split is measured by the relative average box length ε^{RABL} that is defined as

$$\varepsilon^{RABL}(\overline{\underline{\boldsymbol{y}}}, \overline{\underline{\boldsymbol{x}}}) := \frac{\sum_{i=1}^{n_{var}} \frac{w(\overline{\underline{y}}_i)}{w(\overline{\underline{x}}_i)}}{n_{var}} \quad , \tag{3.54}$$

for all non-degenerate variable intervals in $\overline{\underline{\boldsymbol{x}}}$ and with $\overline{\underline{\boldsymbol{x}}}$ being the box before and $\overline{\underline{\boldsymbol{y}}}$ being the box after the reduction. The average box length is calculated using the number of all n_{var} variables instead of the number of non-degenerate variable intervals only . If n_{var} were replaced by the latter in Eq.3.54, a just solved variable interval may cause an increase in the average box length due to the decreasing number in the denominator, although the total box volume drops. The average box length is determined in method average_boxlength that is part of the Appendix figure A.7a. The resulting average box lengths of both subboxes after splitting are summed up and compared with the current minimum value of this sum from former splits a_{min}. Initially, a_{min} is set to 2, because in worst case both subboxes can not be reduced at all by contraction, in which case

$$a = \varepsilon^{RABL}(\overline{\underline{\boldsymbol{y}}}_1, \overline{\underline{\boldsymbol{x}}}) + \varepsilon^{RABL}(\overline{\underline{\boldsymbol{y}}}_2, \overline{\underline{\boldsymbol{x}}}) = 2 \cdot \frac{n_{var} - 0.5}{n_{var}} \quad . \tag{3.55}$$

This will always be lower than 2 for a finite number of variables. Hence, even if all tested split variables can not reduce the original box at all, at least one of them is split and stored in set $\{\overline{\underline{\boldsymbol{z}}}\}$ that corresponds to the current best split. If all potential split variables have been tested, the set $\{\overline{\underline{\boldsymbol{z}}}\}$ is returned – if it is non-empty. An empty set $\{\overline{\underline{\boldsymbol{z}}}\}$ can occur if no potential split variable has actually been bisected, because all of them are already solved or the input argument $\{\boldsymbol{t}\}$ has been empty from the beginning on. In both cases, the set $\{\boldsymbol{t}\}$ is edited to contain all variable indices, i.e., all variables are potential split variables and the procedure is restarted. If the set $\{\overline{\underline{\boldsymbol{z}}}\}$ is still empty after testing all variables, this can only be related to all intervals being solved referring to an already solved box $\overline{\underline{\boldsymbol{x}}}$. Hence, a set with the solved box $\{\overline{\underline{\boldsymbol{x}}}\}$ is returned. Algorithm bisect_box is shown in figure

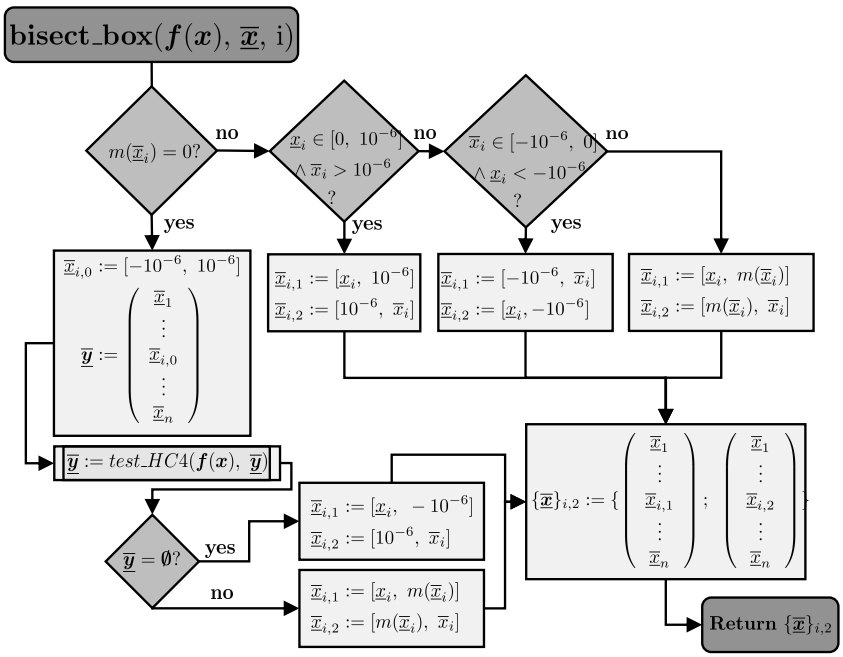


Fig. 3.8: The algorithm of bisect_box.

3.8. In general, it separates the box $\overline{\underline{x}}$ at the current split variable into two equally sized subboxes, except when the center of the interval to be split is just zero or the lower or upper interval bound is close to zero. In these three cases, the box is cut asymmetrically because it is assumed that variable values of zero are more often associated with singular or discontinuous points and these are even inherited to both subboxes when splitting at zero. If the midpoint is zero, a narrow subbox in the range $[-10^{-6}, 10^{-6}]$ is cut out and checked for containing a solution. If it does not, the two outer subboxes $[\underline{x},\ -10^{-6}]$ and $[10^{-6},\ \overline{x}]$ are returned. Otherwise, symmetric splitting is applied. If the lower or upper interval bound is zero or close to zero, the box is also split asymmetrically, into a small subbox containing the zero bound and a large subbox encompassing the remaining interval. The idea is that the small subbox can be discarded early in the following contraction steps. For preselecting potential split variables the following four methods are available:

- `forecastSplit`: All variables are selected
- `forecastTear`: All tearing variables are selected

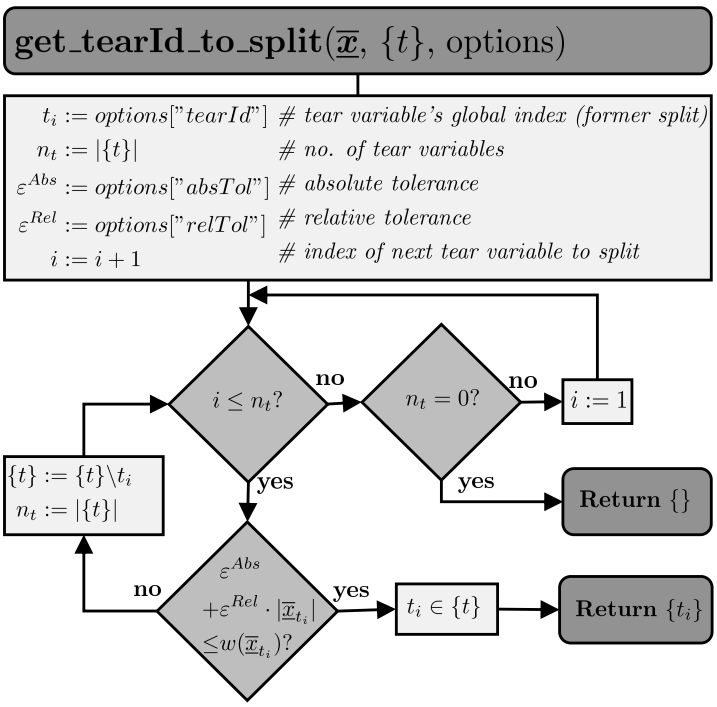


Fig. 3.9: get_tearId_to_split.

– `tearVar`: Only one tearing variable is selected per split, and it is alternated through all tearing variables

– `leastChanged`: One or multiple variables with the lowest relative change in their interval width(s) are selected

If option `forecastSplit` is chosen the method split_box is initialized with an empty set $\{t\}$. For `forecastTear` the set $\{t\}$ contains the global indices of the tearing variables from the BBTF. The difference to option `tearVar` is that the latter bisects just one tearing variable in a split and selects the subsequent tearing variable in the next split, instead of finding the best split among them. Hence, this method is supposed to be faster than the other two for one split. The order of splitting is then given by an increasing global index. If a box has been split at the variable interval with the highest global index, it is restarted with the lowest index in the next split. Tearing variables with solved intervals are skipped from this alternation. The algorithm is shown in figure 3.9. In option `leastChanged` the relative change of each variable's interval width $\overline{\underline{x}}_i$ to its respective interval

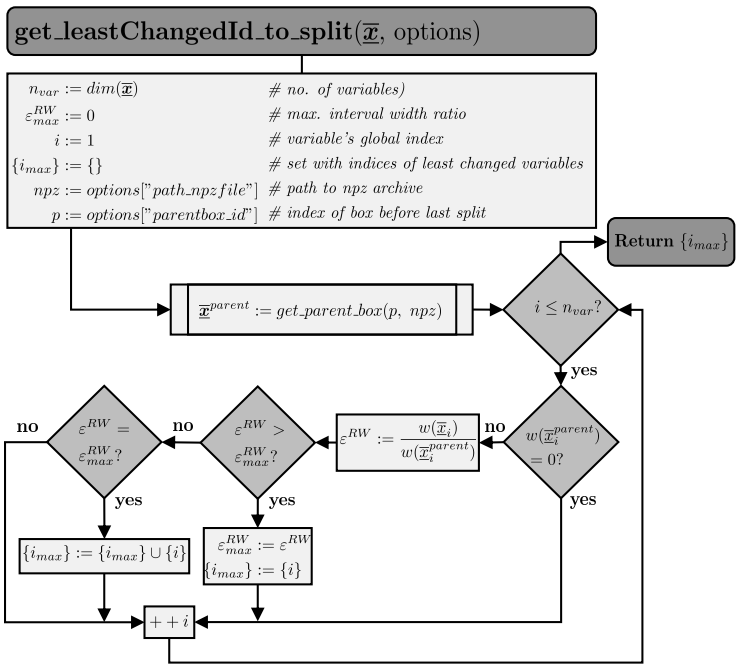


Fig. 3.10: get_leastChangedId_to_split.

width before the last split $\overline{\underline{x}}_i^{parent}$ is calculated by

$$\varepsilon^{RW}(\overline{\underline{x}}_i, \overline{\underline{x}}_i^{parent}) := \frac{w(\overline{\underline{x}}_i)}{w(\overline{\underline{x}}_i^{parent})} \quad . \tag{3.56}$$

The global index of the variable that changed the least, i.e., has the maximum value for ε^{RW}, is returned in a set $\{i_{max}\}$. Especially in the first splits, multiple variable intervals might have not changed at all, which is equivalent to $\varepsilon^{RW} = 1$. For this case, all corresponding indices are stored in $\{i_{max}\}$ and returned. The method behind this option is called get_leastChangedId_to_split and presented in figure 3.10. The box $\overline{\underline{x}}^{parent}$ is the so-called parent box of the current box $\overline{\underline{x}}$ that is supposed to be split. While a parent box is bisected, it produces two child boxes. The origin of each child box is stored in a tuple p with this following information

$$p := (\text{reduction step, index of box}) \quad . \tag{3.57}$$

More about the storage of boxes is described in Appendix A.3.2.

3.6 Processing One Box

Now all steps of the hybrid approach according to figure 3.1 have been explained. It remains to discuss, how the actual reduction step of one box is performed. It depends on the current state of the box, which is described by multiple box properties that have partly been introduced in section 3.1. The relevant ones for the reduction step selection and their abbreviations are:

- k := index of current reduction step
- l := box index of current reduction step
- r := boolean, `true` if box is ready for a split
- s := boolean, `true` if box is solved
- d := boolean, `true` if box is discontinuous
- c := boolean, `true` if box is consistent to contraction
- p := tuple with (reduction step, box index) of parent box
- cb := boolean, `true` if box is consistent to cutting

The method reduce_box, shown in figure 3.11, determines the next step for a box based on its property values.

A solved box keeps its properties and is simply returned. Given that in other steps, such as splitting, multiple boxes can occur, all properties and boxes are generally returned in a set, even if they did not change. A consistent and continuous box is first attempted to be cut in method cut_box, if it is still applicable for cutting. This is the case when its property cb has not been turned to `false` after reaching consistency in the former reduction steps. After cutting, the box is checked for being successfully reduced, which is equivalent to a `true` cb value. In this case, the box is further checked for being solved and returned.

A continuous and consistent box to both contraction and cutting is then checked by the value of the parameter r for its permission to be split. The latter is initially `true`, if the capacity of currently processed boxes allows an increasing number

of boxes. Nevertheless, r can still be set to `false` even if there is enough capacity present when priority is given to another box to be split first. How and why the prioritization is important is explained in section 3.7. If a box is not eligible for splitting, it is just returned with unchanged properties. Through the discarding process of empty boxes, parameter r might become `true` later. If the box is ready for splitting, it follows the preselection of split variables and the actual split in split_box. After the split, the set of split boxes, that can consist of one or two boxes, is checked for root inclusion and if it matches the tolerances for being solved, in which case property s changes. A set of non-empty boxes and their updated properties or an empty set is then returned by the method. For a box that is consistent but discontinuous, it is also checked if it is eligible for splitting. However, if r is `true` a discontinuity from the domain will be used for the split, which has been previously labeled in one of the three contraction methods as described in section 3.1. If a consistent and discontinuous box is not ready for a split, it will be returned with unchanged properties. Finally, a box that is not consistent traces the same path as a consistent and discontinuous box.

As long as a box is not consistent, r and cb are always `true`. Hence, it is continued with reducing an inconsistent box by contract_box. The method contract_box contains the two checks for a box being solved or consistent, according to figure 3.5. It also sets d to `false`, if a box has been split by its one and only discontinuity during the contraction. If a split takes place in contraction, the origin of the resulting boxes is updated in $\{p\}$. Hence, contract_box returns the set of the reduced box or reduced boxes $\{\overline{\underline{\boldsymbol{y}}}\}$, and the corresponding sets of updated properties $\{s\}$, $\{d\}$, $\{c\}$ and $\{p\}$. The cb value is then set `true` for all generated boxes. In all unsolved boxes in $\{\overline{\underline{\boldsymbol{y}}}\}$, the root-finding algorithm is started. Solutions determined by the root-finding algorithm are stored in num and later updated to the dictionary $options$ to be always available in the upcoming reduction steps for the unique solution test in contract_box. If root-finding is deselected by setting the hyper parameter "hybridApproach" from $options$ to `false` a pure IA-based generalized bisecting algorithm is applied.

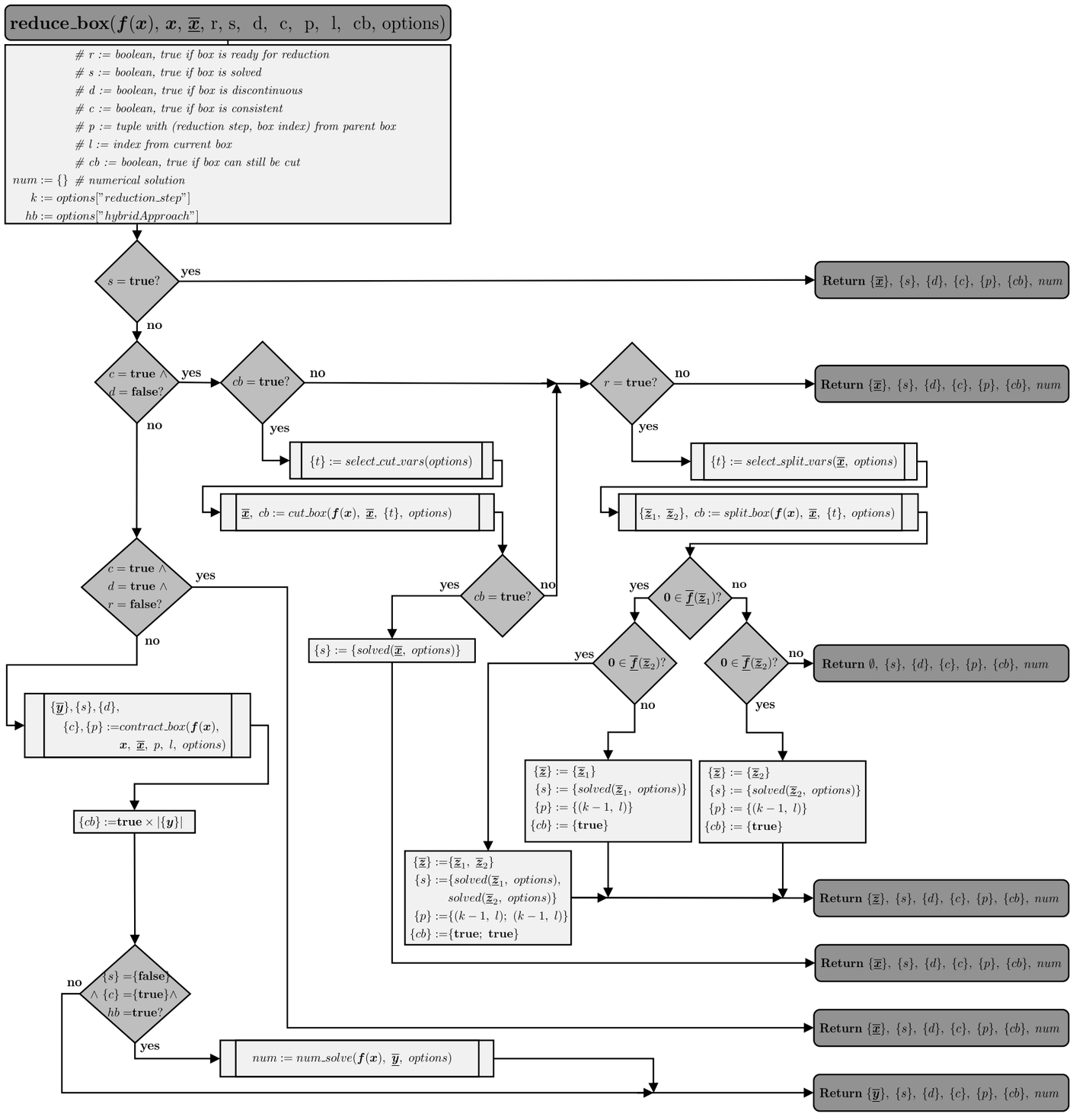


Fig. 3.11: The algorithm of reduce_box.

3.7 Processing Multiple Boxes

The upmost implementation layer of the hybrid approach consists of the algorithm solve_NLE presented in figure 3.12. It requires the functions $\boldsymbol{f}(\boldsymbol{x})$ related to the problem $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, the symbolic variables $\boldsymbol{x}$, the initial box $\overline{\underline{\boldsymbol{x}}}^{(0)}$ corresponding to the user-specified variable bounds and the dictionary *options* that holds all parameters the user can set. The algorithm tries to reduce all boxes currently processed. One pass over all boxes equals one reduction step. To prevent the risk of long-running processes, the number of reduction steps is restricted by the parameter n_{mstep} specified by the user. Beside this, the current number of processed boxes n_{box} can never exceed the maximum number of boxes n_{mbox}. Initially, n_{mbox} equals one and is increased by one whenever all of the processed boxes become consistent to contraction and cutting, i.e., a split is unavoidable for progress. The properties to describe the state of a box have been explained in the previous section. In solve_NLE the individual properties of the boxes are stored in sets to be used in the next reduction step. The order is important here, and each property's position in the set is equivalent to the one of its corresponding box in $\{\overline{\underline{\boldsymbol{x}}}\}^{new}$, which is the set of all boxes resulting from the reduction step. The reduction step starts when the initial box is non-empty, unsolved in the given tolerances, and n_{mstep} is greater than zero. The parameter n_{mbox} will not be increased before the first reduction step, because the initial box is assumed to be inconsistent and its properties values *c* and *cb* are `false`. In a new reduction step the dictionary *options* is updated for the current number of boxes n_{box}, the maximum number of boxes n_{mbox}, the current reduction step k, and new solutions determined by the root-finding algorithm in the last reduction step $\{num\}$. These properties are relevant to all box reductions in reduce_boxes for restricting the number of box splits and checking the unique solution test. Algorithm update_options can be found in figure A.7b in the Appendix. If more than one box shall be reduced, the order of boxes is important as first boxes are split first.

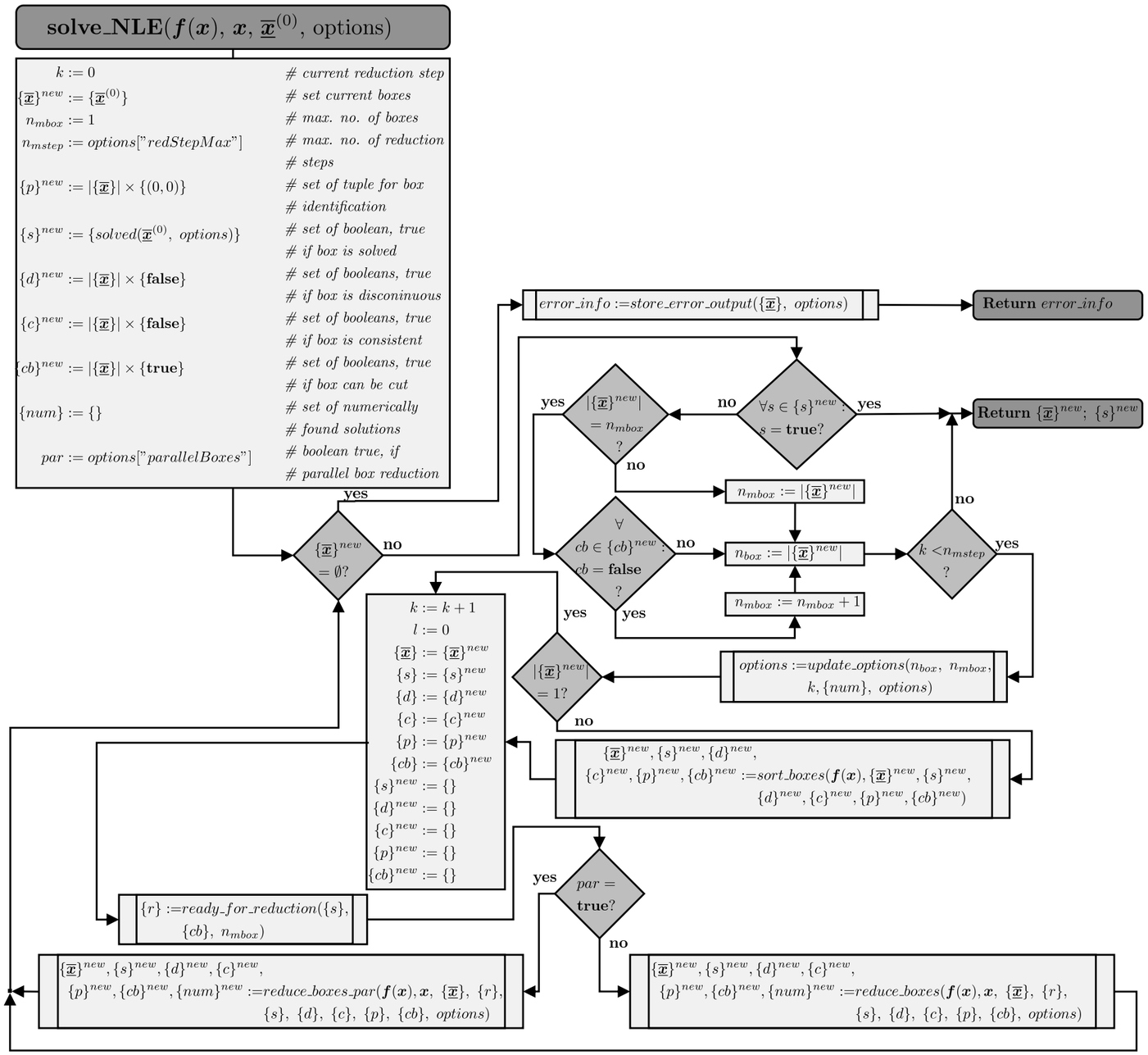


Fig. 3.12: The algorithm of solve_NLE.

Important for this hybrid approach is to quickly find any solution of the system rather than finding all system's solutions or to proof the nonexistence of other solutions in the entire initial box. Especially for large dimensional systems with enormous initial boxes, the identification of all solutions can become computationally intractable. In consequence, it is more efficient to focus on only those boxes that seem to be most promising for containing a solution. As a measure all functions f_j from $\boldsymbol{f}(\boldsymbol{x})$ are evaluated at each box's midpoint. The absolute values of these function residuals are summed up

$$fres_l = \sum_{j=1}^{n_{func}} |f_j(m(\overline{\underline{\boldsymbol{x}}}_l))| \quad , \tag{3.58}$$

to obtain the overall residual $fres_l$ of a box $\overline{\underline{\boldsymbol{x}}}_l$. This is performed for all boxes in $\{\overline{\underline{\boldsymbol{x}}}\}$ to sort them as well as their properties in increasing residual order. Hence, boxes with a low residual are split first. The detailed algorithm sort_boxes is presented in figure A.8 in the Appendix. To continue in solve_NLE from figure 3.12 the sorted boxes and properties, which have been obtained from the last reduction step, are still all labeled by $\{\}^{new}$. They are now stored in the corresponding sets without this superscript. The sets $\{\}^{new}$ are then emptied for the upcoming reduction step. Based on the boxes' order, it is determined which box is eligible for a split in reduce_boxes by the method ready_for_reduction. The latter returns a set $\{r\}$ of boolean values in sorted order, which are `true` for boxes that can be processed and split. Figure 3.13 depicts the algorithm ready_for_reduction. Initially the boolean values in $\{r\}$ are all `false`. For solved boxes they remain `false` and for inconsistent boxes they are turned to `true`. The important bit are the consistent boxes. Referring to the current capacity $c_{box} = n_{mbox} - n_{box}$ only the first c_{box} consistent boxes are allowed to be split. In the general implementation one might recognize that after each reduction step the maximum number of boxes n_{mbox} is set to the current number of boxes $|\{\overline{\underline{\boldsymbol{x}}}\}^{new}|$, or it is at most increased by one if $|\{\overline{\underline{\boldsymbol{x}}}\}^{new}| = n_{mbox}$ and all boxes are consistent. Hence, in this approach c_{box} should either be zero or one and there is at most one consistent box split in a reduction step. It is waited for all boxes to reach consistency, because inconsistent boxes might be contracted to a very low residual level and are then the first ones to be split. Otherwise splitting a box whenever it becomes consistent could produce many boxes with lower residuals before the still contracted boxes become

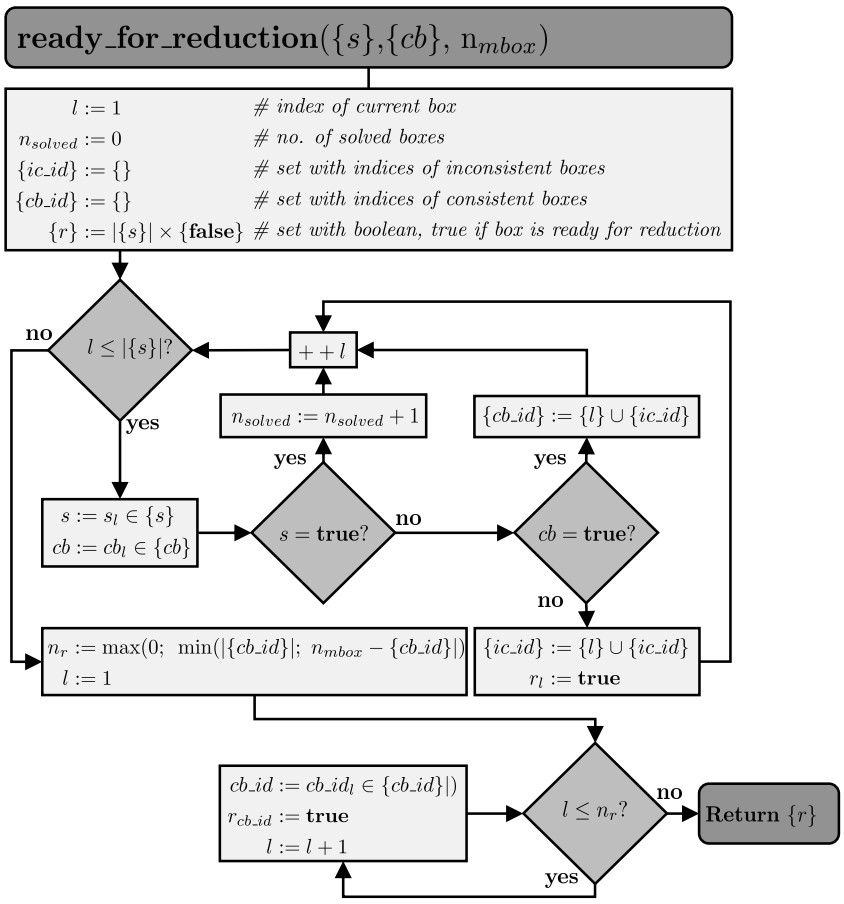


Fig. 3.13: The algorithm of ready_for_reduction.

consistent. Hence, less promising boxes would come first in the box order and would be extensively split but could not be easily discarded because of interval dependency. By the time the promising boxes were consistent, there would not be any capacity for them to be split, before the extensively split boxes disappear, which could take a while.

Next, the set of boxes $\{\overline{\underline{\boldsymbol{x}}}\}$ is pruned in reduce_boxes or reduce_boxes_par, which both return a set of reduced boxes $\{\overline{\underline{\boldsymbol{x}}}\}^{new}$ and their corresponding properties. Whether the first or second method is called depends on the parameter *par* that is `true`, if parallel processing has been selected by the user and is applied for the box reduction. The implementation of reduce_boxes_par will be topic of section 3.8. In the algorithm reduce_boxes presented in figure 3.14 all boxes $\overline{\underline{\boldsymbol{x}}}_l$ are successively

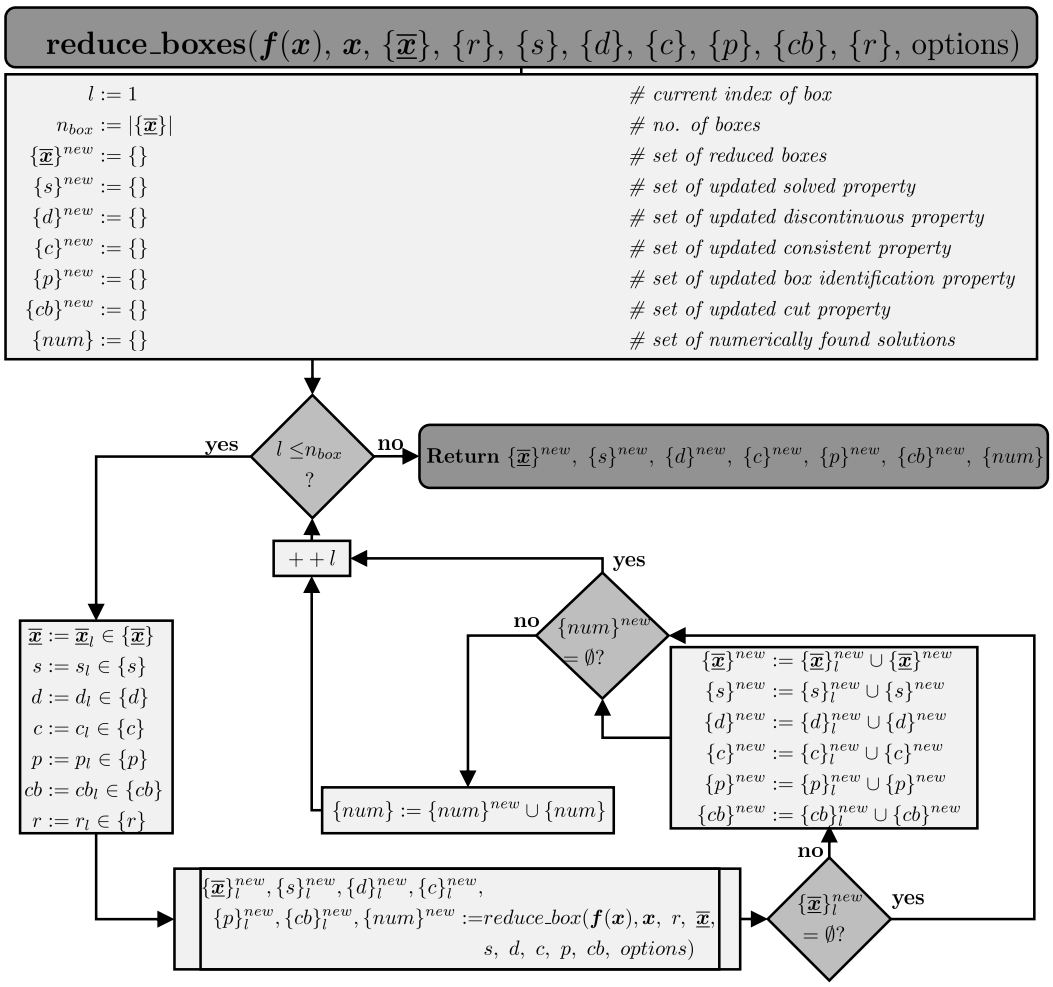


Fig. 3.14: The algorithm of reduce_boxes.

tightened. If the resulting set of reduced boxes $\{\overline{x}\}_l^{new}$ is empty, it is continued with the reduction of the next box. Otherwise, $\{\overline{x}\}_l^{new}$ is unified with the set of already reduced boxes $\{\overline{x}\}^{new}$ from the previous box reductions. The same is done for the property sets. If all boxes are empty then the set $\{\overline{x}\}^{new}$ and the property sets are returned as empty sets to the method solve_NLE. Obviously, the system has no solution in the initial box in this case. For error analysis, the reason for a box to be identified as empty is stored in *options*. The reason statement always encompasses one equation and one empty variable interval. The global ID of this equation and the final intervals of all variables the equation consists of at the time when one interval becomes empty are stored. On top of that, the

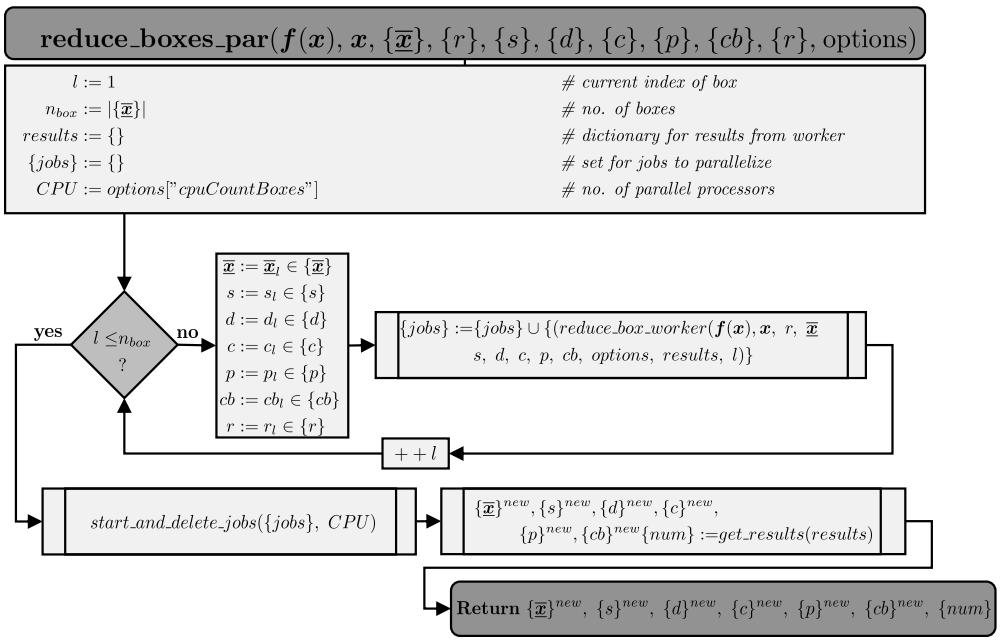


Fig. 3.15: The algorithm of reduce_boxes_par.

ID of the empty box resulting from $\{\overline{\boldsymbol{x}}\}$ is kept to also return the last non-empty bounds before the reduction. This error analysis is not always returned to the user, because one of the algorithm's major tasks is to identify and discard empty boxes. Only when the complete initial box becomes empty, the information about the last processed, empty box are assigned to the dictionary $error\_info$ by the method store_error_output (see figure A.9a in the Appendix). All solutions determined by root-finding are stored in $\{num\}$. After having reduced all boxes from $\{\overline{\boldsymbol{x}}\}$, the method restarts from where it has begun with checking the termination criteria whether all boxes are empty or solved, and finally increasing the reduction step, if n_{mstep} has not been reached yet.

3.8 Parallelization

To decrease the CPU time of the whole procedure, the reduction of multiple boxes has been parallelized. Instead of invoking method reduce_box successively in the algorithm reduce_boxes, the method reduce_boxes_par starts it for multiple boxes

at the same time. The latter is shown in figure 3.15. How many boxes may be reduced in parallel depends on the number of cores physically available on the used computer. By the parameter "cpuCountBoxes" the user may specify how many cores are used in parallel. The user should refrain from choosing "cpuCountBoxes" higher than the number of physically available cores as this may slow down the program enormously as the task scheduler of the ***O**perating **S**ystem* (OS) might start "cpuCountBoxes" processes simultaneously and will always switch between them to progress them equally (Dutton et al., 2008). The python package multiprocessing developed by McKerns et al. (2012) is used for parallel computing. The box reduction processes are completely independent from one another so that no information needs to be shared while they are running. To allow that, the method ready_for_reduction from figure 3.13 is inevitable. It determines the number of boxes to be split before the actual reduction takes place. In consequence, the maximum number of boxes is never exceeded by the procedure. For further information on how to parallelize the box reduction using algorithms reduce_box_worker and get_results, see Appendix A.3.3. The entire approach offers even more opportunities for process parallelization that is referred to in section 6.9.

3.9 Implementation of the Hybrid Approach

The hybrid approach has been implemented as a python package named modOpt that can be downloaded from the git repository[1]. Instructions are given in the git-project how the package and its dependent packages and libraries should be installed. The modOpt package is subdivided into five sub packages:

- constraints
- decomposition
- initialization
- scaling

[1] https://git.tu-berlin.de/dbta/simulation/modOpt

- solver

- tests

to modularise independent functionalities. Doing so, one can for example use the IA-based reduction that only relies on the sub packages constraints and decomposition without root-finding, i.e., skip step three of the hybrid approach. To enable this separation, the *options* dictionary shown in the algorithms of the previous sections consists actually of three dictionaries namely *bxrd_options*, *smpl_options* and *num_options*. If no root-finding is required, the latter two dictionaries do not even have to be defined in the python script. Similar to this, sampling is omitted, if the dictionary *smpl_options* is missing, and the midpoint of the box is used for root-finding, which is the default setting applied in this thesis. The python script can be automatically generated for any NLE from MOSAICmodeling by applying the ***U**ser **D**efined **L**anguage **S**pecificator* (UDLS) modOpt_constraints_V2.9 (ID: 167855). How to use and create a UDLS in MOSAICmodeling is explained in Tolksdorf et al. (2019). The sub package *tests* contains scripts, in which the hybrid approach is applied on some chemical process examples to check if the algorithm works as expected. The scripts also serve as templates for anybody who wants to apply the hybrid approach on their own NLEs. In modOpt the python package mpmath is used for the IA operations. Some of mpmath's functions with limited domains in $\mathbb{R}$ such as $\ln(x)$, $\sqrt{x}$, $x^{\frac{y}{z}}$, $\tan(x)$, $\arccos(x)$, $\arcsin(x)$ have been modified to discard complex ranges, which are out of interest in the field of real applications. How this has been implemented is explained in Ebert (2021, pp. 55-64).

3.10 Analysis Parameters

The success of the hybrid approach is analyzed at each box reduction step k by the following parameters:

- ***R**elative **A**verage **D**omain **L**ength* (RADL) ($\varepsilon^{RADL,(k)}$)

- Number of boxes ($n_{box}^{(k)}$)

– Successful root-finding iterations during the box reduction .

Parameter $\varepsilon^{RADL,(k)}$ originates from the ***R**elative **A**verage **B**ox **L**ength* (RABL) ε^{RABL}, which has been previously introduced in section 3.5. The notation originates from the following definitions: The ***B**ox **L**ength* (BL) ε^{BL} is the sum of the n_{var} variable interval widths of a box $\overline{\underline{\boldsymbol{x}}}$

$$\varepsilon^{BL}(\overline{\underline{\boldsymbol{x}}}) := \sum_{i=1}^{n_{var}} w(\overline{\underline{x}}_i) \quad . \tag{3.59}$$

The ***R**elative **B**ox **L**ength* (RBL) ε^{RBL} sums up the relative change of the interval widths from a certain box $\overline{\underline{\boldsymbol{x}}}$ to another box, for example the initial box $\overline{\underline{\boldsymbol{x}}}^{(0)}$

$$\varepsilon^{RBL}(\overline{\underline{\boldsymbol{x}}}, \overline{\underline{\boldsymbol{x}}}^{(0)}) := \sum_{i=1}^{n_{var}} \frac{w(\overline{\underline{x}}_i)}{w^{(0)}(\overline{\underline{x}}_i)} \quad , \tag{3.60}$$

for all n_{var} variables from $\overline{\underline{\boldsymbol{x}}}^{(0)}$ that are not solved. ε^{RABL} is then the averaged value of ε^{RBL} with respect to the n_{var} variables

$$\varepsilon^{RABL}(\overline{\underline{\boldsymbol{x}}}, \overline{\underline{\boldsymbol{x}}}^{(0)}) := \frac{\varepsilon^{RBL}(\overline{\underline{\boldsymbol{x}}}, \overline{\underline{\boldsymbol{x}}}^{(0)})}{n_{var}} \quad . \tag{3.61}$$

Finally, if multiple boxes $\{\overline{\underline{\boldsymbol{x}}}\}^{(k)}$ are processed in a reduction step k, the ***R**elative **A**verage **D**omain **L**ength* (RADL) ε^{RADL} is defined as

$$\varepsilon^{RADL}(\{\overline{\underline{\boldsymbol{x}}}\}^{(k)}, \overline{\underline{\boldsymbol{x}}}^{(0)}) := \frac{\sum_{l=1}^{n_{box}^{(k)}} \varepsilon^{RABL}(\overline{\underline{\boldsymbol{x}}}_l^{(k)}, \overline{\underline{\boldsymbol{x}}}^{(0)})}{n_{box}^{(k)}} \quad , \tag{3.62}$$

which determines the averaged ε^{RABL} value with respect to all $n_{box}^{(k)}$ boxes $\overline{\underline{\boldsymbol{x}}}_l^{(k)} \in \{\overline{\underline{\boldsymbol{x}}}\}^{(k)}$. Whenever $\varepsilon^{RADL}(\{\overline{\underline{\boldsymbol{x}}}\}^{(k)}, \overline{\underline{\boldsymbol{x}}}^{(0)})$ is shown or discussed in the upcoming sections, it is simply denoted as $\varepsilon^{RADL,(k)}$, i.e., the value of ε^{RADL} in the k-th reduction step references to the initial box so that $\varepsilon^{RADL,(0)} = 1$. If all solutions within $\overline{\underline{\boldsymbol{x}}}^{(0)}$ have been found, ε^{RADL} is zero. Only those n_{var} variables are counted, which initially have non-degenerate intervals, i.e., have not yet been solved. Initial, degenerate variable intervals are equivalent to declaring them as constant parameters. Alternatively to ε^{RADL}, the first idea has been to define a so-called ***R**elative **H**ypercubic **L**ength* (RHL) parameter ε^{RHL} that relates the sum of all reduced box

volumes from reduction step k to the initial box volume and takes the n_{var}-th root of this value

$$\varepsilon^{RHL,(k)} := \sqrt[n_{var}]{\sum_{l=1}^{n_{box}} \prod_{i=1}^{n_{var}} \frac{w(\underline{x}_{l,i}^{(k)})}{w(\underline{x}_{i}^{(0)})}} \quad . \tag{3.63}$$

Doing so, one assumes the initial box and the sum of reduced boxes to be both hypercubic, in which case the n_{var}-th root equals their edge length. Solved intervals are excluded from the product as the actual problem then only reduces by one dimension and the remaining volume is non-zero. However, this quantity seems to be inappropriate for large systems. Assume all 400 initial variable intervals of a certain system are reduced down to 10 % of their width the computer has to calculate

$$\varepsilon^{RHL} = \sqrt[400]{0.1^{400}} \quad , \tag{3.64}$$

with the radicand being non-representable as a floating point number in python's precision (the lowest computable value is $\approx 2.23 \cdot 10^{-308}$). Hence the calculated value becomes zero, although the reduced boxes can be far off from being solved. The number of boxes $n_{box}^{(k)}$ is expected to have a great influence on the CPU time, while the relative, averaged number of variables $n_{var,solved}^{RA,(k)}$ gives an idea on how many variables in a box have already been solved at reduction step k. The last parameter used to validate the hybrid approach's performance is the success of the iteration by a root-finding method in between the box reduction. If only one feasible solution is of interest the hybrid approach can also stop as soon as this has been found and act in this way as local solver. All of the three parameters only validate the effectiveness of the hybrid approach and are by their definition independent of the model's size or structure. However, in the scope of this thesis the model properties shall also be investigated by certain parameters that are introduced in the next chapter.

4 Test Cases

In this chapter, the process engineering examples are presented, on which the hybrid approach is tested. Only NLEs are considered that contain complex subsystems after a DM decomposition (see section 2.4.2). The definition of complex systems as well as parameters applied to quantify the complexity of each example are presented in section 4.1. Sections 4.2 to 4.5 contain descriptions of each test case with a focus set on the formulation of individual equations that have a strong influence on the performance of the IA-based box reduction.

4.1 Characterization of Complex Systems

Complex systems are defined here in such a way that they contain at least one nonlinear equation and consist of at least two equations, which cannot be solved independently of each other. Purely linear, nonsingular systems are usually not a problem thanks to contemporary solution algorithms such as LU decomposition or Gauss elimination (Dahmen and Reusken, 2008, pp. 68-82). In chemical process models, complex systems frequently occur to describe phase equilibria, reactive systems or process units and flowsheets with recycle streams. The test cases contain at least one complex subsystem of this kind. The nonlinearity of complex systems poses a particular problem for conventional numerical solvers, since the system can have more than one solution or non-smooth function curves as discussed in section 1.1. In addition, different scales of equations and unknowns lead to ill-conditioned systems and eventually to an early termination or slow progress of the solver. This is exactly where the hybrid approach is supposed to help. However, problems can also vary in complexity. To the best of the author's knowledge no measure has been defined yet for the complexity of an NLE. Therefore, four parameters are now introduced to enable a gradation of

complexity: The dimension of the complex system n_{ls}, the condition number κ_2 of the Jacobian matrix evaluated at its solution(s) (it was defined in section 2.4), the non-zero density of the Jacobian matrix ρ_{nz} and the nonlinearity ratio ε_{nl}. The non-zero density is a measure for how strongly the equations are coupled with each other. For a $n \times n$ system with n_{nz} non-zero entries in its Jacobian matrix, ρ_{nz} is defined as

$$\rho_{nz} := \frac{n_{nz}}{n^2} \quad . \tag{4.1}$$

The nonlinearity ratio indicates how many of the non-zero entries are nonlinear by calculating the equations' second derivatives with respect to the variables and counting the structurally non-zero elements n_{nl}

$$\varepsilon_{nl} := \frac{n_{nl}}{n_{nz}} \quad . \tag{4.2}$$

Since the Newton-based algorithms work with linear approximations, this quantity provides a measure for the deviation of the approximation from the actual model. If $\varepsilon_{nl} = 0$, the Newton method should be able to solve the linear subsystem in one step. Finally, the higher the values of these parameters, the higher the system's complexity.

The hybrid approach is applied on DM-decomposed NLEs. The dimension of the whole system seems to be less important for solving it than the dimension of its individual subsystems. For example, any number of linear equations, each with one, linearly independent unknown, could be assembled into a system of equations. The system would certainly be easy to solve, although it is very high dimensional. For this reason, the parameters are only determined for the largest, complex subsystem of each example. Table 4.1 shows their values. For the *CSTR* and the *Flash Unit* more than one solution exist so that each row corresponds to one of them, i.e., κ_2 is different. The four test cases have been selected in such a way that they differ greatly in their parameter values and thus, represent a broad variety of complexity. The aim is to check whether the hybrid approach is able to solve them all and how it performs in comparison to conventional numerical methods. A total of eight test cases have been examined within the scope of this work, but the other four do not contribute any additional findings to the conclusion. However, since they support the results, they are included in the analysis of the results in chapter 6, and their models are part of Appendix B.

Tab. 4.1: Structural properties of largest subsystems of tested NLEs at the known solutions: n_{ls} is the dimension, κ_2 is the condition number of the Jacobian matrix evaluated at the solution, ρ_{nz} is the non-zero density of the Jacobian matrix and ε_{nl} is the nonlinearity ratio.

NLE	n_{ls}	κ_2	ρ_{nz}	ε_{nl}
CSTR	3	7.0×10^5	0.667	0.500
	3	1.2×10^3	0.667	0.500
	3	1.5×10^4	0.667	0.500
Flash Unit	8	3.5×10^2	0.281	0.111
	8	2.1×10^2	0.281	0.111
Total Condenser	12	4.8×10^6	0.194	0.357
Heavies Column	158	3.4×10^{10}	0.036	0.560

4.2 CSTR

In the model of the steady-state ***C**ontinuous **S**tirred-**T**ank **R**eactor* (CSTR) adopted from Liu (2017), the catalytic hydrogenation of aromatics (A) in an oil takes place by addition of hydrogen

$$\mathrm{A} + \mathrm{H_2} \longrightarrow \text{Products} \quad .$$

The reaction is exothermic. The CSTR is well-mixed, and operates adiabatically at constant pressure. The entire mathematical model including the reaction rate and properties of the aromatics, catalyst and hydrogen can be found in Appendix B.2. The largest complex subsystem encompasses the energy balance, the component balance of (A) and the exponential reaction rate. No interval dependency occurs so that no special formulations are required. Figure 4.1 represents the mathematically possible conversions of A in the CSTR at different inlet temperatures of the oil and aromatics containing feed. Two steady-state solutions exist in the temperature

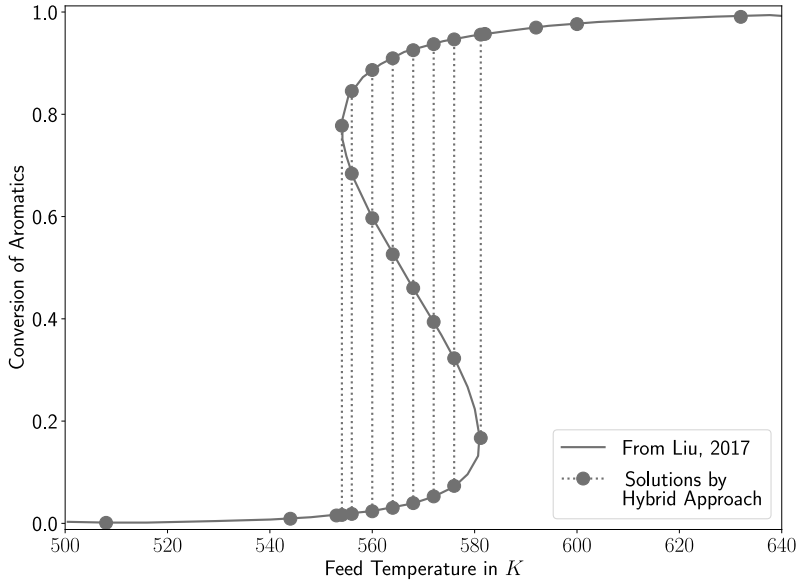


Fig. 4.1: Conversion of aromatics in CSTR for different Feed Temperatures analytically solved referring to Liu (2017) and by hybrid approach.

range from 554.10 K to 581.12 K. One at low conversion when the feed temperature was previously low and is increased up to a temperature lower or equal to 581.12 K, which is the so-called ignition point. The other is related to a high conversion when the feed temperature was previously high and is then cooled down to a temperature equal or greater than 554.10 K, the so-called extinction point. A third unstable and thus unsteady solution exists between extinction and ignition temperature that has an intermediate conversion compared to the ones belonging to the steady states. This example was implemented to ensure that the hybrid approach can exactly find one, two, or three solutions depending on the set feed temperature. In figure 4.1 the solutions detected by the hybrid approach are marked. It finds all solutions according to the feed temperature, in particular only the two solutions at the ignition and extinction point. Which solver options to select for an efficient iteration process will be investigated in the next chapter. Only the test case at a feed temperature of 565.0 K will then be examined.

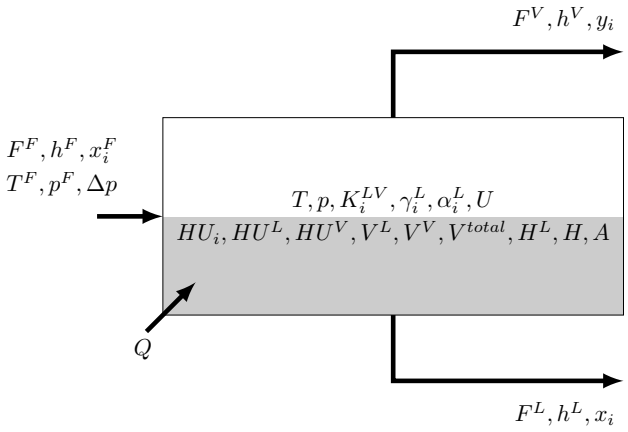


Fig. 4.2: Sketch of the Flash Unit model.

4.3 Flash Unit

In the flash unit, a liquid mixture of ethanol and water is partially evaporated by pressure reduction. The temperature in the unit is kept constant by supplying heat. The two phases within the unit are assumed to be well-mixed and in thermodynamic equilibrium. The real behavior of the liquid phase is described by activity coefficients according to Wilson's model. The associated equations are taken from the monograph of Gmehling & Kolbe (1992, p. 240-241). Ideal gas behavior is assumed for the gas phase. Feed conditions as well as pressure drop and temperature within the flash unit are set, while all quantities associated with the exiting streams and the heat demand are unknown. The design values, parameters, and model equations can be found in Appendix B.3. The largest complex subsystem contains eight equations: A summation relation for each phase, as well as three equations per component to describe the chemical equilibrium, the general equilibrium constant and the activity coefficient. The latter is determined by an exponential expression depending on the unknown molar composition, which occurs several times in the equation and causes interval dependency. Moreover, in this model and in the following ones, functions are used to determine thermodynamic properties of pure components such as molar enthalpies, molar volumes, and vapor pressures, as well as model parameters to describe the chemical equilibrium. This means that these variables are explicitly represented by their functional expression and do not appear as iteration vari-

ables in the equation system. The advantage is that the dimension of the NLE is reduced, i.e., the initialization of the functional variables is eliminated. The disadvantage is that the equations become very long and contain many nonlinear expressions including interval dependency. However, reformulations are not necessary here, because the hybrid approach can still solve the system. The NLE has two solutions in the initial box $\overline{\underline{x}} = [-10^9, 10^9]^{28}$, one of which is physically plausible. The other contains mole fractions outside the interval $[0, 1]$ as well as negative flow rates and is therefore purely mathematical.

4.4 Total Condenser

A superheated ethanol (1)and water (2) stream at known composition ($y_{i=1}^{h,sh,in} = 0.412$, $y_{i=2}^{h,sh,in} = 0.588$) is condensed and subcooled at atmospheric pressure in this example. The vapor stream enters the counter-current heat exchanger at 80 °C. A shell and tube heat exchanger is used for this purpose, in which water is deployed as coolant with an inlet temperature of 25 °C at a pressure of 1 bar. The coolant flows inside the pipes. The heat exchanger model consists of three sections sketched in figure 4.3 referring to the shell-side fluid state, namely the superheated (sh), the two-phase (2ph) and the subcooled (sc) section. The vapor phase is assumed to be ideal, while activity coefficients are considered for the liquid phase. The latter are determined according to Wilson's model based on equations and parameters from Gmehling & Kolbe (1992, p. 240-241). In the 2ph section, the vapor and liquid phases coexist on the shell-side. Thermodynamic equilibrium is assumed between the phases. However, the compositions change from the state of saturated vapor (dew point) to the boiling liquid state (bubble point). In the model, geometrical and hydraulic quantities such as heat transfer surface area per section as well as liquid and gas volume and hold up are also calculated. The total heat transfer surface area and volume are already specified by the chosen heat exchanger geometry including the number and dimension of the tubes. It is not specified with which temperatures the hot and the cold stream exit at the respective outlets. The VLEs at the inlet and outlet of the two-phase section represent a complex system, analogous to the *Flash Unit*, which is solved without problems by the hybrid approach. Accordingly, the temperatures $T^{ph2,h,in}$,

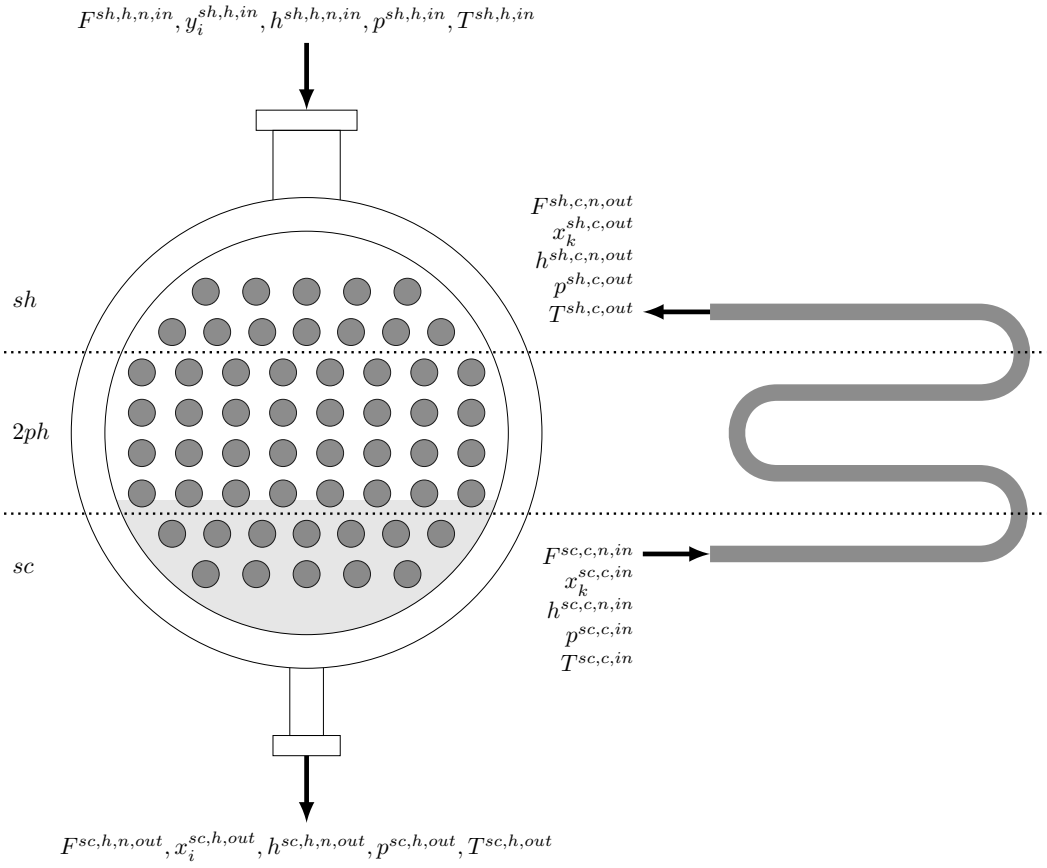


Fig. 4.3: Sketch of the Total Condenser model.

$T^{sc,h,in}$ and enthalpies $h^{ph2,h,n,in}$, $h^{sc,h,n,in}$ of the hot stream are clearly determined for the given pressure and inlet concentration. The temperature and enthalpy of the hot stream after subcooling, $T^{sc,h,out}$ and $h^{sc,h,n,out}$, as well as the temperatures $T^{ph2,c,in}$, $T^{sh,c,in}$, $T^{sh,c,out}$ and enthalpies $h^{ph2,c,n,in}$, $h^{sh,c,n,in}$, $h^{sh,c,n,out}$ of the cooling stream remain unknown. They can only be determined as a function of the surface areas available for heat transfer in each section A^{sh}, A^{ph2} and A^{sc}, which in turn depend on the unknown temperature differences. Thus, these relations build the largest complex subsystem of this NLE with a dimension of 12. Tests have shown that the formulation of the heat transfer has a fundamental role to play in order to sufficiently reduce the intervals of the temperatures and heat transfer surface areas. For one section, the total energy balance around the section and the energy balance around its tube-side are formulated in the following way, shown for the

superheated section

$$0 = F^{sh,h,n,in} \cdot h^{sh,h,n,in} - F^{sh,h,n,out} \cdot h^{sh,h,n,out} + F^{sh,c,n,in} \cdot h^{sh,c,n,in} - F^{sh,c,n,out} \cdot h^{sh,c,n,out} \tag{4.3}$$

$$0 = F^{sh,c,n,out} \cdot \left(\sum_{k=1}^{NCk} x_k^{sh,c,out} \cdot h_k^{sh,c,n,L,out}\right) - F^{sh,c,n,in} \cdot \left(\sum_{k=1}^{NCk} x_k^{sh,c,in} \cdot h_k^{sh,c,n,in}\right) - k^{sh} \cdot A^{sh} \cdot \frac{\left(T^{sh,h,in} - T^{sh,c,out}\right) - \left(T^{sh,h,out} - T^{sh,c,in}\right)}{\ln\left(\frac{\left(T^{sh,h,in} - T^{sh,c,out}\right)}{\left(T^{sh,h,out} - T^{sh,c,in}\right)}\right)} \quad . \tag{4.4}$$

Consequently, the explicit calculation of the mixed streams' molar enthalpies $h^{sh,c,n,in}$, $h^{sh,c,n,out}$ and the heat flux Q^{sh} by

$$h^{sh,c,n,in} = \sum_{k=1}^{NCk} x_k^{sh,c,in} \cdot h_k^{sh,c,n,L,in} \tag{4.5}$$

$$h^{sh,c,n,out} = \sum_{k=1}^{NCk} x_k^{sh,c,out} \cdot h_k^{sh,c,n,L,out} \tag{4.6}$$

$$Q^{sh} = k^{sh} \cdot A^{sh} \cdot \frac{\left(T^{sh,h,in} - T^{sh,c,out}\right) - \left(T^{sh,h,out} - T^{sh,c,in}\right)}{\ln\left(\frac{\left(T^{sh,h,in} - T^{sh,c,out}\right)}{\left(T^{sh,h,out} - T^{sh,c,in}\right)}\right)} \tag{4.7}$$

are omitted. If Eq. 4.5 - 4.7 were implemented and their respective terms in Eq. 4.4 replaced, the linkage is lost in IA that for example the value of $T^{sh,c,in}$ in Eq. 4.7 must be the same as in the functional expression $h_k^{sh,c,n,L,in}(T^{sh,c,in})$ of Eq. 4.5. Especially during the contraction of $\overline{\underline{T}}^{sh,c,in}$ or $\overline{\underline{T}}^{sh,c,out}$ in Eq. 4.4, the suggested formulation achieves tighter bounds. In consequence, the heat transfer surface area can then be further reduced in the upcoming contraction step, so that all in all the sequence of reduced boxes converges more quickly. The reformulation is also applied for the other two sections. Beyond these reformulations, the three auxiliary equations

$$0 = T^{out,c,sc} - T^{sc,c,in} - av^{sc} \tag{4.8}$$

$$0 = T^{sh,c,out} - T^{ph2,c,out} - av^{sh} \tag{4.9}$$

$$0 = T^{ph2,c,out} - T^{sc,c,out} - av^{ph2} \tag{4.10}$$

are implemented with the auxiliary variables av^{sc}, av^{ph2}, av^{sh} whose initial intervals are all set to $[10^{-2}, 10^{9}]$. The general formulation of the logarithmic temperature difference ΔT_{ln} for a counter-current heat exchanger is

$$\Delta T_{ln} := \frac{\overbrace{(T^{h,in} - T^{c,out})}^{:=\Delta T_1} - \overbrace{(T^{h,out} - T^{c,in})}^{:=\Delta T_2}}{\ln\left(\frac{T^{h,in} - T^{c,out}}{T^{h,ou} - T^{c,in}}\right)} \tag{4.11}$$

for the cold side c and the hot side h. The logarithmic temperature difference is also used in Eq. 4.4. It is only valid for the case when $\Delta T_1 \neq \Delta T_2$. In the real number range this condition is not fulfilled at a single point. For example, if $T^{h,in}$, $T^{h,out}$ and $T^{c,in}$ are known, $T^{c,out}$ must not be equal to $T^{h,in} - T^{h,out} + T^{c,in}$. The contraction method Bnormal is capable to filter such a discrete singular point. For the temperature intervals $\underline{\overline{T}}^{h,in}$, $\underline{\overline{T}}^{h,out}$ and $\underline{\overline{T}}^{c,in}$, however, there are infinitely many cases as soon as

$$\underbrace{\underline{\overline{\Delta T}}_1 \cap \underline{\overline{\Delta T}}_2}_{:=\underline{\overline{\Delta T}}_3} \neq \emptyset \quad . \tag{4.12}$$

$\underline{\overline{\Delta T}}_3$ cannot be filtered out when its width is greater than the tolerances. Thus, $\underline{\overline{\Delta T}}_{ln}$ extends to $[-\infty, \infty]$. If this is true for the subcooled section, $\underline{\overline{\Delta T}}_{ln}$ also contains the case where $\Delta T_{ln} = 0$ and the energy balance on the tube-side becomes independent of A^{sc}

$$\begin{aligned} 0 = {} & F^{sc,c,n,out} \cdot \Big(\sum_{k=1}^{NCk} x_k^{sc,c,out} \cdot h_k^{sc,c,n,L,out}\Big) - F^{sc,c,n,in} \cdot \Big(\sum_{k=1}^{NCk} x_k^{sc,c,in} \cdot h_k^{ph2,c,n,out}\Big) \\ & - k^{sc} \cdot A^{sc} \cdot 0 \quad . \end{aligned} \tag{4.13}$$

This equation is fulfilled, if the enthalpy stream difference between section inlet and outlet is zero. In this case, there are infinitely many solutions for A^{sc}, i.e., $\underline{\overline{A}}^{sc}$ cannot be reduced beyond the total heat transfer surface area of the condenser and neither do the dependent variables such as the other heat transfer areas, outlet temperatures, and hold ups. No enthalpy stream difference between inlet and outlet means no heat is transferred in this section and $T^{sc,c,in} = T^{sc,c,out}$. Finally, for all three sections the respective conditions are excluded by Eq. 4.8 - 4.10. The

complete model, variable and parameter specifications, and the physical solution within the investigated range can be found in Appendix B.6.

4.5 Heavies Column

Three components, namely Toluene (1), Biphenyl (2), Benzene (3), enter this column, in which the high-boiling component Biphenyl is removed from the rest. The feed is supplied on the column's top tray, i.e., only the stripping section is used to reduce the loss of Toluene and Benzene in the liquid residual. The column originates from the HDA process, which has been implemented as part of the master thesis of Rajes (2020). The VLE on the trays is described by a φ-φ approach. Fugacity coefficients and enthalpies are calculated based on SRK's equation of state as described in Rao (1997, pp. 74-79, p. 280, p. 321). In total, the model contains five separation trays, plus a total condenser and a partial reboiler. The model is sketched in figure 4.4. Thermodynamic equilibrium between the phases is assumed on all trays and in the heat exchangers. Reflux ratio R^C and boilup ratio R^R are fixed as design specifications. The influence of an increasing number of trays from 5 to 7, 10, 15, and 20 trays on the hybrid approach will be investigated in the next chapter. The model equations, in turn, can be found in Appendix B.8. The NLE with a total of 171 equations can be decomposed into 13 subsystems using DM, of which 12 are one-dimensional. However, a large, complex system remains. The reason is that a part of the unknown outlet streams from the condenser and reboiler are returned to the tray section and thus represent an unknown input at the same time, i.e., the models of the condenser, tray section and reboiler can only be solved simultaneously. Each of them contains the typical equations according to the ***M**aterial balance, **E**quilibrium, **S**ummation, **H**eat equations* (MESH) approach (Henley and Seader, 1981, pp. 556-560) plus the specific correlations the VLEs and enthalpies are calculated with. Only the pressures on the trays, in the condenser and in the reboiler as well as a few parameters of pure components can be calculated outside of the complex subsystem in advance. In case of the latter, some formulations of the equations are particularly important for the performance of the hybrid approach. They are presented now. According

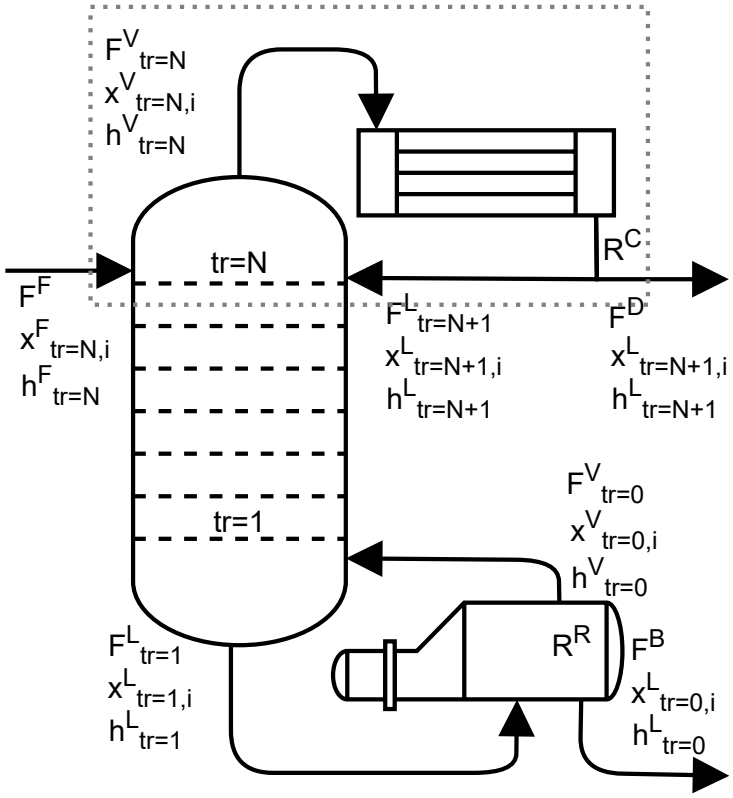


Fig. 4.4: Sketch of the Heavies Column model.

to the φ-φ concept the phase equilibrium is expressed by

$$0 = K_{tr,i} \cdot x_{tr,i} - y_{tr,i} \tag{4.14}$$

$$0 = K_{tr,i} \cdot \varphi^V_{tr,i} - \varphi^L_{tr,i} \quad , \tag{4.15}$$

with the molar fractions $x_{tr,i}$, $y_{tr,i}$, fugacity coefficients $\varphi^L_{tr,i}$, $\varphi^V_{tr,i}$ of liquid (L) and vapor phase (V) and the equilibrium constant $K_{tr,i}$. The fugacity coefficients $\varphi^L_{tr,i}$ and $\varphi^V_{tr,i}$ are originally calculated by the function

$$\begin{aligned} \varphi = \exp((Z-1) \cdot \frac{b}{b_{mix}} - \ln(Z-B) \\ + \frac{a_{mix}}{b_{mix} \cdot R \cdot T} \cdot (\frac{b}{b_{mix}} - 2 \cdot (\frac{a}{a_{mix}})^{0.5}) \cdot ((2-n) \cdot \ln(\frac{Z+B}{Z}) \\ + (n-1) \cdot \frac{1}{2 \cdot (2)^{0.5}} \cdot \ln(\frac{Z+B \cdot (1+(2)^{0.5})}{Z+B \cdot (1-(2)^{0.5})}))) \quad , \end{aligned} \tag{4.16}$$

i.e., the right-hand side of Eq. 4.16 is inserted directly into Eq. 4.15. In the Bachelor thesis of Reum (2022) it was found that the following reformulation of Eq. 4.15 can be processed much faster in a reduction step

$$0 = lnphi^L_{tr,i} - lnphi^V_{tr,i} \cdot \ln\left(K_{tr,i}\right) \quad , \tag{4.17}$$

where, instead of the fugacity coefficients, $lnphi^L_{tr,i}$ and $lnphi^V_{tr,i}$ are implemented directly as function variables. The new function to calculate them is

$$\begin{aligned} lnphi = {} & (Z-1) \cdot \frac{b}{b_{mix}} - \ln(aux) \\ & + \left(\frac{b}{R \cdot T} \cdot \left(\frac{(a_{mix})^{0.5}}{b_{mix}} - \frac{(a)^{0.5}}{b}\right)^2 - \frac{a}{b \cdot R \cdot T}\right) \cdot \left((2-n) \cdot \ln\left(1 + \frac{B}{Z}\right)\right. \\ & \left. + (n-1) \cdot \frac{1}{2 \cdot (2)^{0.5}} \cdot \ln\left(1 + \frac{2 \cdot (2)^{0.5}}{\frac{Z}{B} + 1 - (2)^{0.5}}\right)\right) \quad . \end{aligned} \tag{4.18}$$

This reformulation tries to reduce as many variable instances as possible compared to Eq. 4.16 to prevent interval dependency. The quantities a, b, n, R, T are real-valued parameters in this simulation so that their number of occurrences is not important. The aux variable contains one auxiliary variable each for the liquid and vapor phase, which are calculated as follows

$$0 = Z^L_{min,tr} - B^L_{tr} - aux^L_{tr} \tag{4.19}$$

$$0 = Z^V_{max,tr} - B^V_{tr} - aux^V_{tr} \quad . \tag{4.20}$$

This prevents the logarithmic expression in the second term of Eq. 4.18 to become $[-\infty,\infty]$, as soon as $\overline{Z} \cap \overline{B} \neq \emptyset$ is satisfied. Such a nonempty set cannot be filtered out by the contraction method Bnormal if its width is larger than the required tolerances. However, by allowing only positive values for the auxiliary variables ($\underline{aux}^L_{tr}, \underline{aux}^V_{tr} > 0$) it is excluded from the feasible region. Another reformulation concerns the use of the algorithm HC4revise. It cannot handle negative numbers inside a cubic root and instead returns the empty set. In reality, the cubic root has three solutions, one of which is real and the other two complex. In the case considered here, this problem occurs in the equations for calculating the compressibility factors $Z^L_{min,tr}$ and $Z^V_{max,tr}$, where only the real solutions are of

interest. The original form for the liquid phase is

$$Z^L_{min,tr} = \frac{\frac{-q^L_{tr}}{2} + (D^L_{tr})^{0.5}}{(\frac{-q^L_{tr}}{2} + (D^L_{tr})^{0.5})^{\frac{2}{3}}} + \frac{\frac{-q^L_{tr}}{2} - (D^L_{tr})^{0.5}}{(\frac{-q^L_{tr}}{2} - (D^L_{tr})^{0.5})^{\frac{2}{3}}} - \frac{\alpha^L_{tr}}{3} \quad . \tag{4.21}$$

Eq. 4.21 is replaced by the three equations

$$0 = aux1^L_{Z,tr} + aux2^L_{Z,tr} - \frac{\alpha^L_{tr}}{3} - Z^L_{min,tr} \tag{4.22}$$

$$0 = \frac{-q^L_{tr}}{2} + (D^L_{tr})^{0.5} - (aux1^L_{Z,tr})^3 \tag{4.23}$$

$$0 = \frac{-q^L_{tr}}{2} - (D^L_{tr})^{0.5} - (aux2^L_{Z,tr})^3 \quad . \tag{4.24}$$

The same procedure is pursued for the vapor phase. The complex solutions are avoided in this formulation, and the contraction methods can process those. The complete model, its variable and parameter specifications, and a physical solution are presented next. Regarding the classical MESH equations of the total condenser, its component balances, the summation relation of the liquid phase and the energy balance are replaced by

$$y_{tr=N,i} = x_{tr=N+1,i} \tag{4.25}$$

$$F^V_{tr=N} = F^D \cdot (R^C + 1) \tag{4.26}$$

$$F^L_{tr=N+1} = F^D \cdot R^C \tag{4.27}$$

$$0 = F^D \cdot (h^V_{tr=N+1} - h^L_{tr=N+1}) + Q^C \quad . \tag{4.28}$$

The total mole balance Eq. 4.26 is expressed using the reflux ratio. The reason is that instances of iteration variables can be reduced in this manner. Only two of the three unknown flows $F^V_{tr=N}$, $F^L_{tr=N+1}$ and F^D remain, since the reflux ratio is already specified. The same applies for the heat balance Eq. 4.28. Eq. 4.26 and 4.27 are implemented as functions, i.e., if the calculated variables $F^V_{tr=N}$ and $F^L_{tr=N+1}$ appear in an equation, they are replaced by the functional relationship given by the right-hand side of the equation. Accordingly, they do not appear as iteration variables and no initial bounds need to be specified. Especially for box consistency methods such as Bnormal, this offers further reduction potential as hidden dependencies are not lost (compare the examples from table 2.3). Since for

the distillate flow rate of a stationary, flow-driven simulation of a column already meaningful initial bounds can be given, i.e., it is non-negative and never exceeds the flow rate of the feed, it remains as iteration variable. Another option would have been to replace one of the functions 4.27 and 4.28 by F^D as a function of R^C, $F^L_{tr=N+1}$ and/or $F^V_{tr=N}$. Similar to the total condenser, the internal flow rates of the partial reboiler are expressed by the functions 4.29 and 4.30.

$$F^L_{tr=1} = F^B \cdot (R^R + 1) \tag{4.29}$$

$$F^V_{tr=0} = F^B \cdot R^R \tag{4.30}$$

Component balances and energy balance are formulated according to the Eq. 4.31 and 4.32.

$$0 = F^B \cdot ((y_{tr=0,i} - x_{tr=0,i}) - (R^R + 1) \cdot (y_{tr=0,i} - x_{tr=1,i})) \tag{4.31}$$

$$0 = F^B \cdot ((h^V_{tr=0} - h^L_{tr=0}) - (R^R + 1) \cdot (h^V_{tr=0} - h^L_{tr=1})) \tag{4.32}$$

The component balances of the tray section are replaced by component balances whose balance volume ends at the outlet of the distillate and begins at the vapor inlet of the current tray. This balance volume is sketched for the top tray in figure 4.4 by the dotted line. Thus, the balance equations of the generic tray model turn into

$$0 = \sum_{s=tr}^{NST} F^F_{tr=s} \cdot x^F_{tr=s,i} + F^V_{tr-1} \cdot y_{tr-1,i} - F^D \cdot x_{tr=N+1,i} - F^L_{tr} \cdot x_{tr,i} \tag{4.33}$$

$$0 = \sum_{s=tr}^{NST} F^F_{tr=s} \cdot h^F_{tr=s} - F^D \cdot h^L_{tr=N+1} + F^V_{tr-1} \cdot h^V_{tr-1} - F^L_{tr} \cdot h^L_{tr} + Q_{tr} + Q^C \quad . \tag{4.34}$$

In these formulations, it again pays off that the initial interval of the distillate flow can be chosen more precisely than the intervals of the internal flows. In flowsheet simulators, the feed streams of a column as well as their enthalpies are usually specified in the model and thus, enter it as real defined values.

5 Computational Experiments

The hybrid approach is now studied on the test cases presented in the last chapter regarding its capability as a global solver in section 5.2, i.e., to find all solutions in a given initial box and as a local solver in section 5.3, i.e., to automatically initialize root-finding algorithms that can quickly find one solution. The analysis parameters introduced in section 3.10 are used to evaluate its performance. Previously, settings related to the IA-based contraction will be examined in section 5.1 to ensure it to be highly efficient in the hybrid approach. All tests have been performed on a Notebook Intel i7 Processor (8x 1.8 GHz, 8. Generation) and 16 GB RAM with a Linux operating system. The results of all test runs can be downloaded as a zip-file from depositonce[1].

5.1 Contraction

To identify an efficient way of contracting NLE systems, various combinations of the contraction methods from section 3.1 have been applied on the test cases introduced in chapter 4. In all scenarios, only the first step of the hybrid approach is applied until consistency is reached starting from initial variable intervals that are all set to $[-10^9, 10^9]$. The relative and absolute tolerances are chosen as $\varepsilon^{Rel} = 10^{-3}$ and $\varepsilon^{Abs} = 10^{-8}$ and the resolution is set to 8. Besides, the maximum number of boxes is limited to 1 initially. In order to also consider the removal of discontinuities in the interior of variable ranges, three reduction steps are subsequently applied, as the maximum number of boxes is one in the first reduction step. In the second reduction step the maximum number of boxes is increased by one and a discontinuity can be removed so that both resulting boxes are further

[1] https://doi.org/10.14279/depositonce-18207

Tab. 5.1: Results of combined contraction method, initialized with $\boldsymbol{x}^{(0)} = [-10^9, 10^9]^{n_{var}}$. The following settings were applied: resolution = 8, $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$ and the notation is: Interval Newton (n), Bnormal (bc), optionally for Bnormal: tighten_bounds (tb), HC4revise (hc).

Contraction / NLE	**hc_n_bctb** ε^{RADL}	**CPU (s)**	**hc_n_bc** $\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**CPU (s)**
CSTR	1.61×10^{-8}	0.30	0.0	0.29
Flash Unit	4.40×10^{-1}	0.85	0.0	0.81
Total Condenser	1.58×10^{-1}	27.42	4.05×10^{-5}	12.25
Heavies Column	7.08×10^{-1}	449.50	0.0	133.78

reduced. Finally, the relative average domain length ε^{RADL} and the CPU time are tracked.

The results of the best combinations of contraction methods regarding the reduction of the initial variable domain are shown in table 5.1. All other combinations tested can be found in Appendix C.1. For the four examples, the lowest ε^{RADL} values could be achieved by combining all contraction methods (HC4revise, Interval Newton and Bnormal). Only in the case of the *Total Condenser*, the additional feature tighten_bounds could lower the ε^{RADL} further. However, the CPU time increases noticeably through tighten_bounds. It is more than three times higher in case of the *Heavies Column*, without any reduction in ε^{RADL}. Table 5.2 presents the results of the single contraction methods: HC4revise and Bnormal. The Interval Newton cannot reduce any of the initial boxes of the four examples without the other contraction methods. Hence, its results are not shown here. However, all values of ε^{RADL} are higher in the single contraction methods than in the combined versions, presented in table 5.1. Thus, it seems reasonable to alternate between the three contraction methods, despite the higher CPU time. The order of the contraction methods in the combined versions is chosen as follows:

$$1.)\ HC4revise \ \rightarrow\ 2.)\ IntervalNewton \ \rightarrow\ 3.)\ Bnormal \quad .$$

Tab. 5.2: Results of single contraction methods: HC4revise (hc) and Bnormal (bc), initialized with $\boldsymbol{x}^{(0)} = [-10^9, 10^9]^{n_{var}}$. The following settings were applied: resolution $= 8$, $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$.

Contraction /	**hc**		**bc**	
NLE	$\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**CPU (s)**	$\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**CPU (s)**
CSTR	1.14×10^{-4}	0.16	3.08×10^{-3}	2.20
Flash Unit	2.17×10^{-9}	0.66	9.52×10^{-9}	3.84
Total Condenser	1.82×10^{0}	3.38	1.24×10^{0}	99.75
Heavies Column	2.98×10^{-10}	48.52	3.80×10^{-1}	56.78

The contraction starts with HC4revise, because it is the fastest method with the lowest ε^{RADL} values in three out of the four examples. When HC4revise reaches consistency, Interval Newton continues as it is faster than Bnormal (see Appendix C.1), and potentially HC4revise was already able to tighten the box well enough for it to proceed. Finally, Bnormal is used to reduce the bounds. As long as any of the currently applied methods can further contract the box, the approach cycles through them until consistency is reached, and the reduction step terminates. In the following investigations, the combination of all three contraction methods is always applied. Optionally, the feature tighten_bounds is examined to check whether it can contribute to a faster solution identification.

For the two versions (with and without tighten_bounds), the influence of the resolution parameter is investigated. On one hand, it determines into how many parts an interval, where a function is non-monotonic, is subdivided in order to check them for root inclusion and to remove empty parts. On the other hand, it sets the number of refinements applied in the tighten_bounds algorithm. For the two combinations, the resolution parameter is successively set to 2^n with $n = 0,1,2,\ldots,7$ so that in total 16 cases per test example are investigated. Again, the approach is initiated at $\overline{\underline{\boldsymbol{x}}}^{(0)} := [-10^9, 10^9]^{n_{var}}$ and three reduction steps are applied. Only in the *Total Condenser* an increasing resolution can lower ε^{RADL} at all. Figure 5.1 shows the change in ε^{RADL} and CPU time depending on the resolution for the two versions. Through applying tighten_bounds ε^{RADL} declines clearly

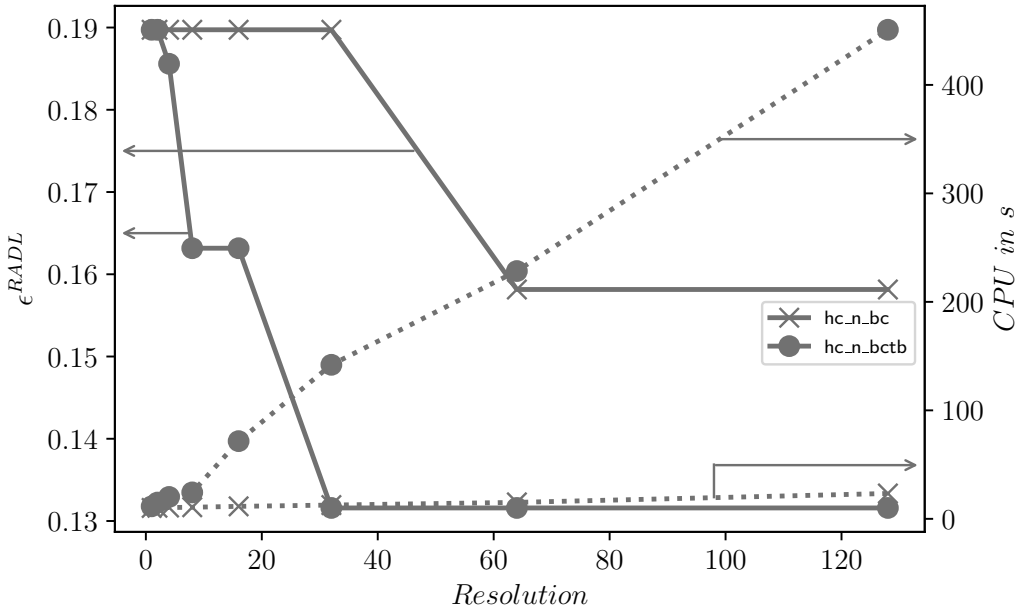


Fig. 5.1: Relative Average Domain Length (left) and CPU time (right) of Total Condenser after three reduction steps, initialized with $\boldsymbol{x}^{(0)} = [-10^9, 10^9]^3$, set tolerances: $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$.

with increasing resolution. However, the CPU time increases noticeably. Without tighten_bounds, ε^{RADL} hardly changes up to a resolution of 32. After that, there is a large drop in ε^{RADL}, which is related to the reduction of a non-monotonic function domain. However, the CPU time grows less than with tighten_bounds. While ε^{RADL} does not change with rising resolution in the other test cases, their CPU time still increases. Like the *Total Condenser*, the version with tighten_bounds takes always longer than without. In particular, the iteration of the largest NLE, which is the *Heavies Column*, lasts for more than 2 h, when tighten_bounds is applied at a resolution of 32. A good compromise seems to be a resolution of 8, because when tighten_bounds is used, it can reduce intervals to some extend without noticeably slowing down the reduction of large NLEs. For simulations beyond this work, it is recommended to choose an integer value out of the range 4 to 32 with tighten_bounds and 8 to 64 without.

Tab. 5.3: Varied options in IA iteration tests.

Parameter	Tested Options	Shorthand Notation
tightBounds	`true`; `false`	tb; -
cutBox	`all`; `tear`	cba; cbtv
splitBox	`forecastSplit`; `forecastTear`;	fcs; fctv;
	`tearVar`; `leastChanged`	tv; lc

5.2 Global Solver

This section addresses the research question of how well the hybrid approach is suited as a global solver. Its performance is compared to the pure IA-based solver without root-finding step. If a numerical solution can be found at an early stage by root-finding, there is the possibility that the box surrounding it fulfills the unique solution test before the box becomes degenerate. Hence, further box reduction steps could be saved compared to the IA-based solver. As part of the examinations, we also check which settings for cutting and splitting are best suited to solve the system globally as quickly as possible. Two options for cutting and four for splitting are tested, whose abbreviations are shown in table 5.3. For the two versions of contraction, this results in 16 settings to be tested for each NLE. The variable bounds are set to $[-10^9, 10^9]$. If a system cannot be globally solved in 3000 s, the variable bounds are first coarsely narrowed. If this still does not resolve the problem, then the method that achieves the largest box reduction measured by ε^{RADL} in 3000 s is considered to be the most efficient. The relative and absolute tolerances are set to $\varepsilon^{Rel} = 10^{-3}$ and $\varepsilon^{Abs} = 10^{-8}$ and the resolution value equals 8. Only for the *Heavies Column* the values of ε^{Rel} and ε^{Abs} need to be adjusted to 10^{-10} and 10^{-15} to compute one mole fraction with the lowest system order of 10^{-13}. As root-finding algorithm, Scipy's Fsolve is applied in the hybrid approach.

Tables 5.4 and 5.5 show the results of the tests. Only the *CSTR* and the *Flash Unit* can be solved globally. The IA-based solver is faster than the hybrid approach in both cases. A successful unique solution test occurs in both systems only in

the last reduction step, i.e., when the boxes are almost degenerate. Since several root-finding steps were performed during the iteration, the hybrid approach is also slower than the IA-based solver. The latter, on the other hand, is very well suited for both systems, since no process knowledge is required for initialization, and all solutions have been found in a short time. The two models differ from the other test examples in the small dimensions of their largest, complex subsystems, which are three and eight.

Tab. 5.4: Most efficient global solution strategies of small systems in initial boxes $\underline{\overline{x}}^{(0)} := [-10^9, 10^9]^{n_{var}}$.

NLE	**Cut & Split**	n_{step}	**IA-based Solver**		**Hybrid Approach**	
			CPU (s)	$\frac{n_{solved}}{n_{box}}$	**CPU (s)**	$\frac{n_{solved}}{n_{box}}$
CSTR	cbtv_lc	7	0.93	3/3	0.98	3/3
Flash Unit	cbtv_tv	20	13.15	2/2	15.04	2/2

Tab. 5.5: Most efficient global solution strategies of intermediate and large systems in more restricted initial boxes that are presented in Appendix C.5.

NLE	**Cut & Split**	n_{step}	**IA-based Solver**		**Hybrid Approach**	
			CPU (s)	ε^{RADL}	**CPU (s)**	$\frac{n_{solved}}{n_{box}}$
Total Condenser	cba_fcs	174	3156	7.6×10^{-6}	3139	0/2
Heavies Column	cbtv_ftv	55	3016	6.6×10^{-1}	9144	0/15

According to the results in table 5.5, the *Total Condenser* and the *Heavies Column* cannot be solved globally in 3000 s. For both systems, the variable bounds have been roughly adjusted and can be found in Appendix C.5. In the case of the *Total Condenser*, the root-finding algorithm Fsolve finds a numerical solution after 471 s, but the unique solution test is not satisfied at any time. After 10 478 s, the

IA-based solver can solve one box of the *Total Condenser*. There are six other boxes present at this time. All intervals of the other boxes are either identical or close to the solved box. If the test run is continued, further boxes are solved. In order to label a box as solved, all of its variable intervals have to become degenerate. This is true for most of them, but a few exist that don't meet the tolerances although they are closeby. Hence, as they are the only ones unsolved, they are exhaustively bisected until each respective subbox is degenerate. In section 3.2 a method has been presented to prevent exhaustive splitting in one variable dimension. An alternative tolerance ε^{uni} is calculated and a unification of the subintervals returned. Nevertheless, this method does not work if multiple variables have to be bisected before the termination criterion is fulfilled. Only one variable at a time is bisected in a reduction step followed by further box contraction so that the connection to its formerly neighboring box is lost. In consequence, the solved boxes cannot be easily unified. With the split method fcs, the cut method cbtv and tight_bounds selected, the IA-based solver was able to solve all boxes after almost four hours. The total number of boxes is 22. The variables, whose intervals differ, are those describing the heat transfer in the subcooled and superheated section, i.e., the heat transfer surface areas A^{sc} and A^{sh} as well as the temperatures $T^{sc,h,out}$, $T^{sc,c,out}$, $T^{ph2,c,out}$ and $T^{sh,c,out}$, because each box lacks at least one intersection with another box in one of their dimensions. As the surface area of the two-phase section is three orders of magnitude larger than that of the superheated section A^{sh} and two orders larger than that of the subcooled section A^{sc}, the system seems to be ill-conditioned, which results in solution intervals for both, rather than distinct values. This leads to the differing section outlet temperatures as well. The ill-condition of the problem can also be seen in the high condition number of the Jacobian matrix evaluated at the numerically found solution ($\kappa_2 = 4.8 \times 10^6$).

In the case of *Heavies Column*, the resulting four boxes after more than 3000 s are not close to degeneration yet. Its largest reduction change with respect to ε^{RADL} happens in the first reduction steps according to figure 5.2. In the fifth reduction step, splitting occurs for the first time but the total box volume decreases only slightly, and thus converges quite slowly to the real solution(s). All of the variables that are independent of the largest complex subsystem are solved at this point. However, the IA-based solver can only slightly reduce the intervals of its

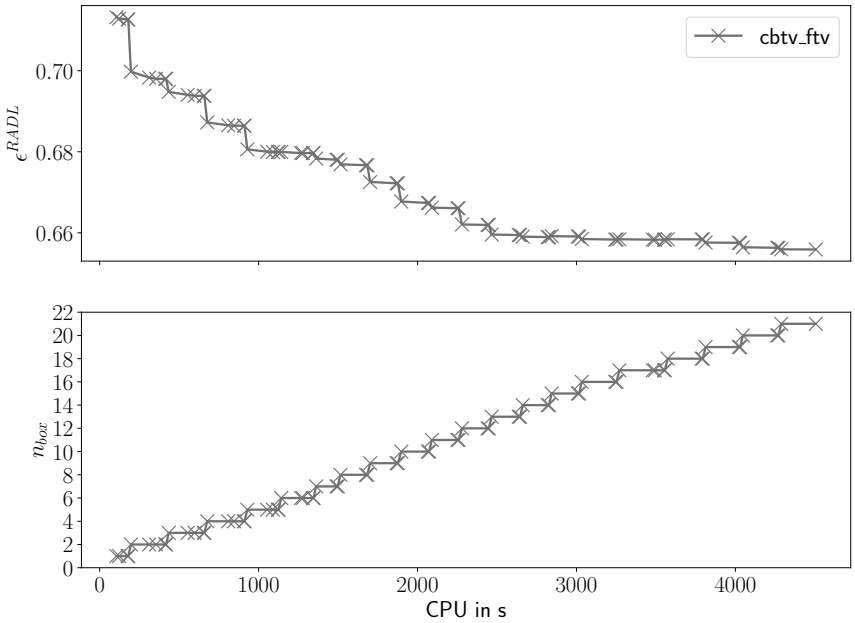


Fig. 5.2: Relative average domain length (top) and number of boxes over CPU time of *Heavies Column* with $\varepsilon^{Rel} = 10^{-10}$, $\varepsilon^{Abs} = 10^{-15}$ and resolution = 8.

iteration variables after the first box reduction step. In the hybrid approach, the root-finding algorithm Fsolve could not find the numerical solution once within the 3000 s, thereby the unique-solution test could not be fulfilled as well. NLEs with large, complex subsystems like this one, whose largest, complex subsystem has a dimension of 158, combined with a bad condition ($\kappa_2 = 3.4 \times 10^{10}$) and a high percentage of nonlinear terms ($\varepsilon_{nl} = 0.56$), cannot be solved globally in an acceptable time with the IA-based methods used here.

The cutting method is essential in all test cases to avoid excessive box splitting within the IA-based solver. Figure 5.3 shows the changes in relative average box length and number of boxes over the first 14 s with and without cutting for the *Flash Unit*. Without cutting, the number of split boxes increases steadily, while the reduction in ε^{RADL} stagnates. In general, however, it cannot be said whether all variables or only the tear variables should be used for cutting. In the *Total Condenser*, the extra effort that results from cutting all variables leads

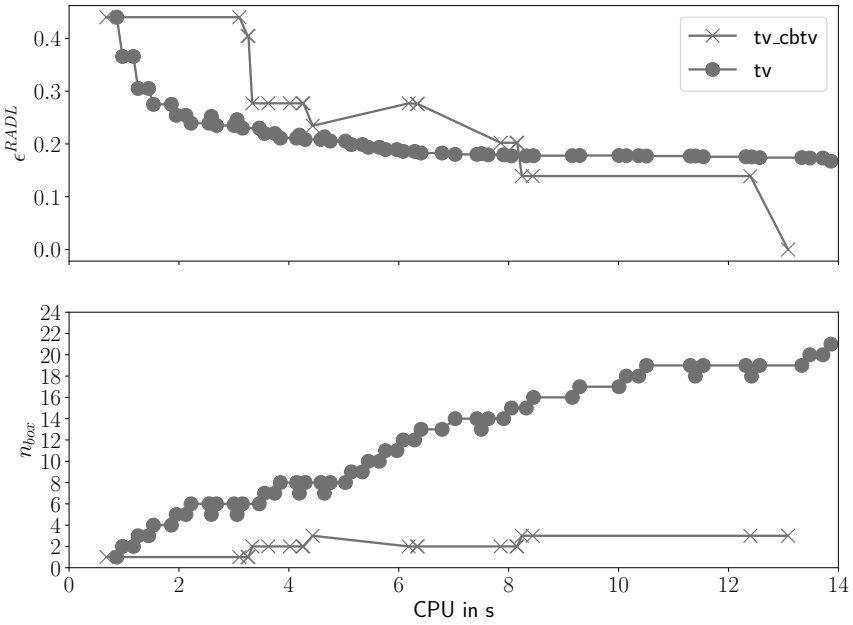


Fig. 5.3: Relative average domain length (top) and number of boxes over CPU time of *Flash Unit* with $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$, $\underline{\overline{\boldsymbol{x}}}^{(0)} = [-10^9, 10^9]^{29}$ and resolution = 8.

to less splitting and the hybrid approach works more efficiently than if only the tear variables are cut. This is not the case for the other three test examples, where cutting the tear variables is more efficient even if it results in more boxes. Regarding splitting, there is no apparent tendency either. However, in three of the four NLEs the most efficient IA methods (see table 5.4 and 5.5) are always related to one of the split strategies based on the method split_box, namely: lc, ftv and fcs. Just in the *Flash Unit*, tv is the most efficient choice, because this system has only one tear variable.

Finally, the results show that the hybrid approach applied as a global solver does not reduce the CPU time compared to the IA-based solver. The IA-based solver, in turn, is just able to find all solutions of the test cases whose largest complex subsystem does not exceed a dimension of 8 in 3000 s.

5.3 Local Solver

Now we compare the hybrid approach and classical root-finding algorithms with respect to their ability to locate solutions quickly. The question is whether the box reduction of the hybrid approach is sufficient to generate an initial point in short time, from which the selected root-finding algorithm converges. In the tests, the root-finding step of the hybrid approach is fulfilled by a certain state-of-the-art solver and its performance is compared to a corresponding reference run, in which only the solver is applied. As state-of-the-art root-finding algorithms Fsolve, SLSQP, IPOPT and Newton[2] are examined. The Newton solver is tested with no scaling as well as the two scaling methods MC29 and MC77, which have been introduced in section 2.4.1. For the reference runs, each example is attempted to be solved from the initial point $\boldsymbol{x} = (0.5, \ldots, 0.5)^T$ in the unbounded box, i.e., all initial intervals are set to $[-10^9, 10^9]$, or the adjusted boxes from the last section. When intervals have been adjusted their midpoint is taken as initial point. For the *Flash Unit*, the bounds of the mole fraction were reduced to $[0, 1]$ to retain only the physically relevant solution. Unless otherwise explicitly mentioned, 10,000 iteration steps and a function tolerance of 10^{-8} are used for these root-finding reference runs. The hybrid approach is initialized with the identical bounds that are used in the reference runs. All 16 possible settings for contraction, cutting and splitting are tested to identify the fastest strategy for each test case. Resolution, tolerance settings, and number of maximum iteration steps for root-finding in the hybrid approach correspond to the values of the last section. In general, a maximum of 10 box reduction steps is used, because as already shown in the last section, the initial box volume reduces the most in the first reduction steps.

Table 5.6 contains the most efficient settings of the hybrid approach for each of the root-finding algorithms that is able to find a numerical solution. Only those root-finding algorithms are listed, that could solve the NLE as part of the hybrid approach, reference run, or both. Generally, only IPOPT and Fsolve in case of the *CSTR* as well as IPOPT and SLSQP in case of the *Flash Unit* are able to find a solution without box reduction and are thus, faster than the hybrid approach. In both examples, however, one box reduction step suffices for most other root-finding algorithms to converge as well. Since after one box reduction step neither

[2]Used nomenclature: fslv for Fsolve, slsqp for SLSQP, ipopt for IPOPT and nwt for Newton

Tab. 5.6: Comparison of root-finding with hybrid approach for the test cases.

NLE	Solver	Cut & Split	n_{step}	CPU (s), Runs: Reference	CPU (s), Runs: Hybrid Approach
CSTR	fslv	–	1	0.03	0.10
	ipopt	–	1	0.06	0.13
	nwt	–	1	–	0.10
	nwt_mc29	–	1	–	0.10
	nwt_mc77	–	1	–	0.10
	slsqp	–	1	–	0.13
Flash Unit	fslv	cbtv	3	–	3.05
	ipopt	–	1	0.54	0.72
	nwt	cbtv	3	–	3.36
	nwt_mc29	–	1	–	0.58
	nwt_mc77	–	1	–	0.60
	slsqp	–	3	0.41	3.18
Total	fslv	cbtv	3	–	39.54
Condenser	ipopt	–	1	–	9.65
	nwt_mc29	–	1	–	9.87
	nwt_mc77	–	1	–	9.00
	slsqp	–	1	–	9.24
Heavies	ipopt	–	1	–	98.12
Column					

cutting nor splitting occur (indicated by $-$ in table 5.6), the box reduction depends in this case only on the combined contraction by HC4revise, Interval Newton and Bnormal. In the two test runs of the *Flash Unit* by Newton and Fsolve, the hybrid approach requires three box reduction steps. Here, the solution is actually not found by the root-finding algorithms, in fact, the system is globally solved by the IA-based box reduction. Hence, another advantage of the hybrid approach is that, whenever a root-finding algorithm does not converge or converges to an undesired solution outside the variable bounds, the hybrid approach still keeps the possibility to find the solution within the bounds as a pure IA-based solver.

In the other two NLEs, numerical solutions can only be found by the hybrid approach. However, not all root-finding algorithms converge within the maximum of 10 box reduction steps. In the ill-conditioned *Total Condenser*, the results show that scaling the Newton algorithm leads to a successful iteration. The *Heavies Column* can only be solved by IPOPT. For this particularly challenging system, more effort has been put into the initialization of the reference run. The initial point and initial bounds can be found in the Appendix in table C.6 as well as in MOSAICmodeling's evaluation (ID: 165321). The initial values have been chosen carefully using prior process knowledge, e.g., the tray temperatures lie between the boiling temperatures of the non-azeotropic mixture and decrease towards the column's head. Furthermore, it has been considered that the functions' residuals of the root-finding problem take finite values at the initial point, i.e., the problem is feasible. None of the root-finding solvers converge under these conditions in the reference runs. Besides, an export to AMPL has been performed for comparison. The AMPL export from MOSAICmodeling creates an optimization problem from the NLE with constant objective function, for which the powerful presolve option can be switched on and is applied in advance. But even with this methodology the selected solver IPOPT cannot find a solution starting from the initial point or the midpoint of the initial box and aborts after a short time with a "Restoration Phase Failed" or "Converged to a point of local infeasibility" exception. However, one box reduction step applied seems to be sufficient here to reduce in particular the SRK parameters ($a_{mix,tr}$, $b_{mix,tr}$, a_{EoS}) and the auxiliary variables (aux_{tr}) to such an extent that IPOPT converges. There seem to be infeasible points or regions especially in their intervals, which otherwise, complicate a numerical iteration.

Tab. 5.7: Comparison of reduction efficiency after $n_{step} = 1$ and $n_{step} = 15$ box reduction steps applied on classical and reformulated NLE *Heavies Column*.

Model	$n_{step} = 1$	$n_{step} = 15$
$\varepsilon^{RADL}_{classical}$	0.7132	0.6777
$\frac{\varepsilon^{RADL}_{reformulated} - \varepsilon^{RADL}_{classical}}{\varepsilon^{RADL}_{classical}}$	3.11×10^{-14}	-2.49×10^{-3}

It is interesting to note that the hybrid approach only works well on the reformulated system as described in section 4.5. Comparing the boxes of the classical column model with those of the reformulated one, it is noticeable that the boxes differ only marginally in their $\varepsilon^{RADL,(1)}$ values after one box reduction step. The original system has even a slightly better performance on the $a^V_{mix,tr}$ parameters, which results in the lower $\varepsilon^{RADL,(1)}$. The reformulation has actually been performed to reduce the occurrence of the internal flow rates and their enthalpies and compositions in the associated balance equations of the tray section in order to improve the reduction performance on the initial box, which does not seem to be the case in the first reduction step. After 15 box reduction steps however, the $\varepsilon^{RADL,(15)}$ of the reformulated system is actually lower. Returning to the hybrid approach, this is still only successful in case of the reformulated system. In consequence, the actual improvement is achieved in the root-finding iteration. This is validated through initializing the reformulated system with the box obtained from one box reduction step by the classical model. In this case, IPOPT also converges starting from the midpoint of this box. One possible reason could be that the truncation error that occurs when balancing tray-to-tray is prevented, since in the reformulated system, the balance volume always reaches from the head of the column to the current tray.

The number of trays of the column has been increased to test an even larger NLE with a larger complex subsystem. The initial intervals of the tray related variables have been chosen in the same way as the ones from the model with five trays. The reference runs are again unsuccessful, while the hybrid approach converges for all tested numbers of trays applying IPOPT after one box reduction step. Time increases approximately linearly with number of trays as shown in figure 5.4,

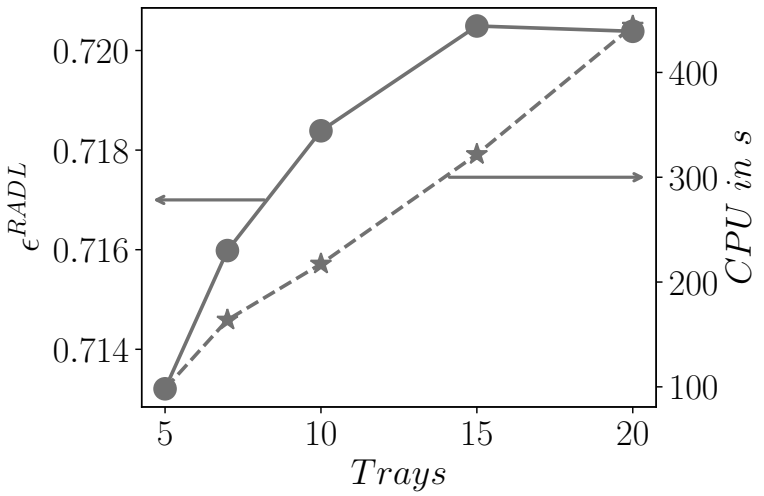


Fig. 5.4: Change of ε^{RADL} and CPU with increasing number of trays in the NLE *Heavies Column*.

while ε^{RADL} rises but seems to approach a value of 0.721. It is nice to see that if the operating range of the column can be roughly estimated, even an increasing number of trays does not present a major hurdle for the hybrid approach.

Finally, it is observed that the hybrid approach is able to solve all test cases in a short time and it is always the more robust solution strategy compared to conventional root-finding. Large NLEs can be solved with just a few box reduction steps and IPOPT applied, even where IPOPT in combination with the presolve strategy from AMPL fails. Cutting has not even been used in most test runs, since only one box reduction step is required before the root-finding algorithm converges. Splitting is not necessary at all. The method tighten_bounds can not achieve any improvement either. However, to ensure that the hybrid approach works efficiently, equations must be formulated carefully, as exemplary discussed in sections 4.4 and 4.5.

5.4 Parallelization

By means of the parallelization described in section 3.8, several boxes resulting from splitting can be processed simultaneously within one reduction step. This

Tab. 5.8: Comparison of fastest runs of parallelized IA-based solver out of 2,3 . . . 6 cores applied, represented by *par* and sequential process represented by *seq*.

NLE	**Cut & Split**	$\mathbf{Cores_{par}}$	$\frac{\varepsilon_{seq}^{RADL}}{\varepsilon_{par}^{RADL}}$	$\frac{\mathbf{CPU_{seq}}}{\mathbf{CPU_{par}}}$ **(s/s)**
Flash Unit	cbtv_tv	2	$\frac{6.32}{6.32} \times 10^{-10}$	$\frac{13.15}{11.21}$
Total Condenser	tb_cbtv_fcs	6	$\frac{2.78}{2.78} \times 10^{-6}$	$\frac{14351}{8706}$
Heavies Column	cbtv_ftv	4	$\frac{6.59}{6.57} \times 10^{-1}$	$\frac{3009}{1557}$

methodology is suitable when multiple boxes occur. The processing time of the *CSTR* is so short that the effort is pointless. When the hybrid approach was used as the local solver, no splitting occurred at all. Therefore, only the test cases, in which the pure IA-based solver searched for the global solutions, are parallelized. The *Heavies Column* is the only example of the four, which could not be solved globally. Hence, only the time is tracked until an equivalent ε^{RADL} to the sequential procedure is reached. A total of two to six process cores are examined for a specific cut and split strategy per example. Table 5.8 shows the results of the fastest runs with parallelization, denoted by *par*, and without, represented by *seq*. In all cases, parallelization reduces the CPU time. Depending on the number of boxes that occur during the iteration, different numbers of cores lead to the fastest runs. A maximum of 5, 34 and 14 boxes occur during IA-based reduction processes of the *Flash Unit*, *Total Condenser* and *Heavies Column* respectively. In this way, it can be concluded that the maximum number of cores does not always lead to the shortest time, but the more boxes occur during the iteration, the more cores should be used to reduce the CPU time.

6 Analysis of Results

In the completed studies, the hybrid approach convinces mainly as a local solution strategy. Only a few IA-based box reduction steps were necessary, before the state-of-the art root-finding algorithms quickly converged to a solution in all tested cases. Although the hybrid approach could globally solve the NLEs with smaller complex subsystem (dimension ≤ 8), it did not show any reduction of the CPU time by a successful unique solution test compared to the pure IA solver. Moreover, the hybrid approach can only efficiently solve the two NLEs with the larger complex subsystems, when they are initialized with some general process knowledge and not without the IA favorable reformulations. Generalized findings from the studies concerning reformulation and initialization are given in section 6.1 and section 6.2. Section 6.3 analyses the test results regarding the complexity of the examined process models. In section 6.4 to 6.6, the application areas for IA-based solver, root-finding algorithm and hybrid approach with respect to the model's complexity are discussed. Tools for debugging are introduced in section 6.7. Finally, other functionalities that have been tested as part of the hybrid approach to increase its efficiency without success so far, are presented in section 6.8, before summarizing the work in section 6.9, and giving an outlook in section 6.10.

6.1 Reformulation

In the scope of this work, the structure of equations was also investigated in order to find advantageous formulations regarding the reduction of intervals by IA-based methods and the speed of real arithmetic methods. This is illustrated now for small examples taken from the tested NLEs.

At first, the hybrid approach could not reduce the initial intervals of the mole fractions in the *Heavies Column* at all. Its VLE is described by the $\varphi - \varphi$ approach

$$\varphi_i^L \cdot x_i = \varphi_i^V \cdot y_i \quad . \tag{6.1}$$

In particular, the function for determining the fugacity coefficients φ_i^L and φ_i^L, which is

$$\begin{aligned} \varphi = \exp((Z-1)\cdot(\frac{b}{b_{mix}}) - \ln(Z-B) \\ + \frac{a_{mix}}{b_{mix}\cdot R\cdot T}\cdot(\frac{b}{b_{mix}} - 2\cdot(\frac{a}{a_{mix}})^{0.5})\cdot((2-n)\cdot\ln(\frac{Z+B}{Z}) \\ + (n-1)\cdot\frac{1}{2\cdot(2)^{0.5}}\cdot\ln(\frac{Z+B\cdot(1+(2)^{0.5})}{Z+B\cdot(1-(2)^{0.5})})) \quad , \end{aligned} \tag{6.2}$$

was responsible for the bad performance of the contracting methods, when applied on the intervals of the mole fractions. Evaluating Eq. 6.2 by IA, produces wide ranges of values for the fugacity coefficients due to interval dependency and non-degenerate intervals with undefined function expressions resulting in $\pm\infty$ interval bounds. Hence, these wide ranges could not be used for any interval reduction in Eq. 6.1. In the thesis of Reum (2022), various reformulations of Eq. 6.2 have been examined in order to reduce the intervals of the fugacity coefficients further. First, a significant reduction in the speed has been achieved by using the natural logarithm instead of the exponential function so that Eq. 6.2 becomes

$$\begin{aligned} lnphi = (Z-1)\cdot(\frac{b}{b_{mix}}) - \overbrace{\ln(Z-B)}^{:=TM_1} \\ + \overbrace{\frac{a_{mix}}{b_{mix}\cdot R\cdot T}\cdot(\frac{b}{b_{mix}} - 2\cdot(\frac{a}{a_{mix}})^{0.5})}^{:=TM_2}\cdot((2-n)\cdot\ln(\overbrace{\frac{Z+B}{Z}}^{:=TM_3}) \\ + (n-1)\cdot\frac{1}{2\cdot(2)^{0.5}}\cdot\ln(\overbrace{\frac{Z+B\cdot(1+(2)^{0.5})}{Z+B\cdot(1-(2)^{0.5})}}^{:=TM_4})) \quad . \end{aligned} \tag{6.3}$$

lnphi is an output variable here, not a function. Eq. 6.1 thus changes to

$$lnphi_i^L - lnphi_i^V = \ln\frac{y_i}{x_i} \quad . \tag{6.4}$$

The reduction in speed due to the replacement of the exponential term by the inverse operation of the natural logarithm is specific to the applied IA package mpmath . Code A.1 in the Appendix contains a simple computer experiment and its results, which verifies this. In order to improve the reduction via IA, Eq. 6.3 has been further reformulated. Thereby, critical terms (TM_1 to TM_4) were identified, generalized and corresponding reformulations developed to improve the reduction efficiency. The generalized form of these terms as well as some more reformulations, either identified by ourselves or found in the book of Moore et al. (2009, pp. 19-50), are summarized in table 6.1. Reformulations derived for Eq. 6.3 are discussed now. Subtracting two variables x and y in the denominator of fractions or in logarithms as in term TM_1, lead to undefined ranges of values in IA-based evaluation as long as the condition $\overline{\underline{x}} \cap \overline{\underline{y}} \neq \emptyset$ holds. Either this condition must be avoided by a suitable choice of initial intervals or an auxiliary variable aux with an initial interval of $0 < \overline{\underline{aux}}$ may be introduced. The latter is applied to replace TM_1

$$TM_1 = \ln aux \tag{6.5}$$

$$aux = Z - B, \quad aux > 0 \quad . \tag{6.6}$$

In TM_2, interval dependency is caused by the multiple occurrences of the variables a_{mix} and b_{mix}. All other variables R, T, a and b are either design variables or functions depending on design variables only so that their intervals are degenerate. The number of instances of a_{mix} and b_{mix} is reduced by means of quadratic completion so that TM_2 is exchanged by

$$TM_2 = \left(\frac{b}{R \cdot T} \cdot \left(\frac{(a_{mix})^{0.5}}{b_{mix}} - \frac{(a)^{0.5}}{b}\right)^2 - \frac{a}{b \cdot R \cdot T}\right) \tag{6.7}$$

The generalized form of quadratic completion is shown in row c) in table 6.1. Interval dependency occurs only when the multipliers p_1 and p_2 have different signs. In TM_3 the occurrence of Z is reduced to avoid interval dependency by

$$TM_3 = 1 + \frac{B}{Z} \quad . \tag{6.8}$$

TM_3 is equivalent to the inverse of function d) from table 6.1, so reformulation d) can also be applied here and leads to Eq. 6.8

$$TM_3 = \frac{Z+B}{Z} = \left(\frac{Z}{Z+B}\right)^{-1} = \left(\frac{1}{1+\frac{B}{Z}}\right)^{-1} = 1+\frac{B}{Z} \quad . \tag{6.9}$$

There are many other formulations that can be similarly put into such a standard form of table 6.1 to save the number of reformulation rules for special cases. Finally, TM_4 can be rewritten as

$$TM_4 = 1 + \frac{2 \cdot (2)^{0.5}}{\frac{Z}{B} + 1 - (2)^{0.5}} \tag{6.10}$$

to reduce the number of instances of Z and B. The generalized reformulation according to Eq. 6.10 is shown in row e) of table 6.1. One important, additional reformulation that improves the performance of the IA-based reduction is taken from the equation system of the *Total Condenser*, namely the calculation of the logarithmic temperature difference

$$\Delta T_{ln} = \frac{\Delta T_l - \Delta T_r}{\ln \frac{\Delta T_l}{\Delta T_r}} \quad . \tag{6.11}$$

To avoid value ranges, where the logarithmic function is not defined, i.e., when $\Delta T_l \cap \Delta T_r \neq \emptyset$, the quotient is substituted by an auxiliary variable aux with $0 < \underline{\overline{aux}}$. Eq. 6.11 is then replaced by

$$\Delta T_{ln} = \frac{\Delta T_l - \Delta T_r}{\ln aux} \tag{6.12}$$

$$aux = \frac{\Delta T_l}{\Delta T_r} \quad . \tag{6.13}$$

This corresponds to the generalization (i) in table 6.1. Table 6.1 is extensible and can be used as a guideline for formulating equations or as a basis for an automated reformulation algorithm.

Tab. 6.1: Recommended reformulations for hybrid approach.

Function	Reformulation	ID	Condition	Reason
$f(x) = \exp(x)$	$aux, \quad \ln(aux) = x$	a)	$-$	Speed
$f(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2$	$p_2 \cdot \left(x + \frac{p_1}{2 \cdot p_2}\right)^2 - \frac{p_1^2}{4 \cdot p_2} + p_0$	b)	$\lvert p_1 \rvert + \lvert p_2 \rvert \neq \lvert p_1 + p_2 \rvert$	Interval dependency
$f(x) = \sum_{p=0}^{n} p_p \cdot x^p$	$p_0 + x \cdot (p_1 + x \cdot (p_2 + \ldots$ $x_{n-1} \cdot (p_{n-1} + p_n \cdot x) \ldots)$	c)	$\sum_{p=0}^{n} \lvert p_p \rvert \neq \lvert \sum_{p=0}^{n} p_p \rvert$	Interval dependency
$f(x,y) = \frac{x}{y+x}$	$\frac{1}{\frac{y}{x}+1}$	d)	$-$	Interval dependency
$f(x,y) = \frac{y-x}{y+x}$	$1 - \frac{2}{\frac{y}{x}+1}$	e)	$-$	Interval dependency
$f(x,y) = \frac{x}{y-x}$	$\frac{1}{\frac{y}{x}-1}$	f)	$-$	Interval dependency
$f(x,y) = \frac{1}{\frac{y}{x}-1}$	$\frac{1}{aux-1}, \quad aux = \frac{y}{x}$	g)	$x \cap y \neq \emptyset$	Infeasibility
$f(x,y) = \frac{1}{y-x}$	$\frac{1}{aux}, \quad aux = y - x$	h)	$x \cap y = \emptyset$	Interval dependency
$f(x,y) = \frac{1}{y \cdot x - 1}$	$\frac{1}{aux-1}, \quad aux = y \cdot x$	i)	$\frac{1}{x} \cap y \neq \emptyset$	Infeasibility
$f(x,y) = \frac{1}{\ln\left(\frac{y}{x}\right)}$	$\frac{1}{\ln(aux)}, \quad aux = \frac{y}{x}$	j)	$x \cap y \neq \emptyset$	Infeasibility
$f(x,y) = \ln(y - x)$	$\ln(aux), \quad aux = y - x$	k)	$x \cap y \neq \emptyset$	Infeasibility

6.2 Initialization

Of course, the tighter the initial variable bounds are chosen, the less work and time is required for the hybrid approach. On the one hand, fewer contraction steps are needed, on the other hand, some infeasibilities disappear and the impact of interval dependencies is lower, so that excessive splitting can be prevented. Table D.1 to table D.3 in the Appendix contain guidelines according to which variables can be initialized under certain model conditions. However, this list has a strong focus on the case studies of this thesis and can be expanded for new process units.

6.3 Model Complexity

A definition of complex systems, as used in the work, has already been given in section 4.1. Parameters were also introduced in that context to quantify the complexity of an NLE's dominant subsystems, namely: Its dimension, the non-zero density of its Jacobian matrix, the nonlinearity ratio of these non-zero entries and the condition number of its Jacobian matrix. Theoretically, more than one dominant subsystem is possible. However, in the ones tested here, it was always limited to one subsystem with a significantly higher dimension. Now, using these parameters, the application range of the hybrid approach is to be defined in comparison to the sole IA-based solver and root-finding based algorithms, taking into account the results from the computational experiments. In addition to the four examples discussed in this thesis, the four other examples from Appendix B will be considered as well. In contrast to the condition number, the other three quantities can be determined independently of the initialization of the problem and therefore seem to be better suited to define the range of application. Figure 6.1 shows the nonlinearity ratio plotted over the dimension of the largest subsystem for all tested examples. In the Appendix in figure D.1 an analogous diagram is shown, which plots the non-zero density versus dimension of the largest subsystem. However, the nonlinearity ratio seems to contribute more decisively to whether a system can be solved or not by a certain solution strategy.

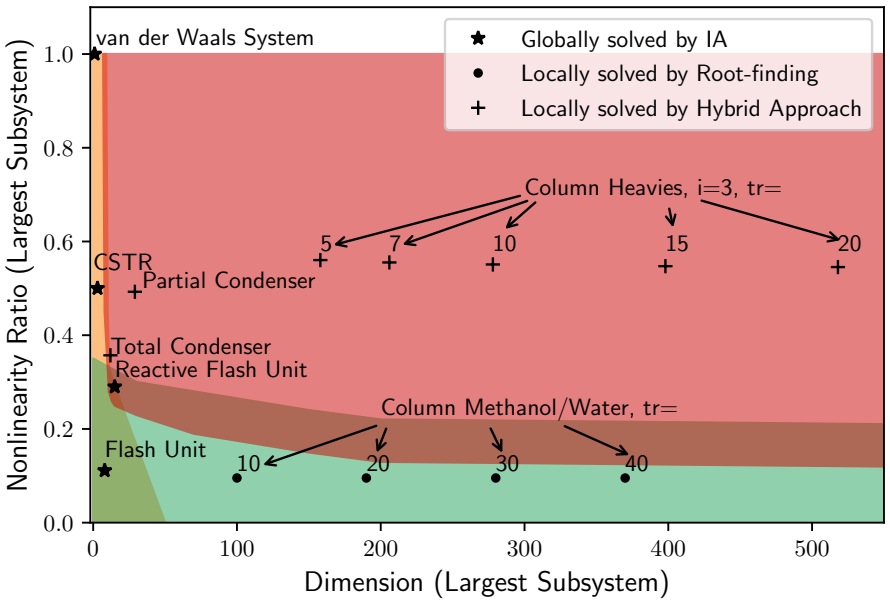


Fig. 6.1: Nonlinearity ratio versus dimension of largest subsystem for the tested examples and recommended application area for global solver (■), root-finding algorithm (■) and hybrid approach (■).

6.4 Global Solution

If the root-finding is turned off, the algorithm acts as a pure IA-based solver. If no wide infeasibilities are present in the initial box or have been successfully removed by reformulation, the IA-based solver seems to find all solutions up to a dimension of the largest subsystem of eight in the case of the tested examples in less than 15 s (covered by the yellow shaded area in figure 6.1). The variable intervals were not restricted at all, i.e., they had their default boundary values of $\pm 10^9$. With increasing dimension this appears to be possible only if some effort has been put into the initial variable bounds. Thus, the *Reactive Flash Unit* can be solved globally, but the IA-based solver needs 552 s, while a local search by IPOPT takes only 6 s to find the solution. Furthermore, clustering around a solution can occur as in the case of *Total Condenser* and lead to the fact that there is not only one solved box in the required tolerance, but very many, which again slows down the algorithm considerably.

6.5 Root-finding

For larger systems, it usually becomes more difficult or practically impossible to find all solutions due to clustering, excessive box splitting, and the increasing time required by an IA-based reduction step. State-of-the-art numerical solvers, on the other hand, find a solution quickly if the dominant subsystem of the process model is not so strongly nonlinear and if, as the dimension of the dominant subsystem increases, a little more effort is put into its initialization as in the case of *Column Methanol Water*. The initialization is included in table C.2 in the Appendix. When there is sole interest in finding one feasible solution, then the pure root-finding is superior to the IA-based procedure and the hybrid approach because of the speed. The larger the nonlinearity ratio, the more difficult the search for an initial point becomes, where a root-finding algorithm converges from. To some extent trust region or line search strategies can help, which have been introduced in section 2.3. However, in ill-conditioned systems the method might stagnate or converge to undesired solutions. In least-squares minimization problems, the solvers may also terminate at one of the numerous local minima. Hence, none of the root-finding algorithms manages to solve the *Total Condenser*, *Partial Condenser* and the *Heavies Column* without box reduction. According to the results, it is suggested that up to a nonlinearity ratio lower than that of the *Total Condenser*, which is around 0.36, and larger than that of the *Column Methanol Water*, which is around 0.1, pure root-finding algorithms in combination with a line search or trust region strategy are sufficient. It is assumed that with an increasing dimension of the largest subsystem, root-finding algorithms can only solve those with lower nonlinearity ratios. Hence their application area decreases with higher dimensions and the hybrid approach should be applied instead. An exact limit between both strategies can not be given at this point. To improve robustness of the numerical iteration, it turned out to be useful to decompose the overall system in general and to solve the subsystems sequentially. This also led to improved conditions in the subsystems as had already been shown in Bublitz et al. (2017a). Scaling the Newton algorithm with MC29 or MC77 in general leads to better convergence, while the unscaled Newton often fails.

6.6 Hybrid Approach

From figure 6.1 it can be seen that the hybrid approach has its merits in the area of high nonlinearity ratios and higher dimensions of the largest subsystem, especially when only one feasible solution is sought. Here the other two strategies take too long or fail, for example in the case of the *Total Condenser*, *Partial Condenser* and *Heavies Column*. For smaller dimensions, of course, it finds all solutions just like the IA-based solver. It turns out that the unique solution test does not save any time in the case studies since it was only satisfied by non-empty boxes close to the solution. To the contrary, the numerical iteration slowed down the hybrid approach compared to the pure IA-based solver. An advantage over the IA-based solver is, however, that a numerical solution is returned as soon as it has been found. This can happen well before the termination of the IA-based solver. In general, IA-based solvers and hybrid approach have the property of finding more than one solution and can prove that a box has only one solution by a successful unique solution test. Thus, they are quite superior to numerical solvers in this application area, although they require more time. Another observation is that the nonlinearity ratio hardly changes with increasing number of stages of the columns (it drops slightly), probably because this ratio is constant for the equations belonging to one stage. The small difference comes from the fading influence of the related equations to reboiler and condenser. Regarding the *Heavies Column*, the hybrid approach can find a solution for all tested numbers of stages without the need to further narrow down the variable bounds with increasing dimension.

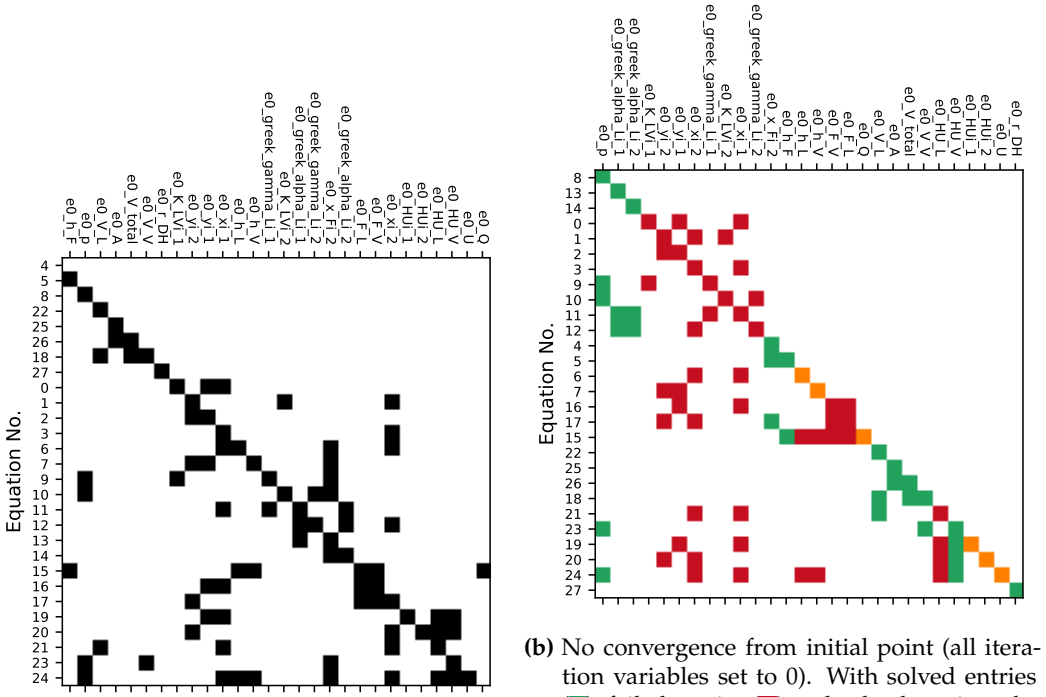
6.7 Debugging

To err is human. During modeling, errors often sneak into the formulation of equations or into the initialization of variables. In complex process models, the error source can be less evident and the search for the error is even more frustrating, because the error messages are usually not very specific. The best advice is to build up the equation system modularly in subsystems from the very start and to check them successively. This also allows for reuse of the modules in other

equation systems, e.g., for the calculation of phase equilibria. Assume that this modularization has already been done, and there remains a subsystem, which cannot be solved. There are now three possible types of error: a structural error in the equation system, an initialization error concerning the initial values or variable bounds, or an error in the solution algorithm. How to efficiently debug the solution algorithm is explained in Appendix D.3. Now, tools are presented, which were developed in the context of the hybrid approach, in order to be able to identify errors of the first two groups.

A **structural error** means that the equations of the system are linearly dependent and there are infinitely many or no solutions. This contradicts the requirement of the hybrid approach, which only works for well-determined NLEs. Some linear dependencies can be detected using the DM decomposition. Applying the UDLS PYTHON_NLE_ModOpt_1.0.moslsp (ID:106247) in MOSAICmodeling, Python code can be generated to solve a DM decomposed NLE directly using the Newton method. In contrast to the hybrid approach, the incidence of the Jacobian matrix is directly output graphically in addition to a separate analysis text file. Some structural errors become immediately apparent by zero entries on the diagonal of the matrix. For example by mistake, both molar fractions of the binary feed mixture of the *Flash Unit* might have been declared as design variables, while the temperature of the vessel is accidentally chosen to be iterated. Hence, the summation relation of the feed is overspecified while there are infinitely many solutions to the composition of the phase equilibrium, i.e., it is under-determined. The corresponding Jacobian matrix is shown in figure 6.2a. One recognizes immediately the incomplete diagonal. The numbering of the matrix rows corresponds to the global indices of the equations, via which one can directly determine the overspecificed summation relation that has the global index four in the considered system. The analysis text file contains a corresponding legend that helps to identify this critical equation and fix the model subsequently.

An **initialization error** produces either an abortion of the numerical iteration, e.g., because the Jacobian matrix is already singular at the initial point, or an abortion of the box reduction due to an empty initial box. An abortion of the numerical iteration is no tragedy, because the hybrid approach simply continues in this case. Nevertheless, it would be nice especially in complex systems that cannot be solved globally, if the numerical solver converges after the first box reduction



(a) Structural error.

(b) No convergence from initial point (all iteration variables set to 0). With solved entries (■), failed entries (■) and solved entries relying on wrong input from failed entries (■).

Fig. 6.2: Jacobian incidence matrices of *Flash Unit* used for error detection.

steps. A solver abortion can often be avoided by reconsidering the initial values or reformulating equations as it has been exemplified in sections 6.1 and 6.2. To identify the critical equations, which should be reformulated, again the UDLS PYTHON_NLE_ModOpt_1.0.moslsp (ID:106247) helps. It generates a colored Jacobian matrix that pictures the solver's success as shown in figure 6.2b. The solver processes the squared blocks along the diagonal. If such a block contains only green entries, the corresponding subsystem could be solved. Blocks with red entries could not be solved and those with yellow entries could be solved but they depend on red blocks, i.e., variables, which originate from the failed subsystems. In consequence, decomposition is used here to debug the model. The red blocks should be successively analyzed and reformulated if necessary. In the example in figure 6.2b the first subblock cannot be solved, because of a division by zero at the initial point. An abortion of the box reduction in general means that the variable bounds do not fit to each other. In this case, it would be helpful to localize the source more precisely. For this reason, the equation and variable that led to an

empty box are saved temporarily. If there are several boxes, the last box identified as empty is saved. The critical equation and variable, as well as the initial and final intervals of all other variables in the equation before termination, are written to a text file.

```
***** Error Analysis *****

Algorithm failed because it could not find any solution for e0_v_ph in equation:

 -e0_v + e0_v_ph + 0.000711

The following table shows the initial and final bounds of all variables in this equation before termination:

VARNAME      INITBOUNDS        FINALBOUNDS
e0_v_ph      [ 5. 10.]   [5, 10]
e0_v   [-1.e+09  1.e+09]       [0.0002374218623692886, 0.01678186000000001]
```

Fig. 6.3: Error text file to identify inconsistent variable intervals of *van der Waals' System*.

For example, let v_{ph} be incorrectly initialized to $[5, 10]$ in the *van der Waals' System*. The hybrid approach terminates after one reduction step with the error text file shown in figure 6.3. It is easy to see that the final intervals do not match in this case. In more complicated cases, the final bounds at least give a clue which variable interval seems to be strange. Possibly, another equation in which this variable appears, was set up incorrectly so that the expected variable intervals are infeasible. Hence, such equations should be carefully checked. If the error cannot be identified via this methodology, only debugging the algorithm will help.

6.8 Things that did not work out

During the development of the hybrid approach further features were tested. However, they could not contribute to its performance increase. Firstly, various preconditioning methods can be applied in the Interval Newton. The point of expansion $\boldsymbol{x}_c$ corresponds to the midpoint of the intervals in this work. Alternatively, a function called `condJ` was implemented, whereby, in addition to the midpoint, the point at the lower bounds of all variable intervals $\underline{\boldsymbol{x}}$ and the point at the upper bounds of all intervals $\overline{\boldsymbol{x}}$ can be considered. From the three points,

the one with the lowest condition number κ_2 is then selected

$$\boldsymbol{x}_c := \min\{\kappa_2\left(\left(\boldsymbol{J}(\underline{\boldsymbol{x}})\right), \kappa_2\left(\boldsymbol{J}(m(\boldsymbol{x}))\right), \kappa_2\left(\boldsymbol{J}(\overline{\boldsymbol{x}})\right)\right\} \quad . \tag{6.14}$$

However, in all tested cases, the midpoint was always chosen and the evaluation of κ_2 at the three points only slowed down the process. Secondly, there are three more options to precondition the Gauss-Seidel step. The real-valued preconditioning matrix $\boldsymbol{P}$, is defined for the three cases as follows

- `inverseCentered`: $\boldsymbol{P} := m(\overline{\boldsymbol{J}})^{-1}$
- `inversePoint`: $\boldsymbol{P} := \boldsymbol{J}(\boldsymbol{x_c})^{-1}$
- `inverseDiagonal`: $p_{j,i} := \begin{cases} 0 & \text{if } j \neq i \\ m(\bar{j}_{j,i})^{-1} & \text{if } j = i \end{cases}$

The entries $p_{j,i}$ of $\boldsymbol{P}$ are used to calculate the preconditioned Jacobian matrix and vector of functions' residuals according to Eq. 3.3 and 3.4. They are in turn inserted into the Gauss-Seidel step according to Eq. 2.96. The method `inverseCentered` has already been discussed in section 2.6.2. The method `inversePoint` seems to be useful in combination with `condJ` as the best conditioned Jacobian matrix out of the three is employed for scaling. Nevertheless, this method can be combined with `center` as well. The preconditioner `inverseDiagonal` does not depend on the choice of $\boldsymbol{x}_c$. It is a diagonal matrix, which simply takes the inverse of the interval midpoints on the diagonal of the Jacobian matrix. This method is the least costly one out of the four. However, all three preconditioning methods resulted in irregular matrices, especially in the first box reduction steps, which in turn caused the variable intervals not to be reduced. Possibly, an increase in efficiency only occurs when the interval bounds are already close to the solution. However, such an investigation was not carried out within the scope of this work.

Next, affine arithmetic using the Python package affapy extends the function evaluation by the classical, so-called natural interval arithmetic. Affine arithmetic can partly tackle the interval dependency issue as it transforms a classical interval

$\overline{\underline{x}}$ into its affine form

$$\overline{\underline{x}} := m(\overline{\underline{x}}) + \epsilon_1 \cdot 0.5 \cdot w(\overline{\underline{x}}), \qquad \epsilon_1 = [-1, 1] \quad . \tag{6.15}$$

For example the mathematical expression

$$f(x) = x^2 - x \tag{6.16}$$

evaluated by natural IA in the interval $\overline{\underline{x}} = [0, 1]$ equals $\overline{\underline{f}}(\overline{\underline{x}}) = [-1, 1]$. Using its affine form

$$\overline{\underline{f}}(\overline{\underline{x}}) := m(\overline{\underline{x}})^2 - m(\overline{\underline{x}}) + \epsilon_1 \cdot 0.5 \cdot w(\overline{\underline{x}}) \cdot (2 \cdot m(\overline{\underline{x}}) - 1) + \epsilon_1 \cdot 0.25 \cdot w(\overline{\underline{x}})^2 \tag{6.17}$$

the interval can be further tightened to $\overline{\underline{f}}(\overline{\underline{x}}) = [-0.5, 0.0]$. However, the evaluation of functions by affapy slowed down the whole procedure very much. Tighter bounds in one reduction step could be achieved but in the same time the boxes could be reduced much further by contraction, cutting, and splitting only. In figure D.2 in the Appendix this is illustrated for the box reduction of the *Flash Unit*.

Another functionality tested has been a set of sampling methods with and without a multi-start procedure. According to this, the reduced, feasible regions of the DM-decomposed subsystems can be well mapped via sampling methods. In the scope of this work samples generated by the latin hypercube method (McKay et al., 1979), the low discrepancy sequences of Hammersley (Hammersley and Handscomb, 1964), and Sobol (Sobol, 1967) have been tested as well as the ***C**ovariance **M**atrix **A**daption **E**volution **S**trategy* (CMAES) (Hansen, 2016) from optuna which is a Python package. The sample or several samples with the lowest functional residuals of the NLE are subsequently used as initial point(s) for the numerical iteration. In general, the solutions could also be found via sampling, but only if this was also possible via the midpoint. However, the computation of the midpoint is much faster, which is why no merit in sampling could be observed so far. The idea was that especially in large and complex systems the chance is not very high that the midpoint of a box is a converging initial point and by sampling one or even several better suited points can be found to increase the robustness. But the space that needs to be sampled in the subsystem of the *Heavies Column* for exam-

ple is much too large to get close to a solution via this approach. Generating 5000 samples by optuna's CMAES took more than 80 min. One idea to increase the efficiency of the sampling methods was to use the BBTF matrix decomposition instead of DM (see section 2.4.2) and to resolve only the relatively low-dimensional subsystem of tearing variables via the sampling methods with a high number of samples. The midpoint was still used for the other subsystems. However, the numerical iteration was always unstable despite scaling of all subsystems.

Finally, in the beginning not only parallelized contraction of the boxes but also of the variables was tested. However, the box can be reduced less within one reduction step, since already reduced variable intervals are not updated before the reduction of the upcoming variable intervals during one contraction step. The program was tested with eight process cores at maximum. If in the future more cores are easily available this option could become relevant again, because the time reduction of one reduction step might pay off the further required reduction steps. In Bublitz et al. (2021b) was already shown for Bnormal, which operations within its algorithm can be parallelized efficiently.

6.9 Conclusion

In this work, a method was sought to bridge the gap between increasingly complex process models and their solvability using conventional numerical methods. In particular, this involved a generally applicable initialization strategy for NLEs. For this purpose, the combination of IA-based and root-finding algorithms was considered promising. The IA-based algorithm can discard infeasible points or regions, where model equations or their derivatives are not defined, and thus, pave the way for a fast root-finding iteration. This so-called hybrid approach was implemented as part of a custom package named modOpt in Python. A UDLS was created in MOSAICmodeling to apply the hybrid approach to NLEs from MOSAICmodeling, where the test cases were implemented.

In the first version, the hybrid approach only included the conventional IA-based contraction methods: Interval Newton and HC4revise in combination with the state-of-the-art root-finding algorithms: Scipy's Fsolve and SLSQP as well as

IPOPT and a self-implemented Newton, according to steps one to three of the hybrid approach. Since the reduction of the interval bounds was insufficient in the VLE calculations, the method Bnormal as well as the additional feature tighten_bounds were developed. Their scope is limited to equations with multiple instances of one or more variables, in which interval dependency occurs, e.g., in the calculation of fugacity or activity coefficients to reduce their intervals further. In addition, Bnormal is able to directly remove infeasible values from the variable domain and thus, clears the way for root-finding algorithms. Since contraction-based IA methods did not always lead to the desired physical solution when the NLE had more than one solution, the hybrid approach was extended by splitting (step five of the hybrid approach). Independent of the variables selected for splitting, it significantly increased the number of boxes and CPU time in complex systems with a dimension of eight and higher. To avoid extreme box splitting, further functionalities were developed such as cutting (step four of the hybrid approach), a linearly increasing restriction of the maximum number of boxes in the ongoing process, and the forecast split strategy. In the latter, splits of different variable intervals are considered and exactly that split is carried out, which can subsequently reduce the two subboxes the most by contraction. For cutting and splitting, all variables or only the tearing variables can be chosen. Moreover, for splitting, those variables can be selected whose intervals have changed the least compared to the last split or to their initial bounds, if no split has occurred yet. To speed up the iteration process in the presence of several boxes, the box reduction was parallelised as well. Thus, the hybrid approach achieves the initially set research goals as follows:

- Infeasible points or regions are filtered out by means of Bnormal
- Initial box is efficiently reduced through the combination of the contraction methods and the development of new methods to avoid extreme box splitting
- Recurrent root-finding steps are an inherent part of the hybrid approach, in which the numerical solver is started from the midpoint of the current box

On this basis, the initially posed research questions will be finally answered now:

RQ1. How well does the hybrid approach perform in locating a desired solution of an NLE compared to state-of-the-art root-finding algorithms?
For some of the tested NLEs, especially those with a large complex subsystem, a high condition number and/or a high nonlinearity ratio, the state-of-the-art numerical solvers could not find any solution. However, at most three box reduction steps were necessary in the hybrid approach to achieve convergence. Yet, there were also a few other NLEs, which could be solved directly by the root-finding algorithms in less time than the hybrid approach due to the absence of the more time-consuming box reduction.

RQ2. How well does the hybrid approach perform to solve an NLE globally compared to a pure IA-based solver?
The expectation that a successful unique solution test in combination with a numerical solution, previously found in a root-finding step, could terminate the IA-based iteration earlier, was not fulfilled. The unique solution test occurred, if at all, only shortly before a box was classified as solved. The additional root-finding steps of the hybrid approach even increased the CPU time compared to the sole IA-based solver.

RQ3. How can the complexity of an NLE be measured?
The non-zero density, nonlinearity ratio and condition number of the Jacobian matrix as well as the systems' dimension were considered to quantify the complexity of an NLE and determined for the test examples in section 4.1. These parameters were only calculated for their largest complex subsystem, since it had the greatest influence on the success of the numerical solvers. Theoretically, an NLE could contain more than one of such complex subsystems. In the test cases examined, however, one always dominated. Hence, an NLE is considered to be more complex the higher the respective parameter values of its dominating subsystem turn out to be.

RQ4. Which equation formulations are useful to ensure an efficient numerical iteration?
Section 6.1 summarizes important formulations of equations as well as reformulations of functional terms that have emerged from the literature review and our own computational experiments. In short, one should minimize the number of variable instances in an equation to prevent interval dependency. In addition, functional terms should be avoided, whose variables allow an

infinite number of combinations of real values in their intervals, where those terms are not defined.

RQ5. Can structural properties of an NLE be used to conclude, which numerical method is appropriate to solve it?
According to figure 6.1, dimension and nonlinearity ratio of the dominating complex subsystem seem to be well suited to distinguish the application areas of the three solution strategies: IA-based solver, root-finding algorithm and hybrid approach. The hybrid approach is particularly effective for systems with a dimension greater or equal to 12 and a nonlinearity ratio of 0.36 and higher. They could not be solved by the other two methods. Systems with lower dimensions could also be solved globally in short time (less than 20 s). Systems with a low nonlinearity ratio in turn, could be solved by root-finding algorithms in less time than the hybrid approach needed. However, these limits are only a rough estimate and seem to depend on the condition of the system as well. For systems with higher condition numbers, the application areas of IA-based solver and root-finding algorithms are expected to decrease in favor of the hybrid approach.

Finally, tools for a quick identification of error sources in failed numerical iterations have been implemented in the scope of this work and are presented in section 6.7.

6.10 Outlook

Beyond the examples studied in this thesis, it would be interesting to apply the hybrid approach to even more complex problems. For this purpose, the existing HDA process of Rajes (2020) is a good starting point, e.g., to take into account challenging recycle streams in the model. Furthermore, NLEs which include heat integration and process intensification would be interesting to examine. While this work has focused on steady-state process models, it would also be interesting to extend the application of the hybrid approach to other areas, where NLE solvers are commonly in use, for instance to initialize and solve ***D**ifferential **A**lgebraic **E**quation systems* (DAEs) and ***P**artial **D**ifferential **A**lgebraic **E**quation systems* (PDAEs).

For this purpose, such systems only need to be transformed into algebraic NLPs by finite differences or orthogonal collocation as described in Hangos & Cameron (2001, pp. 191-222). In the long term, it might be beneficial to build an open access library of process engineering problems, or further extend existing libraries such as the COCONUT benchmarks created by Shcherbina et al. (2003). This saves time to model processes and makes a comparison of newly developed approaches easier. The performance of the hybrid approach's box reduction strongly depends on the initialization of the variable intervals and the formulation of the equations. In the field of chemical process models, the intervals could be set automatically according to simple heuristics analogous to those summarized in Appendix D.2. Equations could be automatically reformulated, whenever critical terms such as the ones from section 6.1 are present. In addition, the research should continue to identify further favorable formulations for IA. The hybrid approach itself could be improved by speeding it up, increasing the efficiency of a box reduction step or improving the root-finding step through the development of a suitable sampling strategy. As shown in the tests, a trade-off between speed and success of a reduction step usually appears. In particular tighten_bounds and the affine arithmetic could not convince here so far due to the long processing time. However, since the package affapy behind the affine arithmetic was first released in May 2020, a performance increase can possibly be expected here in the near future. In the case of tighten_bounds, the current implementation would have to be analyzed more closely in order to accelerate this methodology, or its calls within the program would have to be reduced. Cutting and splitting contribute enormously to box reduction, but the investigations of this work do not show any tendency, which strategy, i.e., variable selection, leads to the highest box reduction in the least amount of time. A more precise analysis is needed to identify promising splits and cuts without applying these methods to all variables each time. Another option to increase the speed of the hybrid approach is to further parallelize processes within the program based on some ideas given in Bublitz et al. (2021b). Finally, for sampling strategies, there are many new developments due to the high interest in data-driven modeling that can be easily linked to the hybrid approach. The same is true for new numerical solution methods. How the latter two can be connected to the hybrid approach is explained in modOpt's git project[1].

[1] https://git.tu-berlin.de/dbta/simulation/modOpt

Appendix A

Algorithms, Scripts and Software

A.1 Bubble point method

A well-known algorithm to solve the cascade of equilibrium trays in a distillation column model, which is illustrated in figure A.1, is the Wang Henke's bubble point method (Wang and Henke, 1966). In the column model, the top tray represents the condenser, where the side stream $F_{U,tr=1}$ corresponds to a potential distillate rate, $F_{L,tr=1}$ to the reflux rate and $Q_{tr=1}$ to the cooling duty. The lowest tray n corresponds to the reboiler with heat duty $Q_{tr=n}$. The bubble point algorithm is schematically shown in figure A.2. First of all, the design variables have to be specified, and the tear variables have to be initialized, which are the tray's temperatures T_{tr} and light phase flow rates $F_{V,tr}$. Furthermore, initial values for the equilibrium constants $K_{tr,c}$ have to be specified, e.g. as a function of the initial temperature guesses by assuming an ideal phase equilibrium. Next, the mole fractions of the heavy phase $x_{L,tr,c}$ at the current values of T_{tr}, $F_{V,tr}$ and $K_{tr,c}$ can be calculated via the component balances ranging from the top of the column to the current tray section. $F_{L,tr}$ and $x_{V,tr,c}$ are previously eliminated via the total mass balances and the general chemical equilibrium

$$F_{L,tr} = F_{V,tr+1} + \sum_{j=1}^{tr} F_{F,j} - (F_{W,j} + F_{U,j}) - F_{V,tr=1} \tag{A.1}$$

$$x_{V,tr,c} = K_{tr,c} \cdot x_{L,tr,c} \quad , \tag{A.2}$$

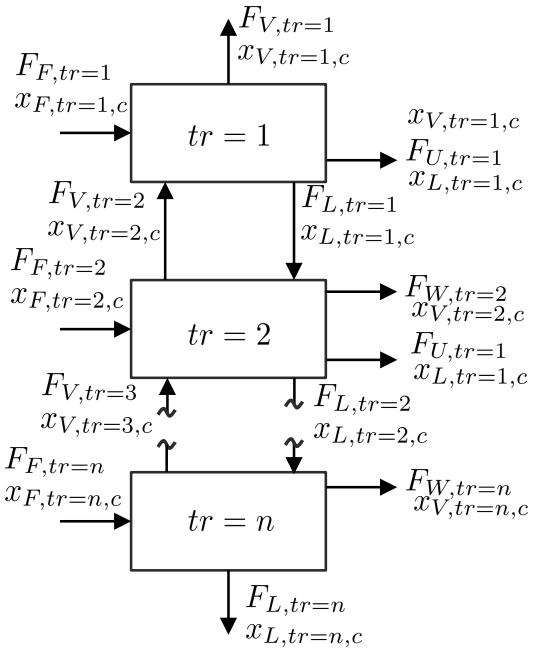


Fig. A.1: Cascade of equilibrium trays of a column model including streams F and mole fractions x.

so that the following linear, tridiagonal equation system results

$$
\begin{aligned}
0 = & \overbrace{\left(F_{V,tr} + \sum_{j=1}^{tr-1} F_{F,j} - (F_{W,j} + F_{U,j}) - F_{V,tr=1}\right)}^{:=A_{tr}} \cdot x_{L,tr-1,c} \\
& \overbrace{- (F_{V,tr+1} + \sum_{j=1}^{tr} F_{F,j} - (F_{W,j} + F_{U,j}) - F_{V,tr=1}}^{:=B_{tr,c}} \\
& + F_{U,tr} - (F_{W,tr} + F_{V,tr}) \cdot K_{tr,c}) \qquad \cdot x_{L,tr,c} \\
& + \overbrace{F_{V,tr+1} \cdot K_{tr+1,c}}^{:=C_{tr,c}} \cdot x_{L,tr+1,c} + \overbrace{F_{F,tr} \cdot x_{F,tr,c}}^{:=-D_{tr,c}} \quad .
\end{aligned}
\tag{A.3}
$$

This can be solved efficiently using the Thomas algorithm (Thomas, 1949). Here $A_{tr=1}$ and $C_{tr=n,c}$ are zero at all times. One system of this type is solved per component c. The calculated $x_{L,tr,c}$ may violate the summation relation, so they are normalized in a next step according to Figure A.2. Subsequently, the temperatures

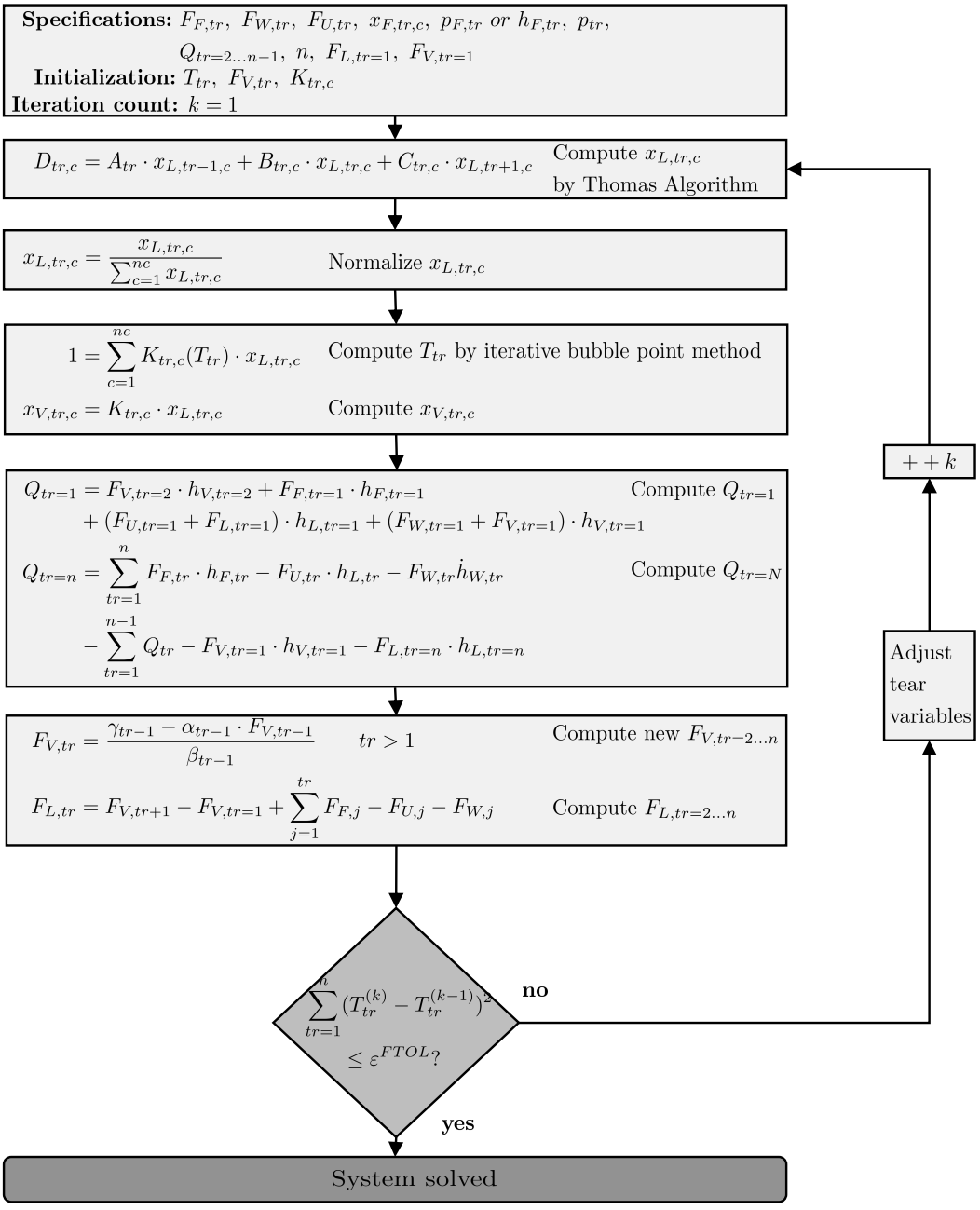


Fig. A.2: Schematic diagram of Wang Henke's bubble point method for solving a cascade of equilibrium trays (Wang and Henke, 1966).

are iteratively determined via the bubble point equations so that the mole fractions of the light phase $x_{V,tr,c}$ are then obtained from the chemical equilibrium. Now, the cooling and heat duties of the condenser and reboiler can be calculated. Since temperatures, pressures and compositions in all streams are known, the specific enthalpies h can be determined explicitly and so can $Q_{tr=1}$ and $Q_{tr=N}$. Next, the flows of the light phase $F_{V,tr}$ are obtained through the heat balances from the top of the column to the current tray section. These equations can be transformed into a sparse linear system with the coefficients α_{tr}, β_{tr}, $gamma_{tr}$

$$0 = \alpha_{tr} \cdot F_{V,tr} + \beta_{tr} \cdot F_{V,tr+1} - \gamma_{tr} \tag{A.4}$$

$$\alpha_{tr} = h_{V,tr} - h_{L,tr-1} \tag{A.5}$$

$$\beta_{tr} = h_{V,tr+1} - h_{L,tr} \tag{A.6}$$

$$\gamma_{tr} = (\sum_{j=1}^{tr-1} F_{F,j} - F_{U,j} - F_{W,j} - F_{V,tr=1}) \cdot (h_{L,tr} - h_{L,tr-1}) \tag{A.7}$$

$$+ F_{F,tr} \cdot (h_{L,tr} - h_{F,tr}) + F_{W,tr} \cdot (h_{V,tr} - h_{L,tr}) + Q_{tr} \quad . \tag{A.8}$$

The unkown $F_{V,tr}$ can be explicitly calculated one after the other according to the formula shown in Figure A.2, starting with the known flow rate $F_{V,tr=1}$. At the end of the calculation sequence, the flow rates of the heavy phase $F_{L,tr}$ are determined by the total mass balances represented in Eq. A.1. Finally, the temperatures of the current iteration step k are compared to those of the previous one. If they differ according to the tolerance ε^{FTOL}, the tear variables are adjusted and the iteration continues with step $k+1$, otherwise the system is solved. To adjust the tear variables, damping strategies are often necessary to keep them within a physically reasonable range. The bubble point algorithm is mainly used for column simulations in which narrow-boiling mixtures are separated, since here $K_{tr,c}$ changes preferably with the temperature than with the concentrations (Friday and Smith, 1964). If this is not the case, small differences in concentration can already lead to large changes in temperature and the algorithm is prone to oscillating tear variable values, which may even result in divergence. Hence, convergence is by no means guaranteed in this method.

A.2 Additional Algorithms of Hybrid Approach

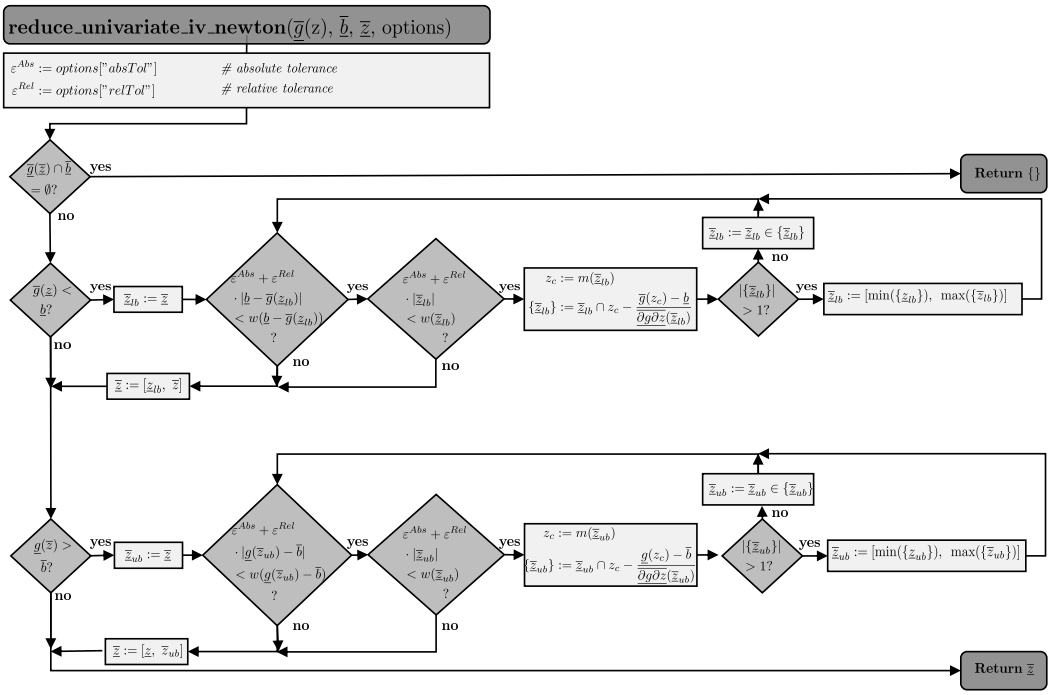


Fig. A.3: The algorithm of the univariate interval newton for monotone increasing functions.

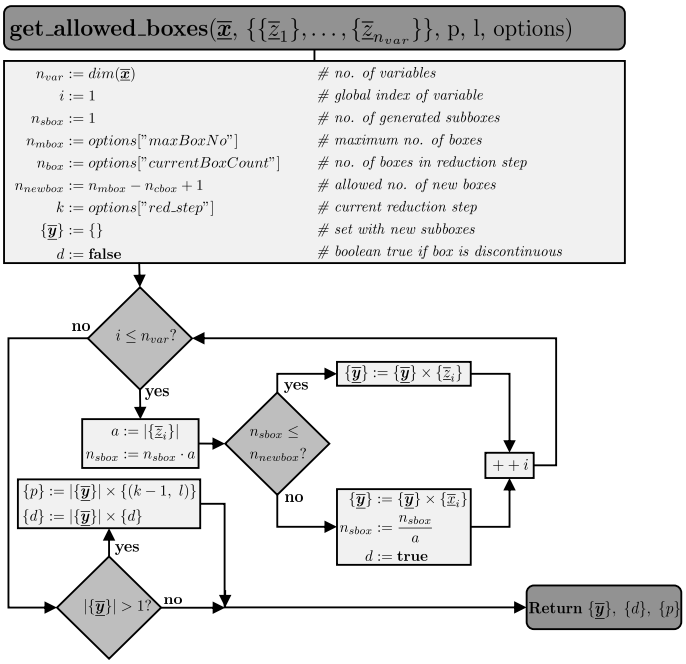
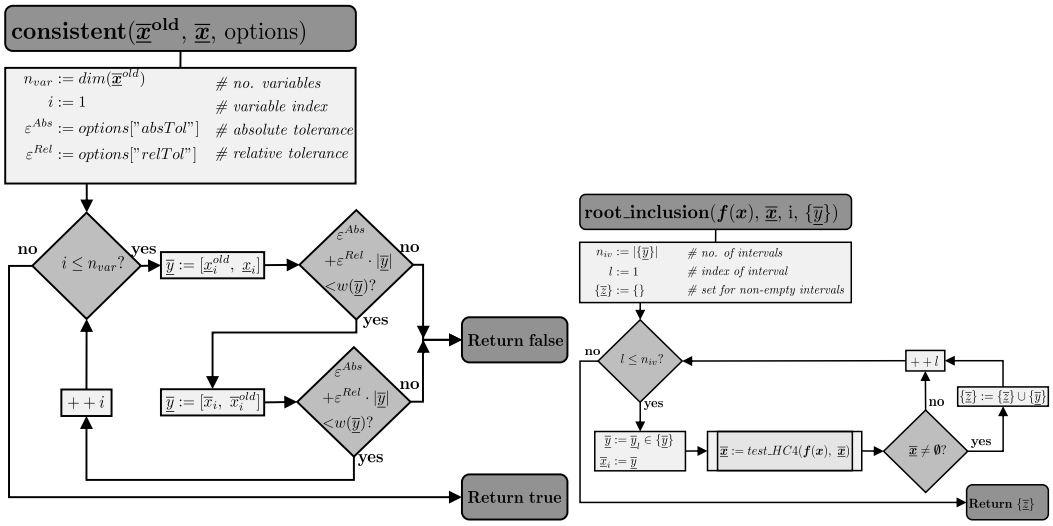
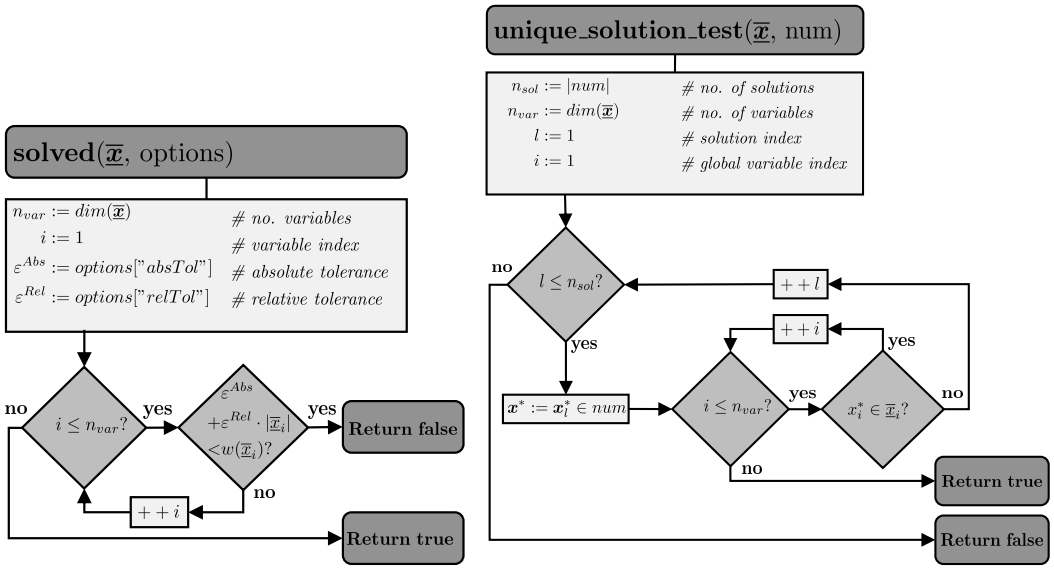


Fig. A.4: The algorithm of get_allowed_boxes.



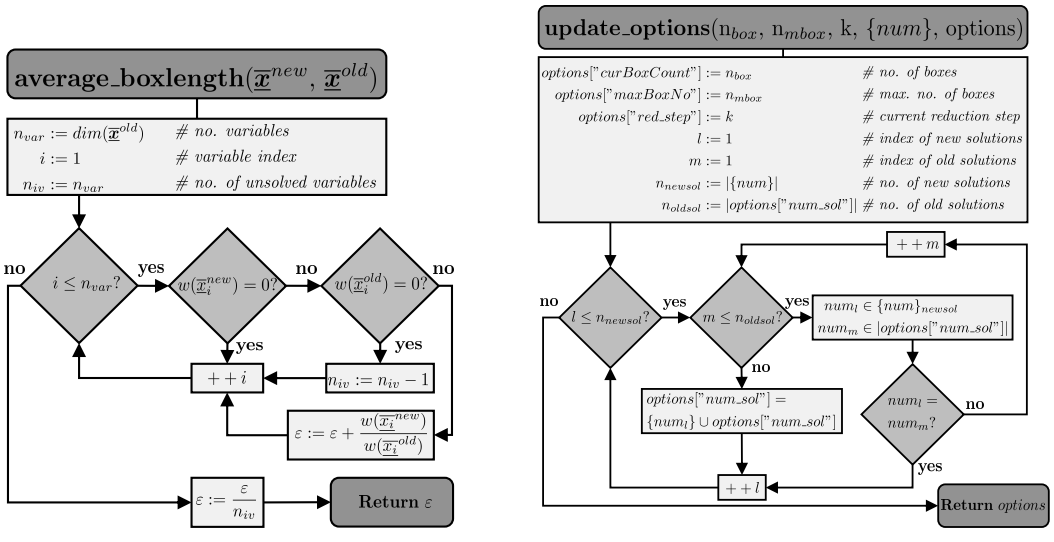
(a) consistent. **(b)** root_inclusion.

Fig. A.5: The algorithms of consistent and root_inclusion.



(a) solved. **(b)** unique_solution_test.

Fig. A.6: The algorithms of solved and unique_soluton_test.



(a) average_boxlength. **(b)** update_options.

Fig. A.7: The algorithms of average_boxlength and update_options.

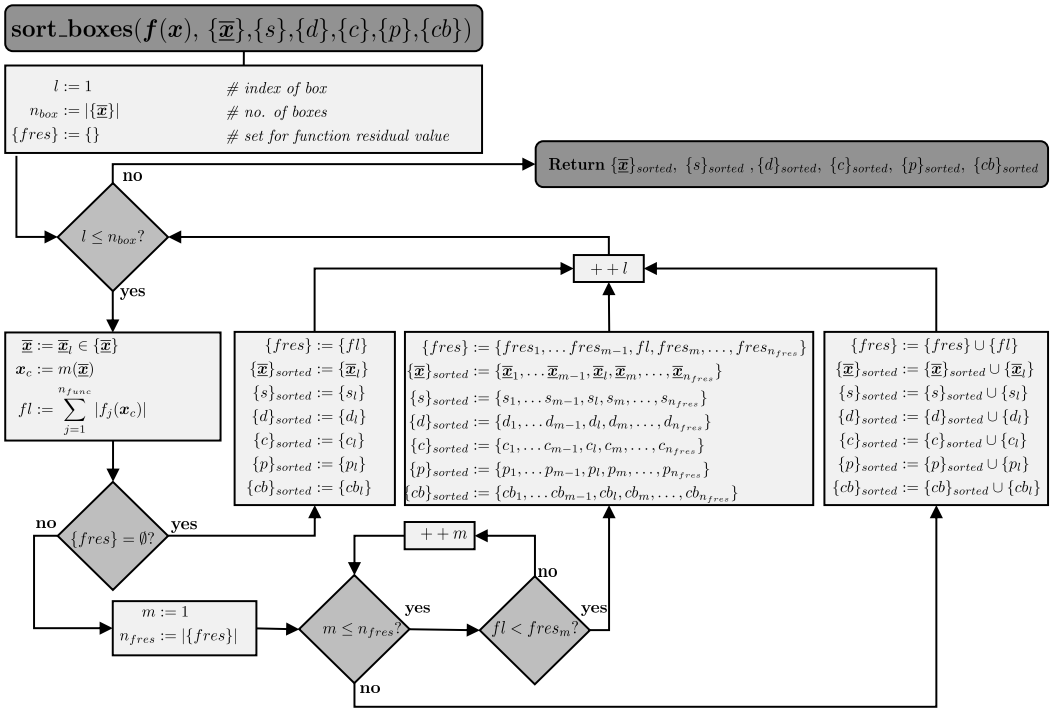
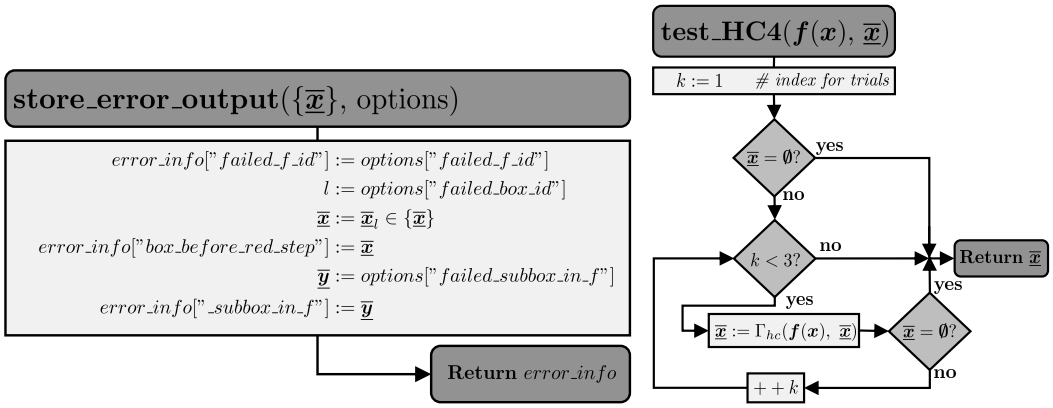


Fig. A.8: The algorithm of sort_boxes.



(a) store_error_output. **(b)** test_HC4.

Fig. A.9: The algorithms of store_error_output and test_HC4.

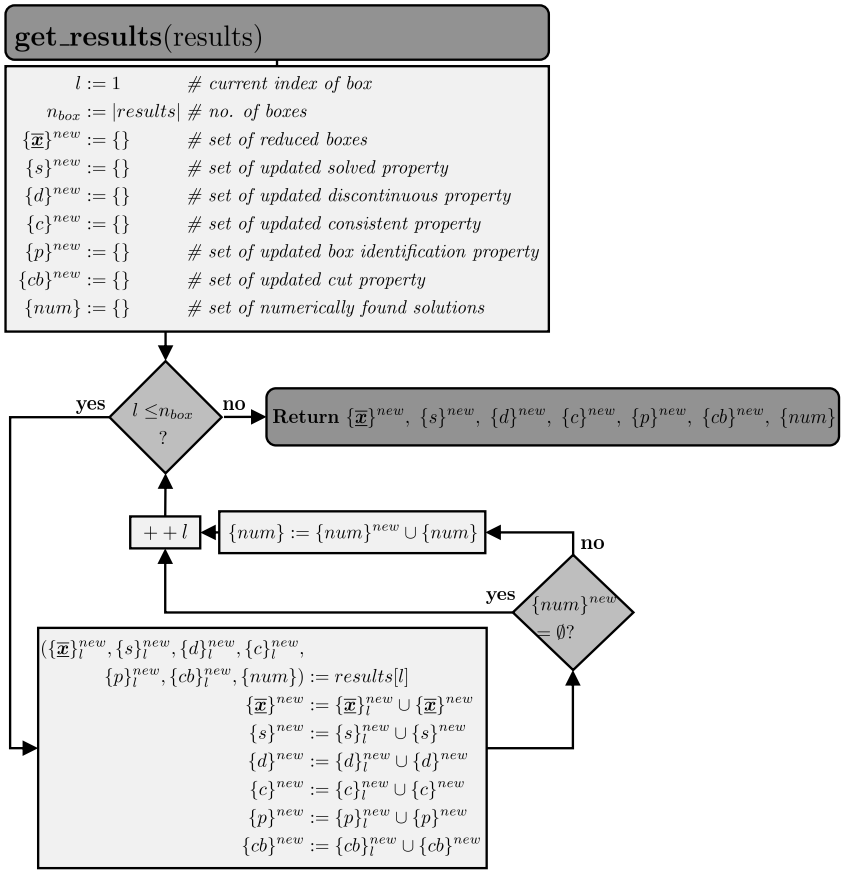


Fig. A.10: The algorithm of get_results.

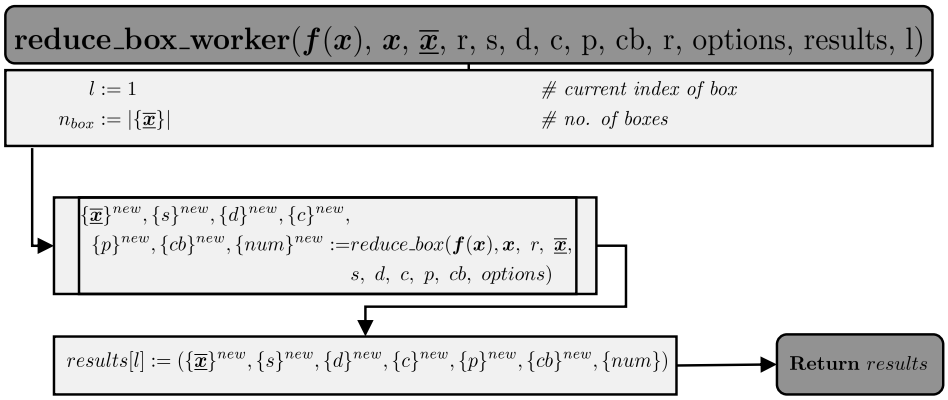


Fig. A.11: The algorithm of reduce_box_worker.

A.3 Python Scripts and Settings

A.3.1 Hyperparameters

The hybrid approach depends on many hyperparameters that are all set in a Python dictionary. Table A.1 contains all hyperparameters that have to be set by the user and the recommended default settings. In the following, recommendations are given on which parameters should deviate from the default values when a new system is to be solved. We focus only on the case, where a solution is to be found as quickly as possible. Recommended settings in dictionary **bxrd_options** of the

- Tolerances: textbfabsTol should be at least one order of magnitude lower than the order of magnitude of the smallest expected value of the solution. For **relTol**, a value one to five orders of magnitude higher than **absTol** is recommended.

- Resolution: The integer value should be at least 2. A maximum value of 128 was tested in the context of this work. According to the eight test examples, the value of **resolution** should rather be set in the lower range, a value between 4 and 32 is recommended with method tighten_bounds and 8 to 64 without so that the time of a box reduction step is not dominated by the refinement. To increase the reduction success of a box reduction step, a higher value is recommended, while a lower value decreases the process time of a step.

- Maximum number of box reduction steps: For a completely new system, it is suggested to leave **redStepMax** at a value of 1 and to make the first box reduction step as reductive as possible without paying too much attention to the CPU time. This means that all contraction methods including **tightBounds** should be selected and a high **resolution** set. In this way, the hybrid approach may already find a solution after one step. If the step takes too long, which depends on the dimension and complexity of the NLE, then the **resolution** should be successively lowered and **tightBounds** should be turned off. If no solution can be found in this way, **redStepMax** should be

set to a value between 3 and 5 next. Cutting and splitting are set according to the advised settings, so that within this iteration at least cutting and possibly splitting are performed. If there is still no solution found, **redStepMax** should be increased further. It is difficult to predict how long the algorithm will take, since the number of boxes can increase significantly as the program proceeds and hence, the time of one box reduction step raises. However, with the following worst-case scenario, one can roughly overestimate an adequate value for **redStepMax** based on a maximum permitted CPU time Δt_{max} and an averaged time for one box reduction step $\bar{t}$ determined from the run with **redStepMax** set to a value between 3 and 5 (because it includes cutting in this case). The heuristic formula is

$$\textbf{redStepMax} = -1.5 + \sqrt{4.5 + \frac{6 \cdot \Delta t_{max}}{\bar{t}}} \quad . \tag{A.9}$$

The formula is based on the assumption that after every third box reduction step one box splits into two and the number of boxes increases accordingly. Furthermore, none of the boxes is identified as empty.

– Cutting and splitting strategies: In order to test whether the solution can already be found in the first box reduction steps, it should be started with settings that allow for a maximum box reduction in one step. For this purpose, **cutBox** is set to "all" and **splitBox** to "forecastSplit". If the runtime of the program is already intractable, less time-consuming options can be chosen, which in turn are accompanied by a loss in the box reduction per step. **cutBox** should in this case be set to "tear" and **splitBox** should be set to "leastChanged", "forecastTear" and "tearVar" in that order, when the previous choice still takes too long.

Recommended settings in dictionary **num_options** of the

– Root-finding based solver: If "casadi-ipopt" is installed on the computer, it is recommended to use this **solver**, else one of the other solvers from table A.1 can be tried.

– Solving mode: In order to terminate the hybrid approach after one solution has been found, **termination** must be set to "one_solution".

Tab. A.1: Dictionary keys of moOpt and default values.

Dictionary	Key	Value and Description
bxrd_options	**fileName**	String, default: "example" naming of output files
	savePath	String, default: "./results" path to location for storage
	redStepMax	Integer, default: 1 max. number of box reduction steps
	maxBoxNo	Integer, default: 1 initial restriction to box number per reduction step
	absTol	Float, default: 10^{-8} absolute tolerance value, see section 3.2
	relTol	Float, default: 10^{-3} relative tolerance value, see section 3.2
	resolution	Integer, default: 8 resolution of refinements, see section 3.1
	parallelBoxes	Boolean, default: False True for parallel box processing, see section 3.8

Dictionary	Key	Value and Description
	cpuCountBoxes	Integer, default: 2 number of processors used in parallelization
	affineArithmetic	Boolean, default: False True for application of affine arithmetic, see section 6.8
	hcMethod	String, default: "HC4" choose hull consistency method, if set to "None" it is not applied
	newtonMethod	String, default: "newton" choose interval newton method, if set to "None" it is not applied
	newtonPoint	String, default: "center" point of expansion, see section 6.8 alternative values: "condJ"
	preconditioning	String, default: "pivotAll" preconditioning for Interval Newton, see section 6.8 alternative values: "inverseCentered", "inversePoint", "inverseDiagonal"

Dictionary	Key	Value and Description
	bcMethod	String, default: "bnormal" choose box consistency method, if set to "None" it is not applied
	tightBounds	Boolean, default: False True for application of tight_bounds
	cutBox	String, default: "tear" variables to be cut, see section 3.4 alternative values: "all", "None"
	splitBox	String, default: "tearVar" variables to be split, see section 3.5 alternative values: "leastChanged", "forecastSplit","forecastTear"
	considerDisconti	Boolean, default: False True for splitting discontinuous boxes first
	hybridApproach	Boolean, default: True False for IA-based solver
	analysis	Boolean, default: True False for no calculation of analysis parameters, see section 3.5

Dictionary	Key	Value and Description
	timer	Boolean, default: True False for no record of CPU time
	debugMode	Boolean, default: False True for printing out intermediate results when the program is running
smpl_options	**smplNo**	Integer, default: 0 number of samples, 0 for mid point, -1 for user-specific initial values
	smplBest	Integer, default: 1 number of best samples in root-finding
	smplMethod	String, default: "sobol" only active if smplNo $\neq 0$ alternative values: "hammersley", "latin_hypercube", "optuna"
num_options	**solver**	String, default: "newton" solver applied used root-finding alternative values: "SLSQP", "fsolve", "casadi-ipopt", "matlab-fsolve-mscript"

Dictionary	Key	Value and Description
	mode	Integer, default: 1 only active in minimizer 1 least square minimization 2 equality constraints
	termination	String, default: "all_solutions" application as global or local solver alternative values: "one_solution"
	FTOL	Float, default: 10^{-10} function tolerance of numerical solvers
	iterMax	Integer, default: 100 number of maximum iteration steps
	scaling	String, default: "None" scales Newton solver, see section 3.3 alternative values: "MC29", "MC77"
	scalingProcedure	String, default: "block_iter" active in scaled Newton solver scales block wise at each iteration step alternative values: "block_init", "tot_init", "tot_iter" scales block wise (block) at initial point (init) or iteration step (iter)

A.3.2 Storage of Reduced Boxes in Python

The boxes from all reduction steps are stored in an npz-file, which is a zipped archive from Python 's package numpy . By calling the method get_parent_box that requires the tuple p and the path to the npz-file npz the parent box of $\overline{\underline{x}}$ is reloaded from this archive.

A.3.3 Processing Multiple Boxes in Parallel

To reduce the memory utilization, multiprocessing works with pickling of objects, i.e., converting their structure and values into a byte stream. Output arguments can be stored and accessed in a multiprocessing specific dictionary object. Hence, the wrapper function reduce_box_worker shown in Fig. A.11 has the task to write the set of reduced boxes and their properties into the so-called results dictionary for each box reduction. The method get_results then converts all elements of the results dictionary back to sets used in solve_NLE. The parallelization does not work on windows operating systems yet due to an internal pickling error of multiprocessing in reduce_box. The program runs error-free on Linux and Unix OS. However, the sequential and the parallel version of reduce_boxes are both part of the algorithm solve_NLE and are individually chosen by setting the parameter "parallelBoxes".

A.3.4 Use of Matlab's fsolve in the Hybrid Approach

To invoke Matlab's fsolve in the hybrid approach, Matlab needs to be installed and executable. Beside this, the Python package matlab must be installed so that Python can start the Matlab engine and receive the results accordingly during the running program. The iteration will be faster, when a Matlab script containing the complete NLE is directly used. This m-File can automatically be generated from the evaluation in MOSAICmodeling using the UDLS modOpt_solver_matlab_Vo (ID: 147665). Matlab's fsolve will invoke this file in an iteration whenever "solver" is set to `matlab-fsolve-mscript`. An alternative, slower way is to set "solver" to `matlab-fsolve`, in which case the symbolic functions are directly converted from sympy's to Matlab's syntax.

Code A.1: Testing mpmath's exp() and ln() function.

```
import mpmath
from time import time

def main():
    x = mpmath.mpi(0.1, 10.0)
    nl = int(1e6)
    aux = mpmath.iv.exp(x)

    print("Testing exp() function:")
    loop_and_track_function(mpmath.iv.exp, x, nl)

    print("Testing ln() function:")
    loop_and_track_function(mpmath.iv.ln, aux, nl)

def loop_and_track_function(fun, x, nl):
    t1 = time()
    for l in range(nl): fun(x)
    t2 = time()
    print('The Interval value is\n%s.' % str(fun(x)))
    print('Elapsed time is %f seconds.\n' % (t2-t1))

if __name__ == "__main__": main()
```

Output:

```
Testing exp() function:
The Interval value is
[1.1051709180756474904, 22026.465794806717895] .
Elapsed time is 11.372387 seconds.

Testing ln() function:
The Interval value is
[0.099999999999999866773, 10.000000000000001776] .
Elapsed time is 10.493530 seconds.
```

A.4 Software Packages

Tab. A.2: List of all used Python packages.

Package name	Version	Reference
`affapy`	0.1	https://pypi.org/project/affapy
`casadi`	3.5.1	https://web.casadi.org
`ipopt`	0.2.0	https://pypi.org/project/ipopt
`matlab`	0.1	https://pypi.org/project/matlab
`matplotlib`	3.2.0	https://matplotlib.org
`modOpt`	0.1	https://git.tu-berlin.de/dbta/simulation/modOpt
`mpmath`	1.1.0	https://mpmath.org
`multi-processing`	2.6.2.1	https://pypi.org/project/multiprocessing
`numpy`	1.19.3	https://numpy.org
`optuna`	2.10.0	https://optuna.org
`pyibex`	1.9.2	https://pypi.org/project/pyibex
`scipy`	1.5.3	https://scipy.org
`sympy`	1.5.1	https://sympy.org

Tab. A.3: List of all other used software.

Name	Subroutines	Version	Reference
`Ampl`		20220310	https://ampl.com/
`HSL`	MC19	1.0.0	https://hsl.rl.ac.uk
	MC27	1.0.0	https://hsl.rl.ac.uk
	MC29	1.0.0	https://hsl.rl.ac.uk
	MC77	1.0.1	https://hsl.rl.ac.uk
`Matlab`		*R*2020*a*	https://mathworks.com
`MOSAICmodeling`		3.0.1	https://mosaic-modeling.de
`Python`		3.7.6	https://python.org/

Appendix B

Tested Models

In addition to the NLEs and specifications of the four examples discussed in detail in this thesis, four additional process models are presented here that have also been tested to verify the applicability of the hybrid approach. Since structurally they do not differ too much from the other four examples, their presentation in the thesis has been omitted. Nevertheless, they have been considered relevant in chapter 6, especially in figure 6.1. Their structural properties are shown in table B.1.

Tab. B.1: Structural properties of largest subsystems of additionally tested NLEs at the known solutions: n_{ls} is the dimension, κ_2 is the condition number of the Jacobian evaluated at the solution, ρ_{nz} is the non-zero density of the Jacobian and ε_{nl} is the nonlinearity ratio.

NLE	n_{ls}	κ_2	ρ_{nz}	ε_{nl}
van der Waals' System	1	1.0×10^0	1.000	1.000
	1	1.0×10^0	1.000	1.000
Reactive Flash Unit	15	8.0×10^6	0.307	0.290
Partial Condenser	29	4.9×10^{11}	0.162	0.493
Methanol/Water Column	100	4.6×10^6	0.045	0.095

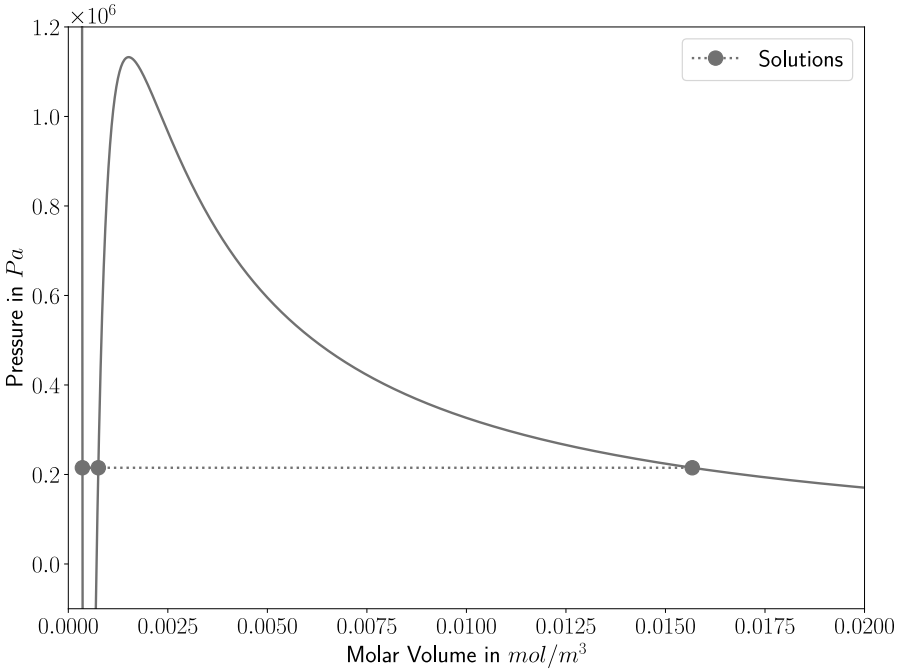


Fig. B.1: Solutions of van der Waals' equation of state applied on n-octane at a pressure of 2.15 bar and a temperature of 427.85 K.

B.1 van der Waals' System

In this example the molar volume v of n-octane is determined by van der Waals' equation of state (van der Waals, 1873). At a pressure of 2.15 bar and temperature of 427.85 K, the component is boiling, i.e., there are two physical solutions. The smaller molar volume belongs to the liquid phase and the larger to the vapor phase. In between, there is a third purely mathematical solution, also called unstable solution. All properties of n-octane and process conditions of this example are taken from Rao (1997, pp. 53-55). Figure B.1 shows the three solutions to the given pressure that exist for the cubic equation of state. In the physical model, only one or both physical solutions are of interest. However, a conventional Newton method is not prevented from converging to the unstable solution when the initial point is not chosen favorably. By restricting the derivative of pressure with

respect to volume to values less than or equal to zero, the unstable solution can be removed by means of IA. In a similar way, by the auxiliary equation

$$v = v^{ph} + v^{crit} \quad , \tag{B.1}$$

with the auxiliary variable v^{ph} and the critical volume v^{crit}, the solution for the liquid phase ($v^{ph} > 0$) or the solution for the vapor phase ($v^{ph} < 0$) can be excluded if required. Thus, via these additional auxiliary conditions, discontinuous, conditional syntax can be avoided and the desired solution(s) are obtained based solely on the initial variable range. The system is the only one in this work that is not complex according to the definition in section 4.1, since all three equations can be solved separately The hybrid approach correctly determines no, up to three solutions depending on the initialization of the auxiliary variables. In the computational experiments only the case with two solutions is examined more closely. To exclude the unstable solution, the initial interval of the pressure derivative with respect to the volume is set to $[-10^9, 0]$. The full equation system including its notation, variable and parameter specifications as well as its solutions is presented next.

Tab. B.2: Notation, base names.

Base Name	Description	Engineering Unit
P	Pressure	Pa
R	Ideal gas constant	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T	Temperature	K
a	Van der Waals constant a	$\mathrm{Pam^6/mol^2}$
b	Van der Waals constant b	$\mathrm{mol\,m^{-3}}$
$dPdv$	Ratio of first pressure derivative with respect to molar volume to critical pressure	$\mathrm{m^3\,mol^{-1}}$
v	Molar volume	$\mathrm{mol\,m^{-3}}$

Tab. B.3: Notation, superscripts.

Superscript	Description
$crit$	critical
ph	phase

Equation System

$$
\begin{aligned}
0 &= P - \frac{R \cdot T}{v - b} + \frac{a}{(v)^2} \\
0 &= dPdv^{crit} + \frac{R \cdot T}{(v - b)^2 \cdot P^{crit}} - \frac{2 \cdot a}{P^{crit} \cdot (v)^3} \\
0 &= v^{ph} + v^{crit} - v
\end{aligned}
$$

Tab. B.4: Design variable specification.

Design Variable		Value	Engineering Unit
P	=	$2.150E5$	Pa
P^{crit}	=	$1.999E7$	Pa
R	=	8.314	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T	=	$4.279E2$	K
a	=	3.789	$\mathrm{Pam^6/mol^2}$
b	=	$2.370E-4$	$\mathrm{mol\,m^{-3}}$
v^{crit}	=	$7.110E-4$	$\mathrm{mol\,m^{-3}}$

Tab. B.5: Liquid solution.

Iteration Variable		Value	Engineering Unit
v	=	$3.535E-4$	$\mathrm{mol\,m^{-3}}$
$dPdv^{crit}$	=	$-4.535E3$	$\mathrm{m^3\,mol^{-1}}$
v^{ph}	=	$-3.575E-4$	$\mathrm{mol\,m^{-3}}$

Tab. B.6: Vapor solution.

Iteration Variable		Value	Engineering Unit
v	=	$1.567E-2$	$\mathrm{mol\,m^{-3}}$
$dPdv^{crit}$	=	$-6.483E-1$	$\mathrm{m^3\,mol^{-1}}$
v^{ph}	=	$1.496E-2$	$\mathrm{mol\,m^{-3}}$

Tab. B.7: Unstable solution.

Iteration Variable		Value	Engineering Unit
v	=	$7.539E-4$	$\mathrm{mol\,m^{-3}}$
$dPdv^{crit}$	=	$2.188E2$	$\mathrm{m^3\,mol^{-1}}$
v^{ph}	=	$4.285E-5$	$\mathrm{mol\,m^{-3}}$

B.2 CSTR

Tab. B.8: Notation, base names.

Base Name	Description	Engineering Unit
E	Activation energy	$\mathrm{J\,mol^{-1}}$
M	Molar mass	$\mathrm{g\,mol^{-1}}$
Q	Volume flow rate	$\mathrm{cm^3\,h^{-1}}$
R	Ideal gas constant	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T	Temperature	K
X	Conversion	$\mathrm{g\,g^{-1}}$
ΔHR	Reaction enthalpy	$\mathrm{J\,mol^{-1}}$
ρ	Density	$\mathrm{g\,cm^{-3}}$
c	Concentration	$\mathrm{mol\,cm^{-3}}$
cp	Heat capacity	$\mathrm{J\,mol^{-1}\,K^{-1}}$
m	Mass	g
r	Reaction rate	$\mathrm{mol\,h^{-1}\,g^{-1}}$
x	Weight fraction	$\mathrm{g\,g^{-1}}$

Tab. B.9: Notation, subscripts.

Subscript	Description
Cat	Catalyst
$Feed$	Feed

Tab. B.10: Notation, indices.

Index	Range	Description
c	$1 \dots NC$	Component index

Equation System

$$
\begin{aligned}
0 &= X_c - \frac{c_{Feed,c} - c_c}{c_{Feed,c}} \\
0 &= Q_{Feed} - (\frac{1}{X_c} - 1) \cdot m_{Cat} \cdot r \\
0 &= c_{Feed,c} - x_{Feed,c} \cdot \frac{\rho}{M_c} \\
0 &= \frac{T - T_{Feed}}{X_c} - \frac{-\Delta HR \cdot c_{Feed,c}}{\rho \cdot cp} \\
0 &= r - 1.3 \cdot (10)^{11} \cdot \exp(\frac{-E}{R \cdot T})
\end{aligned}
$$

Tab. B.11: Component index.

Index	Component
$c = 1$	Aromatics

Tab. B.12: Design variable specification.

Design Variable		Value	Engineering Unit
E	=	$1.355E5$	$\mathrm{J\,mol^{-1}}$
$M_{c=1}$	=	$2.100E2$	$\mathrm{g\,mol^{-1}}$
Q_{Feed}	=	$3.000E1$	$\mathrm{cm^3\,h^{-1}}$
R	=	8.314	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T_{Feed}	=	$5.650E2$	K
ΔHR	=	$-2.000E5$	$\mathrm{J\,mol^{-1}}$
ρ	=	$8.500E-1$	$\mathrm{g\,cm^{-3}}$
cp	=	1.750	$\mathrm{J\,mol^{-1}\,K^{-1}}$
m_{Cat}	=	$2.000E1$	g
$x_{Feed,c=1}$	=	$3.000E-1$	$\mathrm{g\,g^{-1}}$

Tab. B.13: Solution with low conversion.

Iteration Variable		Value	Engineering Unit
T	=	$5.703E2$	K
$X_{c=1}$	=	$3.241E-2$	$\mathrm{g\,g^{-1}}$
$c_{Feed,c=1}$	=	$1.214E-3$	$\mathrm{mol\,cm^{-3}}$
$c_{c=1}$	=	$1.175E-3$	$\mathrm{mol\,cm^{-3}}$
r	=	$5.023E-2$	$\mathrm{mol\,h^{-1}\,g^{-1}}$

Tab. B.14: Solution with high conversion.

Iteration Variable		Value	Engineering Unit
T	=	$7.142E2$	K
$X_{c=1}$	=	$9.140E-1$	$\mathrm{g\,g^{-1}}$
$c_{Feed,c=1}$	=	$1.214E-3$	$\mathrm{mol\,cm^{-3}}$
$c_{c=1}$	=	$1.044E-4$	$\mathrm{mol\,cm^{-3}}$
r	=	$1.594E1$	$\mathrm{mol\,h^{-1}\,g^{-1}}$

Tab. B.15: Unstable solution.

Iteration Variable		Value	Engineering Unit
T	=	$6.482E2$	K
$X_{c=1}$	=	$5.094E-1$	$\mathrm{g\,g^{-1}}$
$c_{Feed,c=1}$	=	$1.214E-3$	$\mathrm{mol\,cm^{-3}}$
$c_{c=1}$	=	$5.958E-4$	$\mathrm{mol\,cm^{-3}}$
r	=	$1.557E1$	$\mathrm{mol\,h^{-1}\,g^{-1}}$

B.3 Flash Unit

Tab. B.16: Notation, base names.

Base Name	Description	Engineering Unit
A	Parameter	various
B	Parameter	various

Base Name	Description	Engineering Unit
C	Parameter	various
D	Parameter	various
E	Parameter	various
F	Molar flow; parameter	$\mathrm{mol\,s^{-1}}$; various
H	Height	m
HU	Molar holdup	mol
K	Equilibrium constant	–
Q	Heat flow	$\mathrm{J\,s^{-1}}$
R	Ideal gas constant	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T	Temperature	K
U	Internal energy	J
V	Volume	$\mathrm{m^3}$
Δh	Enthalpy difference	$\mathrm{J\,mol^{-1}}$
Δp	Pressure drop	bar
α	Wilson parameter	–
γ	Activity coefficient	–
λ	Wilson parameter	–
π	Physical constant pi	–
h	Molar enthalpy	$\mathrm{J\,mol^{-1}}$
p	Pressure	bar
v	Molar volume	$\mathrm{m^3\,mol^{-1}}$
x	Mole fraction (liquid)	$\mathrm{mol\,mol^{-1}}$
y	Mole fraction (vapor)	$\mathrm{mol\,mol^{-1}}$
z	Compressibility factor	–

Tab. B.17: Notation, superscripts.

Superscript	Description
E	Excess
F	Feed
L	Liquid
LV	Saturated
V	Vapor
$total$	Total

Tab. B.18: Notation, subscripts.

Subscript	Description
hL	Correlation enthalpy (liquid)
hV	Correlation enthalpy (vapor)
pLV	Correlation saturation pressure

Tab. B.19: Notation, indices.

Index	Range	Description
i	$1 \dots NC$	Component index

Tab. B.20: Component index.

Index	Component
$i = 1$	Ethanol
$i = 2$	Water

Equation System

$$
\begin{aligned}
0 &= F^F \cdot x_i^F - F^V \cdot y_i - F^L \cdot x_i \\
0 &= K_i^{LV} \cdot x_i - y_i \\
0 &= \sum_{i=1}^{NC} x_i^F - 1 \\
0 &= \sum_{i=1}^{NC} x_i - 1 \\
0 &= \sum_{i=1}^{NC} y_i - 1 \\
0 &= F^F \cdot h^F - F^V \cdot h^V - F^L \cdot h^L + Q \\
0 &= p^F - p - \Delta p
\end{aligned}
$$

$$0 = K_i^{LV} - \frac{\gamma_i^L \cdot p_i^{LV}}{p}$$

$$0 = \gamma_i^L - \frac{1}{x_i + \alpha_i^L \cdot (1 - x_i)} \cdot \exp((1 - x_i) \cdot (\frac{\alpha_i^L}{x_i + \alpha_i^L \cdot (1 - x_i)} - \frac{\sum_{i=1}^{NC} \alpha_i^L - \alpha_i^L}{(\sum_{i=1}^{NC} \alpha_i^L - \alpha_i^L) \cdot x_i + (1 - x_i)}))$$

$$0 = \frac{\sum_{i=1}^{NC} v_i^L - v_i^L}{v_i^L} \cdot \exp(\frac{-\lambda_i^L}{T}) - \alpha_i^L$$

$$0 = \sum_{i=1}^{NC} x_i^F \cdot h_i^F + h^{E,F} - h^F$$

$$0 = \sum_{i=1}^{NC} x_i \cdot h_i^L + h^{E,L} - h^L$$

$$0 = \sum_{i=1}^{NC} y_i \cdot h_i^V + h^{E,V} - h^V$$

$$0 = \frac{HU^V \cdot R \cdot T \cdot z^V}{p \cdot 10^5} - V^V$$

$$0 = (\sum_{i=1}^{NC} v_i^L \cdot x_i + v^{L,E}) \cdot HU^L - V^L$$

$$0 = x_i \cdot HU^L + y_i \cdot HU^V - HU_i$$

$$0 = HU^L \cdot (h^L - p \cdot 10^5 \cdot (\sum_{i=1}^{NC} x_i \cdot v_i^L + v^{L,E})) + HU^V \cdot (h^V - R \cdot T \cdot z^V) - U$$

$$0 = V^L + V^V - V^{total}$$

$$0 = 4 \cdot \frac{V^L}{\pi \cdot (D)^2} - H^L$$

$$0 = r^{D,H} - \frac{D}{H}$$

$$0 = A - \frac{\pi}{4} \cdot (D)^2$$

$$0 = V^{total} - A \cdot H$$

Functions

$$h(T) = A \cdot \frac{T}{1000} + \frac{B}{2} \cdot (\frac{T}{1000})^2 + \frac{C}{3} \cdot (\frac{T}{1000})^3 + \frac{D}{4} \cdot (\frac{T}{1000})^4 + E \cdot (\frac{T}{1000})^{-1} + F + \Delta h$$

Applications:

- $h_i^F(T^F, A_{hL,i} \dots F_{hL,i}, \Delta h_{hL,i})$
- $h_i^L(T, A_{hL,i} \dots F_{hL,i}, \Delta h_{hL,i})$
- $h_i^V(T, A_{hV,i} \dots F_{hV,i}, \Delta h_{hV,i})$

$$p(T) = (10)^{A - \frac{B}{C+T}}$$

Applications:

- $p_i^{LV}(T, A_{pLV,i} \dots C_{pLV,i})$

Tab. B.21: Design variable specification.

Design Variable		Value	Engineering Unit
D	=	0.16	m
F^F	=	80.0	$\mathrm{mol\,s^{-1}}$
H	=	0.5	m
H^L	=	0.25	m
T	=	353.15	K
T^F	=	353.15	K
Δp	=	0.25	bar
π	=	3.14159265359	−
$h^{E,F}$	=	0.0	$\mathrm{J\,mol^{-1}}$
$h^{E,L}$	=	0.0	$\mathrm{J\,mol^{-1}}$
$h^{E,V}$	=	0.0	$\mathrm{J\,mol^{-1}}$
p^F	=	1.0	bar
$v^{E,L}$	=	0.0	$\mathrm{m^3\,mol^{-1}}$
$x^F_{i=1}$	=	0.15	$\mathrm{mol\,mol^{-1}}$
z^V	=	1.0	−

Tab. B.22: Parameter specification.

Parameter		Value	Engineering Unit
$A_{hL,i=1}$	=	102538.0	$J\,mol^{-1}\,K^{-1}$
$A_{hL,i=2}$	=	−203.606	$J\,mol^{-1}\,K^{-1}$
$A_{hV,i=1}$	=	5385.58	$J\,mol^{-1}\,K^{-1}$
$A_{hV,i=2}$	=	30.092	$J\,mol^{-1}\,K^{-1}$
$A_{pLV,i=1}$	=	5.24677	−
$A_{pLV,i=2}$	=	5.0768	−
$B_{hL,i=1}$	=	−138.44	$J\,mol^{-1}\,K^{-2}$
$B_{hL,i=2}$	=	1523.29	$J\,mol^{-1}\,K^{-2}$
$B_{hV,i=1}$	=	236.1088	$J\,mol^{-1}\,K^{-2}$
$B_{hV,i=2}$	=	6.832514	$J\,mol^{-1}\,K^{-2}$
$B_{pLV,i=1}$	=	1598.673	K
$B_{pLV,i=2}$	=	1659.793	K
$C_{hL,i=1}$	=	−0.03469	$J\,mol^{-1}\,K^{-3}$
$C_{hL,i=2}$	=	−3196.413	$J\,mol^{-1}\,K^{-3}$
$C_{hV,i=1}$	=	0.1237	$J\,mol^{-1}\,K^{-3}$
$C_{hV,i=2}$	=	6.793435	$J\,mol^{-1}\,K^{-3}$
$C_{pLV,i=1}$	=	−46.424	K
$C_{pLV,i=2}$	=	−45.854	K
$D_{hL,i=1}$	=	20.4367	$J/mol/K^4$
$D_{hL,i=2}$	=	2474.455	$J/mol/K^4$
$D_{hV,i=1}$	=	$2.3E-5$	$J/mol/K^4$
$D_{hV,i=2}$	=	−2.53448	$J/mol/K^4$
$E_{hL,i=1}$	=	0.0	K
$E_{hL,i=2}$	=	3.855326	K
$E_{hV,i=1}$	=	$3.7E-5$	K
$E_{hV,i=2}$	=	0.082139	K
$F_{hL,i=1}$	=	0.0	$J\,mol^{-1}$
$F_{hL,i=2}$	=	−256.5478	$J\,mol^{-1}$
$F_{hV,i=1}$	=	0.0	$J\,mol^{-1}$
$F_{hV,i=2}$	=	−250.881	$J\,mol^{-1}$
R	=	8.314	$J\,mol^{-1}\,K^{-1}$
$\Delta h_{hL,i=1}$	=	−276000.0	$J\,mol^{-1}$
$\Delta h_{hL,i=2}$	=	−285830.0	$J\,mol^{-1}$
$\Delta h_{hV,i=1}$	=	−234000.0	$J\,mol^{-1}$
$\Delta h_{hV,i=2}$	=	−276000.0	$J\,mol^{-1}$

Parameter		Value	Engineering Unit
$\lambda^{L}_{i=1}$	=	95.68	K
$\lambda^{L}_{i=2}$	=	506.7	K
$v^{L}_{i=1}$	=	$5.869E-5$	$m^3\,mol^{-1}$
$v^{L}_{i=2}$	=	$1.807E-5$	$m^3\,mol^{-1}$

Tab. B.23: Physical solution.

Iteration Variable		Value	Engineering Unit
A	=	$2.011E-2$	m^2
F^{L}	=	$6.429E1$	$mol\,s^{-1}$
F^{V}	=	$1.571E1$	$mol\,s^{-1}$
HU^{L}	=	$2.330E2$	mol
HU^{V}	=	$1.284E-1$	mol
$HU_{i=1}$	=	$2.015E1$	mol
$HU_{i=2}$	=	$2.130E2$	mol
$K^{LV}_{i=1}$	=	4.765	−
$K^{LV}_{i=2}$	=	0.645	−
Q	=	$1.410E5$	$J\,s^{-1}$
U	=	$-6.58E4$	J
V^{L}	=	$5.03E-3$	m^3
V^{V}	=	$5.03E-3$	m^3
V^{total}	=	$1.005E-2$	m^3
$\alpha^{L}_{i=1}$	=	$2.348E-1$	−
$\alpha^{L}_{i=2}$	=	$7.774E-1$	−
$\gamma^{L}_{i=1}$	=	3.299	−
$\gamma^{L}_{i=2}$	=	1.021	−
h^{F}	=	$-2.791E5$	$J\,mol^{-1}$
h^{L}	=	$-2.821E5$	$J\,mol^{-1}$
h^{V}	=	$-2.581E5$	$J\,mol^{-1}$
p	=	$7.5E-1$	bar
$r^{D,H}$	=	$3.2E-1$	−
$x_{i=1}$	=	$8.624E-2$	$mol\,mol^{-1}$
$x_{i=2}$	=	$9.138E-1$	$mol\,mol^{-1}$
$x^{F}_{i=2}$	=	$8.500E-1$	$mol\,mol^{-1}$
$y_{i=1}$	=	$4.109E-1$	$mol\,mol^{-1}$
$y_{i=2}$	=	$5.891E-1$	$mol\,mol^{-1}$

Tab. B.24: Mathematical solution.

Iteration Variable		Value	Engineering Unit
A	=	$2.011E-2$	m^2
F^L	=	$2.025E2$	$mol\,s^{-1}$
F^V	=	$-1.225E2$	$mol\,s^{-1}$
HU^L	=	$6.876E1$	mol
HU^V	=	$1.284E-1$	mol
$HU_{i=1}$	=	$9.343E1$	mol
$HU_{i=2}$	=	$-2.454E1$	mol
$K^{LV}_{i=1}$	=	1.581	−
$K^{LV}_{i=2}$	=	3.217	−
Q	=	$-6.464E5$	$J\,s^{-1}$
U	=	$-1.538E6$	J
V^L	=	$5.03E-3$	m^3
V^V	=	$5.03E-3$	m^3
V^{total}	=	$1.005E-2$	m^3
$\alpha^L_{i=1}$	=	$2.348E-1$	−
$\alpha^L_{i=2}$	=	$7.774E-1$	−
$\gamma^L_{i=1}$	=	1.094	−
$\gamma^L_{i=2}$	=	5.094	−
h^F	=	$-2.791E5$	$J\,mol^{-1}$
h^L	=	$-2.223E5$	$J\,mol^{-1}$
h^V	=	$-1.817E5$	$J\,mol^{-1}$
p	=	$7.5E-1$	bar
$r^{D,H}$	=	$3.2E-1$	−
$x_{i=1}$	=	1.350	$mol\,mol^{-1}$
$x_{i=2}$	=	$-3.547E-1$	$mol\,mol^{-1}$
$x^F_{i=2}$	=	$8.500E-1$	$mol\,mol^{-1}$
$y_{i=1}$	=	2.141	$mol\,mol^{-1}$
$y_{i=2}$	=	−1.141	$mol\,mol^{-1}$

B.4 Reactive Flash Unit

This model describes the synthesis of methyl acetate (CH_3COOCH_3) by the esterification of acetic acid (CH_3CCOH) and methanol (CH_3OH)

$$CH_3COOH + CH_3OH \leftrightarrows CH_3COOCH_3 + H_2O \quad .$$

A liquid, equimolar mixture of acetic acid and methanol is fed to the unit at a pressure of 1 bar and a temperature of 323.15 K. Catalyst such as sulfuric acid is additionally inserted. How much methyl acetate is produced in the liquid phase depends on the liquid's composition at chemical equilibrium and how far away the mixture is from it. Reaction kinetics are accordingly taken into account in this model. To increase the yield of methyl acetate, heat is supplied to partially evaporate the liquid mixture at a temperature of 353.15 K. Due to its low boiling temperature methyl acetate accumulates mainly in the vapor stream next to methanol. The two phases within the flash unit are assumed to be well-mixed and in phase equilibrium. Reaction kinetics and phase equilibrium are described by the approach of Huss et al. (2003), in which the activity coefficients of the liquid phase are expressed according to Wilson's model. Equations for the equilibrium constant and the reaction rate as well as the respective parameters are retrieved from studies of Song et al. (1998). The model encompasses 53 equations, has one solution at the given conditions in the initial box $\underline{\overline{x}} = [-10^9, 10^9]^{53}$ and its largest, complex subsystem consists of 15 equations. The NLE, parameter and variable specifications as well as the solution are listed below.

Tab. B.25: Notation, base names.

Base Name	Description	Engineering Unit
A	Parameter	various
B	Parameter	various
C	Parameter	various
HU	Hold-up	mol
K	Equilibrium constant	$-$
L	Liquid ratio	$-$
N	Mole flow	$\mathrm{mol\,s^{-1}}$
Q	Heat flux	$\mathrm{J\,s^{-1}}$
R	Ideal gas constant	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T	Temperature	K
γ	Activity coefficient	$-$
ν	Stochiometric coefficient	$-$
c	Heat capacity	$\mathrm{J\,mol^{-1}\,K^{-1}}$
h	Molar enthalpy [	$\mathrm{J\,mol^{-1}}$

Base Name	Description	Engineering Unit
k	Rate constant	s^{-1}
p	Pressure	Pa
r	Reaction rate	s^{-1}
v	Specific volume	$cm^3\,mol^{-1}$
x	Liquid mole fraction	$mol\,mol^{-1}$
y	Vapor mole fraction	$mol\,mol^{-1}$

Tab. B.26: Notation, superscripts.

Superscript	Description
Antoine	Antoine equation for vapor pressure calculation
F	feed
L	liquid
LV	vapor liquid equilibrium
V	vapor
Wilson	Wilson's gE model
ref	reference conditions

Tab. B.27: Notation, subscripts.

Subscript	Description
eq	equilibrium
f	forward
p	constant pressure

Tab. B.28: Notation, indices.

Index	Range	Description
i	$1 \ldots Ni$	component index
j	$1 \ldots Nj$	secondary component index
k	$1 \ldots Nk$	ternary component index

Tab. B.29: Component index.

Index	Component
$i = j = k = 1$	Acetic Acid
$i = j = k = 2$	Methanol
$i = j = k = 3$	Methyl Acetate
$i = j = k = 4$	Water

Equation System

$$0 = N^F \cdot x_i^F - N^L \cdot x_i - N^V \cdot y_i + \nu_i \cdot r$$
$$0 = K_{eq} - 2.32 \cdot \exp(\frac{782.98}{T})$$
$$0 = \sum_{i=1}^{Ni} x_i - 1$$
$$0 = \sum_{i=1}^{Ni} y_i - 1$$
$$0 = N^F \cdot h^F - N^L \cdot h^L - N^V \cdot h^V + Q$$
$$0 = \sum_{i=1}^{Ni} y_i \cdot h_i^V - h^V$$
$$0 = \sum_{i=1}^{Ni} x_i \cdot h_i^L - h^L$$
$$0 = \sum_{i=1}^{Ni} x_i^F \cdot h_i^F - h^F$$
$$0 = \frac{v_{i=j}}{v_i} \cdot \exp(\frac{-1.0 \cdot A_{i,j}^{Wilson}}{R \cdot T}) - L_{i,j}$$
$$0 = \frac{9.732 \cdot (10)^8}{3600} \cdot \exp(\frac{-6287.7}{T}) - k_f$$
$$0 = k_f \cdot (\gamma_{i=1} \cdot x_{i=1} \cdot \gamma_{i=2} \cdot x_{i=2} - \frac{\gamma_{i=3} \cdot x_{i=3} \cdot \gamma_{i=4} \cdot x_{i=4}}{K_{eq}}) - r$$

$$0 = c_{p,i}^{V} \cdot (T - T_i^{LV}) + h_i^{LV} + c_{p,i}^{L} \cdot (T_i^{LV} - T_i^{ref}) + h_i^{ref} - h_i^{V}$$
$$0 = c_{p,i}^{L} \cdot (T - T_i^{ref}) + h_i^{ref} - h_i^{L}$$
$$0 = c_{p,i}^{L} \cdot (T^{F} - T_i^{ref}) + h_i^{ref} - h_i^{F}$$
$$0 = y_i \cdot p - \gamma_i \cdot x_i \cdot p_i^{LV}$$
$$0 = 1 - \ln(\sum_{j=1}^{Nj} x_{i=j} \cdot L_{i,j}) - \sum_{k=1}^{Nk} \frac{x_{i=k} \cdot L_{i=k,j=i}}{\sum_{j=1}^{Nj} x_{i=j} \cdot L_{i=k,j}} - \ln(\gamma_i)$$
$$0 = A_i^{Antoine} + \frac{B_i^{Antoine}}{T + C_i^{Antoine}} - \ln(p_i^{LV})$$

Tab. B.30: Design variable specification.

Design Variable		Value	Engineering Unit
$A_{i=1,j=1}^{Wilson}$	=	0.0	–
$A_{i=1,j=2}^{Wilson}$	=	2535.2019	–
$A_{i=1,j=3}^{Wilson}$	=	1123.1444	–
$A_{i=1,j=4}^{Wilson}$	=	237.5248	–
$A_{i=1}^{Antoine}$	=	22.1001	–
$A_{i=2,j=1}^{Wilson}$	=	−547.5248	–
$A_{i=2,j=2}^{Wilson}$	=	0.0	–
$A_{i=2,j=3}^{Wilson}$	=	813.1843	–
$A_{i=2,j=4}^{Wilson}$	=	107.3832	–
$A_{i=2}^{Antoine}$	=	23.4999	–
$A_{i=3,j=1}^{Wilson}$	=	−696.5031	–
$A_{i=3,j=2}^{Wilson}$	=	−31.1932	–
$A_{i=3,j=3}^{Wilson}$	=	0.0	–
$A_{i=3,j=4}^{Wilson}$	=	645.7225	–
$A_{i=3}^{Antoine}$	=	21.152	–
$A_{i=4,j=1}^{Wilson}$	=	658.0266	–
$A_{i=4,j=2}^{Wilson}$	=	469.5509	–
$A_{i=4,j=3}^{Wilson}$	=	1918.232	–
$A_{i=4,j=4}^{Wilson}$	=	0.0	–
$A_{i=4}^{Antoine}$	=	23.2256	–

Design Variable		Value	Engineering Unit
$B_{i=1}^{Antoine}$	=	-3654.62	K
$B_{i=2}^{Antoine}$	=	-3643.3136	K
$B_{i=3}^{Antoine}$	=	-2662.78	K
$B_{i=4}^{Antoine}$	=	-3835.18	K
$C_{i=1}^{Antoine}$	=	-45.392	K
$C_{i=2}^{Antoine}$	=	-33.434	K
$C_{i=3}^{Antoine}$	=	-53.46	K
$C_{i=4}^{Antoine}$	=	-45.343	K
HU	=	10.0	mol
N^F	=	1.0	$\mathrm{mol\,s^{-1}}$
R	=	8.314	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T^F	=	323.15	K
T	=	323.15	K
$T_{i=1}^{LV}$	=	391.2	K
$T_{i=1}^{ref}$	=	298.15	K
$T_{i=2}^{LV}$	=	337.8	K
$T_{i=2}^{ref}$	=	298.15	K
$T_{i=3}^{LV}$	=	330.0	K
$T_{i=3}^{ref}$	=	298.15	K
$T_{i=4}^{LV}$	=	373.15	K
$T_{i=4}^{ref}$	=	298.15	K
$\nu_{i=1}$	=	-1.0	$-$
$\nu_{i=2}$	=	-1.0	$-$
$\nu_{i=3}$	=	1.0	$-$
$\nu_{i=4}$	=	1.0	$-$
$c_{p,i=1}^{L}$	=	123.0	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$c_{p,i=1}^{V}$	=	50.0	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$c_{p,i=2}^{L}$	=	80.0	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$c_{p,i=2}^{V}$	=	45.0	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$c_{p,i=3}^{L}$	=	140.0	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$c_{p,i=3}^{V}$	=	86.0	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$c_{p,i=4}^{L}$	=	75.6	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$c_{p,i=4}^{V}$	=	35.0	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$h_{i=1}^{LV}$	=	875160.0	$\mathrm{J\,mol^{-1}}$
$h_{i=1}^{ref}$	=	483520.0	$\mathrm{J\,mol^{-1}}$

Design Variable		Value	Engineering Unit
$h_{i=2}^{LV}$	=	725700.0	J mol^{-1}
$h_{i=2}^{ref}$	=	238400.0	J mol^{-1}
$h_{i=3}^{LV}$	=	1583000.0	J mol^{-1}
$h_{i=3}^{ref}$	=	445890.0	J mol^{-1}
$h_{i=4}^{LV}$	=	40650.0	J mol^{-1}
$h_{i=4}^{ref}$	=	285830.0	J mol^{-1}
p	=	100000.0	Pa
$v_{i=1}$	=	57.54	cm^3 mol^{-1}
$v_{i=2}$	=	44.44	cm^3 mol^{-1}
$v_{i=3}$	=	79.84	cm^3 mol^{-1}
$v_{i=4}$	=	18.07	cm^3 mol^{-1}
$x_{i=1}^{F}$	=	0.5	mol mol^{-1}
$x_{i=2}^{F}$	=	0.5	mol mol^{-1}
$x_{i=3}^{F}$	=	0.0	mol mol^{-1}
$x_{i=4}^{F}$	=	0.0	mol mol^{-1}

Tab. B.31: Physical Solution.

Iteration Variable		Value	Engineering Unit
K_{eq}	=	$2.130E1$	–
$L_{i=1,j=1}$	=	1.000	–
$L_{i=1,j=2}$	=	$3.257E-1$	–
$L_{i=1,j=3}$	=	$9.465E-1$	–
$L_{i=1,j=4}$	=	$2.896E-1$	–
$L_{i=2,j=1}$	=	1.560	–
$L_{i=2,j=2}$	=	1.000	–
$L_{i=2,j=3}$	=	1.362	–
$L_{i=2,j=4}$	=	$3.920E-1$	–
$L_{i=3,j=1}$	=	$9.136E-1$	–
$L_{i=3,j=2}$	=	$5.626E-1$	–
$L_{i=3,j=3}$	=	1.000	–
$L_{i=3,j=4}$	=	$1.816E-1$	–
$L_{i=4,j=1}$	=	2.545	–
$L_{i=4,j=2}$	=	2.096	–
$L_{i=4,j=3}$	=	2.299	–
$L_{i=4,j=4}$	=	1.000	–

Iteration Variable		Value	Engineering Unit
N^L	=	$7.585E-1$	$\mathrm{mol\,s^{-1}}$
N^V	=	$2.415E-1$	$\mathrm{mol\,s^{-1}}$
Q	=	$1.883E5$	$\mathrm{J\,s^{-1}}$
$\gamma_{i=1}$	=	1.026	$-$
$\gamma_{i=2}$	=	1.154	$-$
$\gamma_{i=3}$	=	1.077	$-$
$\gamma_{i=4}$	=	$8.109E-1$	$-$
h^F	=	$3.635E5$	$\mathrm{J\,mol^{-1}}$
h^L	=	$3.913E5$	$\mathrm{J\,mol^{-1}}$
h^V	=	$1.056E6$	$\mathrm{J\,mol^{-1}}$
$h^F_{i=1}$	=	$4.866E5$	$\mathrm{J\,mol^{-1}}$
$h^L_{i=1}$	=	$4.903E5$	$\mathrm{J\,mol^{-1}}$
$h^V_{i=1}$	=	$1.368E6$	$\mathrm{J\,mol^{-1}}$
$h^F_{i=2}$	=	$2.404E5$	$\mathrm{J\,mol^{-1}}$
$h^L_{i=2}$	=	$2.428E5$	$\mathrm{J\,mol^{-1}}$
$h^V_{i=2}$	=	$9.680E5$	$\mathrm{J\,mol^{-1}}$
$h^F_{i=3}$	=	$4.494E5$	$\mathrm{J\,mol^{-1}}$
$h^L_{i=3}$	=	$4.536E5$	$\mathrm{J\,mol^{-1}}$
$h^V_{i=3}$	=	$2.035E6$	$\mathrm{J\,mol^{-1}}$
$h^F_{i=4}$	=	$2.877E5$	$\mathrm{J\,mol^{-1}}$
$h^L_{i=4}$	=	$2.900E5$	$\mathrm{J\,mol^{-1}}$
$h^V_{i=4}$	=	$3.315E5$	$\mathrm{J\,mol^{-1}}$
k_f	=	$5.006E-3$	$\mathrm{s^{-1}}$
$p^{LV}_{i=1}$	=	$2.759E4$	Pa
$p^{LV}_{i=2}$	=	$1.807E5$	Pa
$p^{LV}_{i=3}$	=	$2.125E5$	Pa
$p^{LV}_{i=4}$	=	$4.738E4$	Pa
r	=	$1.344E-3$	$\mathrm{s^{-1}}$
$x_{i=1}$	=	$5.885E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{i=2}$	=	$3.855E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{i=3}$	=	$1.025E-2$	$\mathrm{mol\,mol^{-1}}$
$x_{i=4}$	=	$1.579E-2$	$\mathrm{mol\,mol^{-1}}$
$y_{i=1}$	=	$1.665E-1$	$\mathrm{mol\,mol^{-1}}$
$y_{i=2}$	=	$8.040E-1$	$\mathrm{mol\,mol^{-1}}$
$y_{i=3}$	=	$2.347E-2$	$\mathrm{mol\,mol^{-1}}$
$y_{i=4}$	=	$6.067E-3$	$\mathrm{mol\,mol^{-1}}$

B.5 Partial Condenser

In this unit, a vaporous stream of the five components Toluene (1), Biphenyl (2), Benzene (3), Hydrogen (4) and Methane (5) is partially condensed in order to remove a great part of the light-boiling components (4 to 5) from the heavy-boiling products, which remain in the condensate. Both phases in the condenser are assumed to be well-mixed and thermodynamic equilibrium holds between them. The VLE is described by SRK in the way suggested by Rao (1997, pp. 74-79, p. 280, p. 321). Further information about the actual implementation of this model can be found in the master thesis of Rajes (2020, p. 32-33). The modifications that have been made to this model to solve it more efficiently include the calculations of the VLE. They are analogous to those of *Heavies Column*. The system has a dimension of 34, of which the largest complex subsystem comprises 29 equations and it can be placed in the intermediate sized range with respect to the other systems. In comparison with the other systems, it has a high condition number of 4.9×10^{11} and nonlinearity ratio of 0.493. The complete model, its variable and parameter specifications, and a physical solution are presented next.

Tab. B.32: Notation, base names.

Base Name	Description	Engineering Unit
Δh	Molar enthalpy difference	$\mathrm{J\,mol^{-1}}$
Δp	Pressure drop per tray	Pa
Θ	Parameter Equation of State	–
α	Parameter Equation of State	–
β	Parameter Equation of State	–
γ	Parameter Equation of State	–
ω	Acentric factor	–
π	Mathematical constant	–
φ	Fugacity coefficient	–
A	Area; Parameter Equation of State	$\mathrm{m^2}$; *various*
a	Parameter Equation of State	$\mathrm{Pam^6/mol^2}$
$a0$	Parameter 1	*various*
$a1$	Parameter 2	*various*
$a2$	Parameter 3	*various*
$a3$	Parameter 4	*various*

Base Name	Description	Engineering Unit
$a4$	Parameter 5	*various*
$a5$	Parameter 6	*various*
$a6$	Parameter 7	*various*
$a7$	Parameter 8	*various*
$a8$	Parameter 9	*various*
$a9$	Parameter 10	*various*
$a10$	Parameter 11	*various*
aux	Auxiliary parameter	–
$aux1$	Auxiliary parameter 1	–
$aux2$	Auxiliary parameter 2	–
B	Parameter Equation of State	–
b	Parameter Equation of State	$\mathrm{mol\,m^{-3}}$
cp	Specific isobar heat capacity	$\mathrm{J\,mol^{-1}\,K^{-1}}$
D	Parameter Equation of State	*various*
F	Flow rate	$\mathrm{mol\,s^{-1}}$
h	Molar enthalpy	$\mathrm{MJ\,mol^{-1}}$
K	Equilibrium constant	–
$lnphi$	ln of fugacity coefficient	–
Pc	Critical pressure	Pa
n	Selection of Soave Redlich Kwong (n=1) and Peng Robinson (n=2)	–
p	Pressure; Parameter Equation of State	bar; –
Q	Heat duty	$\mathrm{MJ\,s^{-1}}$
q	Parameter Equation of State	–
R	Ideal gas constant	$\mathrm{J\,mol^{-1}\,K^{-1}}$
r	Parameter Equation of State	–
T	Temperature	K
Tc	Critical temperature	K
U	Internal energy	J
V	Volume	$\mathrm{m^3}$
x	Mole fraction liquid phase	$\mathrm{mol\,mol^{-1}}$
y	Mole fraction vapor phase	$\mathrm{mol\,mol^{-1}}$
Z	Compressibility factor	–

Tab. B.33: Notation, superscripts.

Superscript	Description
F	Feed
D	Distillate
B	Bottom
R	Reboiler
C	Condenser
L	Liquid phase
LV	Liquid-vapor
o	Standard state
V	Vapor phase

Tab. B.34: Notation, subscripts.

Subscript	Description
cp	Heat capacity
dep	Departure function
EoS	Equation of state
$EoSa$	Function a of Equation of state EQS
$EoSalp$	Function alpha of Equation of state EQS
$EoSb$	Function b of Equation of state EQS
f	Standard formation
hLV	Enthalpy of evaporation
max	Maximal
min	Minimal
mix	Mixture
Z	Auxiliary parameter Z

Tab. B.35: Notation, indices.

Index	Range	Description
i	$1..NC$	Component index
j	$1..NC$	Second component index
tr	$1..Ntr$	Tray index

Tab. B.36: Component index.

Index	Component
$i = j = 1$	Toluene
$i = j = 2$	Biphenyl
$i = j = 3$	Benzene
$i = j = 4$	Hydrogen
$i = j = 5$	Methanol
$tr = 1$	Tray

Equation System

$$0 = F^F_{tr} \cdot y_{F,tr,i} - F^L_{tr} \cdot x_{tr,i} - F^V_{tr} \cdot y_{tr,i}$$

$$0 = K_{tr,i} \cdot x_{tr,i} - y_{tr,i}$$

$$0 = \sum_{i=1}^{NC} x_{tr,i} - 1$$

$$0 = \sum_{i=1}^{NC} y_{tr,i} - 1$$

$$0 = \frac{F^F_{tr} \cdot h^F_{tr} - F^L_{tr} \cdot h^L_{tr} - F^V_{tr} \cdot h^V_{tr} + Q}{h^{sca}}$$

$$0 = \sum_{i=1}^{NC} x_{tr,i} \cdot (a_{tr,i})^{0.5} \cdot \sum_{j=1}^{NC} x_{tr,i=j} \cdot (a_{tr,i=j})^{0.5} \cdot (1 - k_{i,j}) - a^L_{mix,tr}$$

$$0 = \sum_{i=1}^{NC} y_{tr,i} \cdot (a_{tr,i})^{0.5} \cdot \sum_{j=1}^{NC} y_{tr,i=j} \cdot (a_{tr,i=j})^{0.5} \cdot (1 - k_{i,j}) - a^V_{mix,tr}$$

$$0 = \sum_{i=1}^{NC} x_{tr,i} \cdot b_i - b^L_{mix,tr}$$

$$0 = \sum_{i=1}^{NC} y_{tr,i} \cdot b_i - b^V_{mix,tr}$$

$$0 = lnphi^L_{tr,i} - lnphi^V_{tr,i} - \ln(\frac{y_{tr,i}}{x_{tr,i}})$$

$$0 = \frac{-q^L_{tr}}{2} + (D^L_{tr})^{0.5} - (aux1^L_{Z,tr})^3$$

$$0 = \frac{-q^L_{tr}}{2} - (D^L_{tr})^{0.5} - (aux2^L_{Z,tr})^3$$

$$0 = \frac{-q_{tr}^{V}}{2} + (D_{tr}^{V})^{0.5} - (aux1_{Z,tr}^{V})^{3}$$

$$0 = \frac{-q_{tr}^{V}}{2} - (D_{tr}^{V})^{0.5} - (aux2_{Z,tr}^{V})^{3}$$

$$0 = aux1_{Z,tr}^{L} + aux2_{Z,tr}^{L} - \frac{\alpha_{tr}^{L}}{3} - Z_{min,tr}^{L}$$

$$0 = \frac{-q_{tr}^{V}}{2} + (D_{tr}^{V})^{0.5} - (aux1_{Z,tr}^{V})^{3}$$

$$0 = aux1_{Z,tr}^{V} + aux2_{Z,tr}^{V} - \frac{\alpha_{tr}^{V}}{3} - Z_{max,tr}^{V}$$

$$0 = Z_{min,tr}^{L} - B_{tr}^{L} - aux_{tr}^{L}$$

$$0 = Z_{max,tr}^{V} - B_{tr}^{V} - aux_{tr}^{V}$$

$$0 = \sum_{i=1}^{NC} x_{tr,i} \cdot (a_{tr,i})^{0.5} \cdot \sum_{j=1}^{NC} 0.5 \cdot x_{tr,i=j} \cdot a_{EoS,tr,i=j} \cdot (a_{tr,i=j})^{-0.5} \cdot (1 - k_{i,j})$$
$$+ \sum_{i=1}^{NC} 0.5 \cdot x_{tr,i} \cdot a_{EoS,tr,i} \cdot (a_{tr,i})^{-0.5} \cdot \sum_{j=1}^{NC} x_{tr,i=j} \cdot (a_{tr,i=j})^{0.5} \cdot (1 - k_{i,j}) - a_{EoS,tr}^{L}$$

$$0 = \sum_{i=1}^{NC} y_{tr,i} \cdot (a_{tr,i})^{0.5} \cdot \sum_{j=1}^{NC} 0.5 \cdot y_{tr,i=j} \cdot a_{EoS,tr,i=j} \cdot (a_{tr,i=j})^{-0.5} \cdot (1 - k_{i,j})$$
$$+ \sum_{i=1}^{NC} 0.5 \cdot y_{tr,i} \cdot a_{EoS,tr,i} \cdot (a_{tr,i})^{-0.5} \cdot \sum_{j=1}^{NC} y_{tr,i=j} \cdot (a_{tr,i=j})^{0.5} \cdot (1 - k_{i,j}) - a_{EoS,tr}^{V}$$

$$0 = \sum_{i=1}^{NC} x_{tr,i} \cdot cp_{EoS,tr,i} - cp_{EoS,tr}^{L}$$

$$0 = \sum_{i=1}^{NC} y_{tr,i} \cdot cp_{EoS,tr,i} - cp_{EoS,tr}^{V}$$

Functions

$$a_{EoS} = \frac{-a \cdot (a0 + a1 \cdot \omega + a2 \cdot (\omega)^2)}{(\alpha \cdot T \cdot Tc)^{0.5}}$$

Applications:

- $a_{EoS,tr,i}(\alpha_{tr,i}, a0_{EoSalp}, a1_{EoSalp}, a2_{EoSalp}, \omega_i, T_{tr})$

$$h = (R \cdot T \cdot (Z-1) + \frac{T \cdot a_{EoS} - a}{b} \cdot ((2-n) \cdot \ln(1 + \frac{B}{Z}) + (n-1) \cdot \frac{1}{2 \cdot (2)^{0.5}} \cdot \ln(1 + \frac{2 \cdot (2)^{0.5}}{\frac{Z}{B} + 1 - (2)^{0.5}}))) \cdot (10)^{-6}$$

Applications:

- $h^V_{dep,tr}(B^V_{tr}, T_{tr}, Z^V_{max,tr}, a^V_{mix,tr}, a^V_{EoS,tr}, b^V_{mix,tr}, n)$
- $h^L_{dep,tr}(B^L_{tr}, T_{tr}, Z^L_{min,tr}, a^L_{mix,tr}, a^L_{EoS,tr}, b^L_{mix,tr}, n)$

$$\begin{aligned} cp_{EoS} = h^o + ((&a0 \cdot T + \frac{1}{2} \cdot a1 \cdot (T)^2 + \frac{1}{3} \cdot a2 \cdot (T)^3 + \frac{1}{4} \cdot a3 \cdot (T)^4 \\ &+ \frac{1}{5} \cdot a4 \cdot (T)^5 + \frac{1}{6} \cdot a5 \cdot (T)^6 + \frac{1}{7} \cdot a6 \cdot (10)^{-10} \cdot (T)^7 + \frac{1}{8} \cdot a7 \cdot (10)^{-10} \cdot (T)^8 \\ &+ \frac{1}{9} \cdot a8 \cdot (10)^{-20} \cdot (T)^9 + \frac{1}{10} \cdot a9 \cdot (10)^{-20} \cdot (T)^{10} + \frac{1}{11} \cdot a10 \cdot (10)^{-20} \cdot (T)^{11}) \\ &- (a0 \cdot T^o + \frac{1}{2} \cdot a1 \cdot (T^o)^2 + \frac{1}{3} \cdot a2 \cdot (T^o)^3 + \frac{1}{4} \cdot a3 \cdot (T^o)^4 + \frac{1}{5} \cdot a4 \cdot (T^o)^5 \\ &+ \frac{1}{6} \cdot a5 \cdot (T^o)^6 + \frac{1}{7} \cdot a6 \cdot (10)^{-10} \cdot (T^o)^7 + \frac{1}{8} \cdot a7 \cdot (10)^{-10} \cdot (T^o)^8 \\ &+ \frac{1}{9} \cdot a8 \cdot (10)^{-20} \cdot (T^o)^9 + \frac{1}{10} \cdot a9 \cdot (10)^{-20} \cdot (T^o)^{10} + \frac{1}{11} \cdot a10 \cdot (10)^{-20} \\ &\cdot (T^o)^{11})) \cdot (10)^{-6} \end{aligned}$$

Applications:

- $cp_{EoS,tr,i}(h^o_i, T^o, a1_{cp,i}, a2_{cp,i}, a3_{cp,i}, a4_{cp,i}, a5_{cp,i}, a6_{cp,i}, a7_{cp,i}, a8_{cp,i}, a9_{cp,i}, a10_{cp,i})$

$$h = h_{dep} + cp_{EoS}$$

Applications:

- $h^V_{tr}(cp^V_{EoS,tr}, h^V_{dep,tr})$
- $h^L_{tr}(cp^L_{EoS,tr}, h^L_{dep,tr})$

$$D = \frac{(q)^2}{4} + \frac{(p)^3}{27}$$

Applications:

- $D^V_{tr}(p^V_{tr},q^V_{tr})$
- $D^L_{tr}(p^L_{tr},q^L_{tr})$

$$\alpha = -1 + (n-1) \cdot B$$

Applications:

- $\alpha^V_{tr}(B^V_{tr},n)$
- $\alpha^L_{tr}(B^L_{tr},n)$

$$\gamma = -A \cdot B + (n-1) \cdot ((B)^2 + (B)^3)$$

Applications:

- $\gamma^V_{tr}(A^L_{tr},B^L_{tr},n)$
- $\gamma^L_{tr}(A^V_{tr},B^V_{tr},n)$

$$q = 2 \cdot \frac{(\alpha)^3}{27} - \frac{\beta \cdot \alpha}{3} + \gamma$$

Applications:

- $q^V_{tr}(\alpha^V_{tr},\beta^V_{tr},\gamma^V_{tr})$
- $q^L_{tr}(\alpha^L_{tr},\beta^L_{tr},\gamma^L_{tr})$

$$b = \frac{a0 \cdot R \cdot Tc}{Pc}$$

Applications:

- $b_i(a0_{EoSb}, Pc_i, R, Tc_i)$

$$\alpha = (1 + (a0 + a1 \cdot \omega + a2 \cdot (\omega)^2) \cdot (1 - (\frac{T}{Tc})^{0.5}))^2$$

Applications:

- $\alpha_{tr,i}(\omega_i, a0_{EoSalp}, a1_{EoSalp}, a2_{EoSalp}, T_{tr}, Tc_i)$

$$B = \frac{b \cdot p}{R \cdot T} \cdot (10)^5$$

Applications:

- $B^L_{tr}(T_{tr}, b^L_{mix,tr}, p_{tr}, R)$
- $B^V_{tr}(T_{tr}, b^V_{mix,tr}, p_{tr}, R)$

$$p = -\frac{(\alpha)^2}{3} + \beta$$

Applications:

- $p^L_{tr}(\alpha^L_{tr}, \beta^L_{tr})$
- $p^V_{tr}(\alpha^V_{tr}, \beta^V_{tr})$

$$\begin{aligned} lnphi = {} & (Z-1) \cdot \frac{b}{b_{mix}} - \ln(aux) \\ & + (\frac{b}{R \cdot T} \cdot (\frac{(a_{mix})^{0.5}}{b_{mix}} - \frac{(a)^{0.5}}{b})^2 - \frac{a}{b \cdot R \cdot T}) \cdot ((2-n) \cdot \ln(1 + \frac{B}{Z}) \\ & + (n-1) \cdot \frac{1}{2 \cdot (2)^{0.5}} \cdot \ln(1 + \frac{2 \cdot (2)^{0.5}}{\frac{Z}{B} + 1 - (2)^{0.5}})) \end{aligned}$$

Applications:

- $lnphi^L_{tr,i}(B^L_{tr}, T_{tr}, Z^L_{min,tr}, a_{tr,i}, a^L_{mix,tr}, b_i, b^L_{mix,tr}, n, R, aux)$
- $lnphi^V_{tr,i}(B^V_{tr}, T_{tr}, Z^V_{max,tr}, a_{tr,i}, a^V_{mix,tr}, b_i, b^V_{mix,tr}, n, R, aux)$

$$a = \frac{a0 \cdot (R)^2 \cdot (Tc)^2}{Pc} \cdot \alpha$$

Applications:

- $a_{tr,i}(a0_{EoSalp}, \alpha_{tr,i}, Pc_i, R, Tc_i)$

$$A = \frac{a \cdot p}{(R \cdot T)^2} \cdot (10)^5$$

Applications:

- $A^V_{tr}(T_{tr}, a^V_{mix,tr}, p_{tr}, R)$
- $A^L_{tr}(T_{tr}, a^L_{mix,tr}, p_{tr}, R)$

$$\beta = A - (B + 0.5)^2 + 0.25 - (n - 1) \cdot 2 \cdot ((B + 0.25)^2 - \frac{1}{16})$$

Applications:

- $\beta^V_{tr}(A^V_{tr}, B^V_{tr}, n)$
- $\beta^L_{tr}(A^L_{tr}, B^L_{tr}, n)$

Tab. B.37: Design variable specification.

Design Variable		Value	Engineering Unit
$F^F_{tr=1}$	=	75.0	$\mathrm{mol\,s^{-1}}$
$T_{tr=1}$	=	298.15	K
$h^F_{tr=1}$	=	-0.00257	$\mathrm{MJ\,mol^{-1}}$
$k_{i=1,j=1}$	=	0.0	−
$k_{i=1,j=2}$	=	0.0	−
$k_{i=1,j=3}$	=	0.0	−
$k_{i=1,j=4}$	=	0.39	−
$k_{i=1,j=5}$	=	0.0978	−
$k_{i=2,j=1}$	=	0.0	−
$k_{i=2,j=2}$	=	0.0	−
$k_{i=2,j=3}$	=	0.0	−
$k_{i=2,j=4}$	=	0.0	−
$k_{i=2,j=5}$	=	0.0	−

Design Variable		Value	Engineering Unit
$k_{i=3,j=1}$	=	0.0	−
$k_{i=3,j=2}$	=	0.0	−
$k_{i=3,j=3}$	=	0.0	−
$k_{i=3,j=4}$	=	0.0	−
$k_{i=3,j=5}$	=	0.08	−
$k_{i=4,j=1}$	=	0.39	−
$k_{i=4,j=2}$	=	0.0	−
$k_{i=4,j=3}$	=	0.0	−
$k_{i=4,j=4}$	=	0.0	−
$k_{i=4,j=5}$	=	−0.0133	−
$k_{i=5,j=1}$	=	0.0978	−
$k_{i=5,j=2}$	=	0.0	−
$k_{i=5,j=3}$	=	0.08	−
$k_{i=5,j=4}$	=	−0.0133	−
$k_{i=5,j=5}$	=	0.0	−
$p_{tr=1}$	=	35.0	bar
$y_{F,tr=1,i=1}$	=	0.0042649948	$\mathrm{mol\,mol^{-1}}$
$y_{F,tr=1,i=2}$	=	0.001113764794	$\mathrm{mol\,mol^{-1}}$
$y_{F,tr=1,i=3}$	=	0.079187738812	$\mathrm{mol\,mol^{-1}}$
$y_{F,tr=1,i=4}$	=	0.36066689299	$\mathrm{mol\,mol^{-1}}$
$y_{F,tr=1,i=5}$	=	0.5547666052	$\mathrm{mol\,mol^{-1}}$

Tab. B.38: Parameter specification.

Parameter		Value	Engineering Unit
$Pc_{i=1}$	=	4108000.0	Pa
$Pc_{i=2}$	=	3847270.0	Pa
$Pc_{i=3}$	=	4895000.0	Pa
$Pc_{i=4}$	=	1313000.0	Pa
$Pc_{i=5}$	=	4599000.0	Pa
R	=	8.314	$\mathrm{J\,mol^{-1}\,K^{-1}}$
T^o	=	298.15	K
$Tc_{i=1}$	=	591.75	K
$Tc_{i=2}$	=	780.0	K
$Tc_{i=3}$	=	562.05	K
$Tc_{i=4}$	=	33.19	K

Parameter		Value	Engineering Unit
$Tc_{i=5}$	=	190.56	K
$\omega_{i=1}$	=	0.264	−
$\omega_{i=2}$	=	0.3643	−
$\omega_{i=3}$	=	0.209	−
$\omega_{i=4}$	=	−0.215993	−
$\omega_{i=5}$	=	0.011	−
$a0_{EoSalp}$	=	0.48	−
$a0_{EoSa}$	=	0.42748	−
$a0_{EoSb}$	=	0.08664	−
$a0_{cp,i=1}$	=	90.437308284	$MJ\,mol^{-1}\,K^{-2}$
$a0_{cp,i=2}$	=	128.05624781	$MJ\,mol^{-1}\,K^{-2}$
$a0_{cp,i=3}$	=	51.129279111	$MJ\,mol^{-1}\,K^{-2}$
$a0_{cp,i=4}$	=	−1.5493755831	$MJ\,mol^{-1}\,K^{-2}$
$a0_{cp,i=5}$	=	33.026084022	$MJ\,mol^{-1}\,K^{-2}$
$a10_{cp,i=1}$	=	$3.8712156396E-8$	$MJ/mol/K^{12}$
$a10_{cp,i=2}$	=	0.0	$MJ/mol/K^{12}$
$a10_{cp,i=3}$	=	$-1.1438815377E-7$	$MJ/mol/K^{12}$
$a10_{cp,i=4}$	=	$-1.2743155754E-7$	$MJ/mol/K^{12}$
$a10_{cp,i=5}$	=	$3.8907702866E-8$	$MJ/mol/K^{12}$
$a1_{EoSalp}$	=	1.574	−
$a1_{cp,i=1}$	=	−0.9985041803	$MJ\,mol^{-1}\,K^{-3}$
$a1_{cp,i=2}$	=	−1.1747162507	$MJ\,mol^{-1}\,K^{-3}$
$a1_{cp,i=3}$	=	−0.42303172933	$MJ\,mol^{-1}\,K^{-3}$
$a1_{cp,i=4}$	=	0.48497196032	$MJ\,mol^{-1}\,K^{-3}$
$a1_{cp,i=5}$	=	0.0076633861482	$MJ\,mol^{-1}\,K^{-3}$
$a2_{EoSalp}$	=	−0.176	−
$a2_{cp,i=1}$	=	0.0080485154551	$MJ/mol/K^{4}$
$a2_{cp,i=2}$	=	0.0088938352242	$MJ/mol/K^{4}$
$a2_{cp,i=3}$	=	0.0030031574926	$MJ/mol/K^{4}$
$a2_{cp,i=4}$	=	−0.0036640321738	$MJ/mol/K^{4}$
$a2_{cp,i=5}$	=	$-9.1006983668E-5$	$MJ/mol/K^{4}$
$a3_{cp,i=1}$	=	$-2.5807841212E-5$	$MJ/mol/K^{5}$
$a3_{cp,i=2}$	=	$-2.2994177666E-5$	$MJ/mol/K^{5}$
$a3_{cp,i=3}$	=	$-3.3486993073E-6$	$MJ/mol/K^{5}$
$a3_{cp,i=4}$	=	$1.6281929297E-5$	$MJ/mol/K^{5}$
$a3_{cp,i=5}$	=	$7.2093965149E-8$	$MJ/mol/K^{5}$
$a4_{cp,i=1}$	=	$5.1269606228E-8$	$MJ/mol/K^{6}$

Parameter		Value	Engineering Unit
$a4_{cp,i=2}$	=	$3.3640587786E-8$	$MJ/mol/K^6$
$a4_{cp,i=3}$	=	$-1.0329470431E-8$	$MJ/mol/K^6$
$a4_{cp,i=4}$	=	$-4.5501303806E-8$	$MJ/mol/K^6$
$a4_{cp,i=5}$	=	$2.7753312144E-9$	$MJ/mol/K^6$
$a5_{cp,i=1}$	=	$-6.8621127767E-11$	$MJ/mol/K^7$
$a5_{cp,i=2}$	=	$-3.0247984794E-11$	$MJ/mol/K^7$
$a5_{cp,i=3}$	=	$4.0748420476E-11$	$MJ/mol/K^7$
$a5_{cp,i=4}$	=	$8.2690901469E-11$	$MJ/mol/K^7$
$a5_{cp,i=5}$	=	$-1.0507135092E-11$	$MJ/mol/K^7$
$a6_{cp,i=1}$	=	$6.3016209804E-4$	$MJ/mol/K^8$
$a6_{cp,i=2}$	=	$1.6555674018E-4$	$MJ/mol/K^8$
$a6_{cp,i=3}$	=	$-6.4864211744E-4$	$MJ/mol/K^8$
$a6_{cp,i=4}$	=	$-9.8950188959E-4$	$MJ/mol/K^8$
$a6_{cp,i=5}$	=	$1.798208819E-4$	$MJ/mol/K^8$
$a7_{cp,i=1}$	=	$-3.9191597331E-7$	$MJ/mol/K^9$
$a7_{cp,i=2}$	=	$-5.0634912115E-8$	$MJ/mol/K^9$
$a7_{cp,i=3}$	=	$5.853852096E-7$	$MJ/mol/K^9$
$a7_{cp,i=4}$	=	$7.7256546544E-7$	$MJ/mol/K^9$
$a7_{cp,i=5}$	=	$-1.7355522847E-7$	$MJ/mol/K^9$
$a8_{cp,i=1}$	=	1.5780545132	$MJ/mol/K^{10}$
$a8_{cp,i=2}$	=	0.066363537276	$MJ/mol/K^{10}$
$a8_{cp,i=3}$	=	−3.1168733123	$MJ/mol/K^{10}$
$a8_{cp,i=4}$	=	−3.783493389	$MJ/mol/K^{10}$
$a8_{cp,i=5}$	=	0.97650324341	$MJ/mol/K^{10}$
$a9_{cp,i=1}$	=	$-3.7117182927E-4$	$MJ/mol/K^{11}$
$a9_{cp,i=2}$	=	0.0	$MJ/mol/K^{11}$
$a9_{cp,i=3}$	=	$9.147761772E-4$	$MJ/mol/K^{11}$
$a9_{cp,i=4}$	=	0.0010542025496	$MJ/mol/K^{11}$
$a9_{cp,i=5}$	=	$-2.9979456966E-4$	$MJ/mol/K^{11}$
h^{sca}	=	1.0	−
$h^{o}_{i=1}$	=	0.05017	$MJ\,mol^{-1}$
$h^{o}_{i=2}$	=	0.182088	$MJ\,mol^{-1}$
$h^{o}_{i=3}$	=	0.08288	$MJ\,mol^{-1}$
$h^{o}_{i=4}$	=	0.0	$MJ\,mol^{-1}$
$h^{o}_{i=5}$	=	−0.07452	$MJ\,mol^{-1}$
n	=	1.0	−

Tab. B.39: Physical solution.

Iteration Variable		Value	Engineering Unit
$F^{L}_{tr=1}$	=	6.3601	$\mathrm{mol\,s^{-1}}$
$F^{V}_{tr=1}$	=	$6.8639E1$	$\mathrm{mol\,s^{-1}}$
$K_{tr=1,i=1}$	=	$2.045E-3$	–
$K_{tr=1,i=2}$	=	$5.032E-6$	–
$K_{tr=1,i=3}$	=	$6.399E-3$	–
$K_{tr=1,i=4}$	=	$6.900E1$	–
$K_{tr=1,i=5}$	=	1.025	–
Q	=	-2.604	$\mathrm{MJ\,s^{-1}}$
$Z^{V}_{max,tr=1}$	=	$1.367E-1$	–
$Z^{L}_{min,tr=1}$	=	$9.845E-1$	–
$a^{L}_{EoS,tr=1}$	=	$-4.304E-3$	$\mathrm{Pam^6/mol^2/K}$
$a^{V}_{EoS,tr=1}$	=	$-2.172E-4$	$\mathrm{Pam^6/mol^2/K}$
$a^{L}_{mix,tr=1}$	=	2.656	$\mathrm{Pam^6/mol^2}$
$a^{V}_{mix,tr=1}$	=	$9.553E-2$	$\mathrm{Pam^6/mol^2}$
$aux1^{L}_{Z,tr=1}$	=	$5.019E-1$	–
$aux1^{V}_{Z,tr=1}$	=	$3.512E-1$	–
$aux2^{L}_{Z,tr=1}$	=	$-6.985E-1$	–
$aux2^{V}_{Z,tr=1}$	=	$2.999E-1$	–
$aux^{L}_{tr=1}$	=	$2.220E-2$	–
$aux^{V}_{tr=1}$	=	$9.481E-1$	–
$b^{L}_{mix,tr=1}$	=	$8.111E-5$	$\mathrm{mol\,m^{-3}}$
$b^{V}_{mix,tr=1}$	=	$2.557E-5$	$\mathrm{mol\,m^{-3}}$
$cp^{L}_{EoS,tr=1}$	=	$7.288E-2$	$\mathrm{MJ\,mol^{-1}\,K^{-1}}$
$cp^{V}_{EoS,tr=1}$	=	$-4.430E-2$	$\mathrm{MJ\,mol^{-1}\,K^{-1}}$
$x_{tr=1,i=1}$	=	$4.920E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{tr=1,i=2}$	=	$1.313E-2$	$\mathrm{mol\,mol^{-1}}$
$x_{tr=1,i=3}$	=	$8.734E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{tr=1,i=4}$	=	$5.704E-3$	$\mathrm{mol\,mol^{-1}}$
$x_{tr=1,i=5}$	=	$5.860E-2$	$\mathrm{mol\,mol^{-1}}$
$y_{tr=1,i=1}$	=	$1.006E-4$	$\mathrm{mol\,mol^{-1}}$
$y_{tr=1,i=2}$	=	$6.607E-8$	$\mathrm{mol\,mol^{-1}}$
$y_{tr=1,i=3}$	=	$5.589E-3$	$\mathrm{mol\,mol^{-1}}$
$y_{tr=1,i=4}$	=	$3.936E-1$	$\mathrm{mol\,mol^{-1}}$
$y_{tr=1,i=5}$	=	$6.007E-1$	$\mathrm{mol\,mol^{-1}}$

B.6 Total Condenser

Tab. B.40: Notation, base names.

Base Name	Description	Engineering Unit
α	Wilson coefficient	–
γ	Activity coefficient	–
λ	Wilson coefficient	–
π	Mathematical constant pi	–
ρ	Molar density	$kmol\,m^{-3}$
A	Area ; Parameter	m^2, various
av	Auxiliary variable	various
B	Parameter	various
C	Parameter	various
cp	Specific heat capacity	$kJ\,kmol^{-1}\,K^{-1}$
D	Diameter; Parameter	m; various
d	Diameter	m
E	Parameter	various
F	Flow rate	$kmol\,s^{-1}$
H	Height; Parameter	m; various
h	Specific enthalpy	$kJ\,kmol^{-1}$
HU	Molar Holdup	kmol
K	Equilibrium constant	–
k	Heat transfer coefficient	$W\,K^{-1}\,m^{-2}$
M	Molar weight	$g\,mol^{-1}$
N	Count of items	–
p	Pressure	Pa
R	Ideal gas constant	$kJ\,kmol^{-1}\,K^{-1}$
T	Temperature	K
U	Internal energy	kJ
V	Volume	m^3
v	Molar volume	$m^3\,mol^{-1}$
x	Mole fraction liquid phase	$mol\,mol^{-1}$
y	Mole fraction vapor phase	$mol\,mol^{-1}$

Tab. B.41: Notation, superscripts.

Superscript	Description
aux	Auxiliary
c	Cold
cp	Heat capacity
crit	Critical
dim	Dimension
h	Hot; Enthalpy
i	Inner
in	Inlet
n	Molar
L	Liquid phase
LV	Liquid-vapor
o	Outter; Reference
out	Outlet
p	Pipe; Pressure
*ph*2	Two phase region
rho	Density
sc	Subcooled
sh	Superheated
total	Total
V	Vapor phase

Tab. B.42: Notation, subscripts.

Subscript	Description
cNP	Correlation for pipes

Tab. B.43: Notation, indices.

Index	Range	Description
i	$1 \ldots NC$	Component index
k	$1 \ldots NCk$	Coolant index

Tab. B.44: Component index.

Index	Component
$i = 1$	Ethanol
$i = 2$	Water
$k = 1$	Water

Equation System

$$0 = F^{c,sc,n,in} \cdot x_k^{c,sc,in} - F^{c,sc,n,out} \cdot x_k^{c,sc,out}$$

$$0 = F^{h,sc,n,in} \cdot x_i^{h,sc,in} - F^{h,sc,n,out} \cdot x_i^{h,sc,out}$$

$$0 = F^{c,ph2,n,in} \cdot x_k^{c,ph2,in} - F^{c,ph2,n,out} \cdot x_k^{c,ph2,out}$$

$$0 = F^{h,ph2,V,n,in} \cdot y_i^{h,ph2,in} + F^{h,ph2,L,n,in} \cdot x_i^{h,ph2,in} - F^{h,ph2,V,n,out} \cdot y_i^{h,ph2,out} - F^{h,ph2,L,n,out} \cdot x_i^{h,ph2,out}$$

$$0 = F^{c,ph2,n,in} \cdot x_k^{c,ph2,in} - F^{c,ph2,n,out} \cdot x_k^{c,ph2,out}$$

$$0 = F^{c,sh,n,in} \cdot x_k^{c,sh,in} - F^{c,sh,n,out} \cdot x_k^{c,sh,out}$$

$$0 = F^{h,sh,n,in} \cdot y_i^{h,sh,in} - F^{h,sh,n,out} \cdot y_i^{h,sh,out}$$

$$0 = K_i^{h,ph2,out} \cdot x_i^{h,ph2,out} - y_i^{h,ph2,out}$$

$$0 = K_i^{h,ph2,in} \cdot x_i^{h,ph2,in} - y_i^{h,ph2,in}$$

$$0 = \sum_{i=1}^{NC} x_i^{h,sc,out} - 1$$

$$0 = \sum_{k=1}^{NCk} x_k^{c,sc,out} - 1$$

$$0 = \sum_{i=1}^{NC} x_i^{h,ph2,in} - 1$$

$$0 = \sum_{k=1}^{NCk} x_k^{c,ph2,out} - 1$$

$$0 = \sum_{i=1}^{NC} x_i^{h,ph2,out} - 1$$

$$0 = \sum_{i=1}^{NC} x_i^{h,ph2,in} - 1$$

$$0 = \sum_{i=1}^{NC} y_i^{h,ph2,out} - 1$$

$$0 = \sum_{k=1}^{NCk} x_k^{c,sh,out} - 1$$

$$0 = \sum_{i=1}^{NC} y_i^{h,sh,out} - 1$$

$$0 = F^{c,sc,n,in} \cdot h^{c,sc,n,in} - F^{c,sc,n,out} \cdot h^{c,sc,n,out} + F^{h,sc,n,in} \cdot h^{h,sc,n,in} - F^{h,sc,n,out} \cdot h^{h,sc,n,out}$$

$$0 = k^{sc} \cdot A^{sc} \cdot \frac{(T^{h,sc,in} - T^{c,sc,out}) - (T^{h,sc,out} - T^{c,sc,in})}{\ln(\frac{(T^{h,sc,in} - T^{c,sc,out})}{(T^{h,sc,out} - T^{c,sc,in})})}$$
$$+ F^{c,sc,n,in} \cdot (\sum_{k=1}^{NCk} x_k^{c,sc,in} \cdot h_k^{c,sc,n,in}) - F^{c,sc,n,out} \cdot (\sum_{k=1}^{NCk} x_k^{c,sc,out} \cdot h_k^{c,sc,n,out})$$

$$0 = F^{n,L,h,ph2,in} \cdot h^{h,L,ph2,n,in} + F^{n,V,h,ph2,in} \cdot h^{h,V,ph2,n,in}$$
$$- F^{n,L,h,ph2,out} \cdot h^{h,L,ph2,n,out}$$
$$- F^{n,V,h,ph2,out} \cdot h^{h,V,ph2,n,out} + F^{c,ph2,n,in} \cdot h^{c,ph2,n,in} - F^{c,ph2,n,out} \cdot h^{c,ph2,n,out}$$

$$0 = -F^{c,ph2,n,in} \cdot (\sum_{k=1}^{NCk} x_k^{c,ph2,in} \cdot h_k^{c,sc,n,out}) + F^{c,ph2,n,out} \cdot (\sum_{k=1}^{NCk} x_k^{c,ph2,out} \cdot h_k^{c,ph2,n,out})$$
$$- k^{ph2} \cdot A^{ph2} \cdot \frac{(T^{h,ph2,in} - T^{c,ph2,out}) - (T^{h,ph2,out} - T^{c,ph2,in})}{\ln(\frac{(T^{h,ph2,in} - T^{c,ph2,out})}{(T^{h,ph2,out} - T^{c,ph2,in})})}$$

$$0 = F^{h,sh,n,in} \cdot h^{h,sh,n,in} - F^{h,sh,n,out} \cdot h^{h,sh,n,out} + F^{c,sh,n,in} \cdot h^{c,sh,n,in}$$
$$- F^{c,sh,n,out} \cdot h^{c,sh,n,out}$$

$$0 = F^{c,sh,n,out} \cdot (\sum_{k=1}^{NCk} x_k^{c,sh,out} \cdot h_k^{c,sh,n,L,out}) - F^{c,sh,n,in} \cdot (\sum_{k=1}^{NCk} x_k^{c,sh,in} \cdot h_k^{c,ph2,n,out})$$
$$- k^{sh} \cdot A^{sh} \cdot \frac{(T^{h,sh,in} - T^{c,sh,out}) - (T^{h,sh,out} - T^{c,sh,in})}{\ln(\frac{(T^{h,sh,in} - T^{c,sh,out})}{(T^{h,sh,out} - T^{c,sh,in})})}$$

$$0 = \sum_{k=1}^{NCk} x_k^{c,sc,in} \cdot h_k^{c,sc,n,in} - h^{c,sc,n,in}$$

$$0 = \sum_{k=1}^{NCk} x_k^{c,sc,out} \cdot h_k^{c,sc,n,out} - h^{c,sc,n,out}$$

$$0 = \sum_{k=1}^{NCk} x_k^{c,ph2,out} \cdot h_k^{c,ph2,n,out} - h^{c,ph2,n,out}$$

$$0 = \sum_{i=1}^{NC} y_i^{h,ph2,out} \cdot h_i^{n,h,V,ph2,out} - h^{n,h,V,ph2,out}$$

$$0 = \sum_{i=1}^{NC} x_i^{h,ph2,in} \cdot h_i^{n,h,L,ph2,in} - h^{n,h,L,ph2,in}$$

$$0 = \sum_{i=1}^{NC} x_i^{h,ph2,out} \cdot h_i^{n,h,L,ph2,out} - h^{n,h,L,ph2,out}$$

$$0 = \sum_{i=1}^{NC} y_i^{h,sh,in} \cdot h_i^{h,sh,n,in} - h^{h,sh,n,in}$$

$$0 = \sum_{i=1}^{NC} y_i^{h,sh,out} \cdot h_i^{h,sh,n,out} - h^{h,sh,n,out}$$

$$0 = \sum_{k=1}^{NCk} x_k^{c,sh,out} \cdot h_k^{c,sh,out,L,n} - h^{c,sh,n,out}$$

$$0 = T^{c,out,sc} - T^{c,in,sc} - av^{sc}$$

$$0 = T^{c,out,sh} - T^{c,out,ph2} - av^{sh}$$

$$0 = T^{c,out,ph2} - T^{c,out,sc} - av^{ph2}$$

$$0 = \frac{\sum_{i=1}^{NC} v_i^L - v_i^L}{v_i^L} \cdot \exp(\frac{-\lambda_i}{T})$$

$$0 = \frac{p_i^{LV}}{p} \cdot \gamma_i - K_i$$

$$0 = \frac{1}{x_i + \alpha_i \cdot (1 - x_i)} \cdot \exp((1 - x_i) \cdot (\frac{\alpha_i}{x_i + \alpha_i \cdot (1 - x_i)} - \frac{\sum_{i=1}^{NC} \alpha_i - \alpha_i}{(\sum_{i=1}^{NC} \alpha_i - \alpha_i) \cdot x_i + (1 - x_i)}))$$

$$0 = \sum_{i=1}^{NC} x_i^{h,sc,out} \cdot \rho_i^{L,h,sc,n,out} - \rho^{L,h,sc,n,out}$$

$$0 = \sum_{i=1}^{NC} x_i^{h,ph2,out} \cdot \rho_i^{L,h,ph2,n,out} - \rho^{L,h,ph2,n,out}$$

$$0 = \rho^{L,h,sc,n,out} \cdot V^{h,sc} - HU^{h,sc,n}$$

$$0 = \rho^{L,h,ph2,n,out} \cdot V^{ph2,L,h} - HU^{L,h,ph2,n}$$

$$0 = \frac{HU^{V,h,ph2,n} \cdot R \cdot T^{h,ph2,out}}{p^{h,ph2,out}} - V^{ph2,V,h}$$

$$0 = \frac{HU^{h,sh,n} \cdot R \cdot T^{h,sh,out}}{p^{h,sh,out}} - V^{h,sh}$$

$$0 = x_i^{h,sc,out} \cdot HU^{h,n,sc} - HU_i^{h,n,sc}$$

$$0 = x_i^{h,ph2,out} \cdot HU^{h,n,ph2,L} + y_i^{h,ph2,out} \cdot HU^{h,n,ph2,V} - HU_i^{h,n,ph2}$$

$$0 = y_i^{h,sh,out} \cdot HU^{h,n,sh} - HU_i^{h,n,sh}$$

$$0 = HU^{h,n,sc} \cdot (h^{h,sc,n,out} + \frac{p^{h,sc,out}}{\rho^{L,h,sc,n,out}}) - U^{h,sc}$$

$$0 = HU^{h,L,n,ph2} \cdot (h^{h,ph2,L,n,out} + \frac{p^{h,ph2,out}}{\rho^{L,h,ph2,n,out}}) + HU^{h,V,n,ph2} \cdot (h^{h,ph2,V,n,out} + R \cdot T^{h,ph2,out}) - U^{h,ph2}$$

$$0 = HU^{h,n,sh} \cdot (h^{h,sh,n,out} + R \cdot T^{h,sh,out}) - U^{h,sh}$$

$$0 = A^{sh} + A^{ph2} - A^{V}$$

$$0 = V^{h,sh} + V^{ph2,V,h} - V^{V,h}$$

$$0 = V^{h,sc} + V^{h,ph2,L} - V^{L}$$

$$0 = V^{L,total} + V^{V,total} - V$$

$$0 = A^{V} + A^{sc} - A$$

$$0 = 1 - \frac{2 \cdot H^{L,total}}{D^{i}} - A^{aux}$$

$$0 = N^{p,total} \cdot \pi \cdot d^{o} \cdot L - A$$

$$0 = ((\frac{D^{i}}{2})^{2} \cdot (\frac{\pi}{2} - A^{aux} - \frac{(A^{aux})^{3}}{6} - \frac{3 \cdot (A^{aux})^{5}}{40}) - (\frac{D^{i}}{2} - H^{L,total}) \cdot (D^{i} \cdot H^{L,total} - (H^{L,total})^{2})^{0.5}) \cdot L - V^{L,total}$$

$$0 = \frac{\pi \cdot (D^{i})^{2}}{4} \cdot L - V$$

$$0 = (N^{p,total} - N^{p,ph2,L} - N^{p,sc} - N^{p,sh}) \cdot \frac{\pi \cdot (d^{o})^{2}}{4} \cdot L - V^{p,ph2,V}$$

$$0 = V^{V,h,ph2} + V^{p,ph2,V} - V^{V,ph2,total}$$

$$0 = N^{p,ph2,L} \cdot \frac{\pi \cdot (d^{o})^{2}}{4} \cdot L - V^{p,ph2,L}$$

$$0 = V^{L,h,ph2} + V^{p,ph2,L} - V^{L,ph2,total}$$

$$0 = V^{sh,total} + V^{ph2,V,total} - V^{V,total}$$

$$0 = V^{sc,total} + V^{ph2,L,total} - V^{L,total}$$

$$0 = A_{cNP} \cdot \sin(B_{cNP} \cdot (\frac{H^{L,total} - \frac{D^{i}}{2}}{H^{dim}})) + \frac{N^{p,total}}{2} + C_{cNP} \cdot \exp(\frac{-H^{L,total}}{H^{dim}} \cdot 1000) - N^{p,sc} - N^{p,L,ph2}$$

$$0 = 1 - \frac{2 \cdot H}{D^{i}} - A^{aux}$$

$$0 = ((\frac{D^{i}}{2})^{2} \cdot (\frac{\pi}{2} - A^{aux} - \frac{(A^{aux})^{3}}{6} - \frac{3 \cdot (A^{aux})^{5}}{40}) - (\frac{D^{i}}{2} - H) \cdot (D^{i} \cdot H - (H)^{2})^{0.5}) \cdot L - V^{total}$$

$$0 = N^{p} \cdot \frac{\pi \cdot (d^{o})^{2}}{4} \cdot L - V^{p}$$

$$0 = V + V^{p} - V^{total}$$

$$0 = ((\frac{D^{i}}{2})^{2} \cdot (\frac{\pi}{2} - A^{aux} - \frac{(A^{aux})^{3}}{6} - \frac{3 \cdot (A^{aux})^{5}}{40}) - (\frac{D^{i}}{2} - H^{sh,total}) \cdot (D^{i} \cdot H^{sh,total} - (H^{sh,total})^{2})^{0.5}) \cdot L - V^{sh,total}$$

$$0 = 1 - \frac{2 \cdot H^{sh,total}}{D^{i}} - A^{aux,sh}$$

$$0 = (A_{cNP} \cdot sin(B_{cNP} \cdot (\frac{H^{sh,total} - \frac{D^{i}}{2}}{H^{dim}})) + \frac{N^{p,total}}{2} + C_{cNP} \cdot \exp(\frac{-H^{sh,total}}{H^{dim}} \cdot 1000)) - N^{p,sh}$$

$$0 = N^{p,sh} \cdot \pi \cdot d^{o} \cdot L - A^{sh}$$

$$0 = N^{p,sh} \cdot \frac{\pi \cdot (d^{o})^{2}}{4} \cdot L - V^{p,sh}$$

$$0 = V^{h,sh} + V^{p,sh} - V^{sh,total}$$

$$0 = ((\frac{D^{i}}{2})^{2} \cdot (\frac{\pi}{2} - A^{aux} - \frac{(A^{aux})^{3}}{6} - \frac{3 \cdot (A^{aux})^{5}}{40}) - (\frac{D^{i}}{2} - H^{sc,total}) \cdot (D^{i} \cdot H^{sc,total} - (H^{sc,total})^{2})^{0.5}) \cdot L - V^{sc,total}$$

$$0 = 1 - \frac{2 \cdot H^{sc,total}}{D^{i}} - A^{aux,sc}$$

$$0 = (A_{cNP} \cdot sin(B_{cNP} \cdot (\frac{H^{sc,total} - \frac{D^{i}}{2}}{H^{dim}})) + \frac{N^{p,total}}{2} + C_{cNP} \cdot \exp(\frac{-H^{sc,total}}{H^{dim}} \cdot 1000)) - N^{p,sc}$$

$$0 = N^{p,sc} \cdot \pi \cdot d^{o} \cdot L - A^{sc}$$

$$0 = N^{p,sc} \cdot \frac{\pi \cdot (d^{o})^{2}}{4} \cdot L - V^{p,sc}$$

$$0 = V^{h,sc} + V^{p,sc} - V^{sc,total}$$

$$0 = A_{cNP} \cdot sin(B_{cNP} \cdot (\frac{H^{L,total} - \frac{D^{i}}{2}}{H^{dim}})) + \frac{N^{p,total}}{2} + C_{cNP} \cdot \exp(\frac{-H^{L,total}}{H^{dim}} \cdot 1000) - N^{p,sc} - N^{L,p,ph2}$$

Functions

$$h(T) = h^{V,n} - h^{LV,n} + \frac{(p - p^{LV})}{\rho^{L,n}}$$

Applications:

- $h_i^{h,ph2,L,n,in}(\rho_i^{L,h,in,n}, h_i^{LV,ph2,in,n}, h_i^{V,h,in,ph2,n}, p^{h,in,ph2}, p_i^{ph2,LV,in})$
- $h_i^{h,ph2,L,n,out}(\rho_i^{L,h,out,n}, h_i^{LV,ph2,out,n}, h_i^{V,h,out,ph2,n}, p^{h,out,ph2}, p_i^{ph2,LV,out})$
- $h_i^{h,sc,L,n,out}(\rho_i^{L,h,out,n}, h_i^{LV,sc,out,n}, h_i^{V,h,out,sc,n}, p^{h,out,sc}, p_i^{sc,LV,out})$
- $h_k^{c,sc,n,in}(\rho_i^{L,c,in,n}, h_i^{LV,sc,in,n}, h_i^{V,c,in,sc,n}, p^{c,in,sc}, p_k^{sc,LV,in})$
- $h_k^{c,sc,n,out}(\rho_i^{L,c,out,n}, h_i^{LV,sc,out,n}, h_i^{V,c,out,sc,n}, p^{c,out,sc}, p_k^{sc,LV,out})$
- $h_k^{c,ph2,n,out}(\rho_i^{L,c,out,n}, h_i^{LV,ph2,out,n}, h_i^{V,c,out,ph2,n}, p^{c,out,ph2}, p_k^{ph2,LV,out})$
- $h_k^{c,sh,n,out}(\rho_i^{L,c,out,n}, h_i^{LV,sh,out,n}, h_i^{V,c,out,sh,n}, p^{c,out,sh}, p_k^{sh,LV,out})$

$$h(T) = h^o + A \cdot (T - T^{h,o}) + B \cdot \frac{((T)^2 - (T^{h,o})^2)}{2} + C \cdot \frac{((T)^3 - (T^{h,o})^3)}{3} + D \cdot \frac{((T)^4 - (T^{h,o})^4)}{4} + E \cdot \frac{((T)^5 - (T^{h,o})^5)}{5}$$

Applications:

- $h_i^{h,sh,n,in}(T^{h,sh,in}, h^o, T^{h,o}, A_i^{cp}, \ldots, E_i^{cp})$
- $h_i^{h,sh,n,out}(T^{h,sh,out}, h^o, T^{h,o}, A_i^{cp}, \ldots, E_i^{cp})$
- $h_i^{h,ph2,n,in}(T^{h,ph2,in}, h^o, T^{h,o}, A_i^{cp}, \ldots, E_i^{cp})$
- $h_i^{h,ph2,n,out}(T^{h,ph2,out}, h^o, T^{h,o}, A_i^{cp}, \ldots, E_i^{cp})$
- $h_i^{h,sc,n,out}(T^{h,sc,out}, h^o, T^{h,o}, A_i^{cp}, \ldots, E_i^{cp})$
- $h_k^{c,sc,n,in}(T^{c,sc,in}, h^o, T^{c,o}, A_k^{cp}, \ldots, E_k^{cp})$
- $h_k^{c,sc,n,out}(T^{c,sc,out}, h^o, T^{c,o}, A_k^{cp}, \ldots, E_k^{cp})$

- $h_k^{c,ph2,n,out}(T^{c,ph2,out}, h^o, T^{c,o}, A_k^{cp}, \ldots, E_k^{cp})$
- $h_k^{c,sh,n,out}(T^{c,sh,out}, h^o, T^{c,o}, A_k^{cp}, \ldots, E_k^{cp})$

$$h(T) = A \cdot (1 - \frac{T}{T^{crit}})^{B+C\cdot\frac{T}{T^{crit}}+D\cdot(\frac{T}{T^{crit}})^2+E\cdot(\frac{T}{T^{crit}})^3}$$

Applications:

- $h_i^{h,ph2,LV,n,in}(T^{h,ph2,in}, T_i^{crit}, A_i^{LV,h}, \ldots, E_i^{LV,h})$
- $h_i^{h,ph2,LV,n,out}(T^{h,ph2,out}, T_i^{crit}, A_i^{LV,h}, \ldots, E_i^{LV,h})$
- $h_i^{h,sc,LV,n,out}(T^{h,sc,out}, T_i^{crit}, A_i^{LV,h}, \ldots, E_i^{LV,h})$
- $h_i^{c,sc,LV,n,in}(T^{c,sc,in}, T_i^{crit}, A_i^{LV,h}, \ldots, E_i^{LV,h})$
- $h_i^{c,sc,LV,n,out}(T^{c,sc,out}, T_i^{crit}, A_i^{LV,h}, \ldots, E_i^{LV,h})$
- $h_i^{c,ph2,LV,n,out}(T^{c,ph2,out}, T_i^{crit}, A_i^{LV,h}, \ldots, E_i^{LV,h})$
- $h_i^{c,sh,LV,n,out}(T^{c,sh,out}, T_i^{crit}, A_i^{LV,h}, \ldots, E_i^{LV,h})$

$$\rho(T) = \frac{A}{(B)^{1+(1-\frac{T}{C})^D}}$$

Applications:

- $\rho_{i=1}^{L,h,sc,n,out}(T^{h,sc,out}, A_{i=1}^{L,n,rho}, \ldots, D_{i=1}^{L,n,rho})$
- $\rho_{i=1}^{L,h,ph2,n,in}(T^{h,ph2,in}, A_{i=1}^{L,n,rho}, \ldots, D_{i=1}^{L,n,rho})$
- $\rho_{i=1}^{L,h,ph2,n,out}(T^{h,ph2,out}, A_{i=1}^{L,n,rho}, \ldots, D_{i=1}^{L,n,rho})$

$$\rho(T) = A + B \cdot T + C \cdot (T)^2 + D \cdot (T)^3 + E \cdot (T)^4$$

Applications:

- $\rho_{i=2}^{L,h,sc,n,out}(T^{h,sc,out}, A_{i=2}^{L,n,rho}, \ldots, E_{i=2}^{L,n,rho})$
- $\rho_{i=2}^{L,h,ph2,n,in}(T^{h,ph2,in}, A_{i=2}^{L,n,rho}, \ldots, E_{i=2}^{L,n,rho})$
- $\rho_{i=2}^{L,h,ph2,n,out}(T^{h,ph2,out}, A_{i=2}^{L,n,rho}, \ldots, E_{i=2}^{L,n,rho})$
- $\rho_{k=1}^{L,c,sc,n,in}(T^{h,sc,in}, A_{k=1}^{L,n,rho}, \ldots, E_{k=1}^{L,n,rho})$
- $\rho_{k=1}^{L,c,sc,n,out}(T^{h,sc,out}, A_{k=1}^{L,n,rho}, \ldots, E_{k=1}^{L,n,rho})$
- $\rho_{k=1}^{L,c,sh,n,out}(T^{c,sh,out}, A_{k=1}^{L,n,rho}, \ldots, E_{k=1}^{L,n,rho})$
- $\rho_{k=1}^{L,c,ph2,n,out}(T^{h,ph2,out}, A_{k=1}^{L,n,rho}, \ldots, E_{k=1}^{L,n,rho})$

$$p(T) = \exp(A + \frac{B}{T} + C \cdot \ln(T) + D \cdot (T)^E)$$

Applications:

- $p_i^{LV,h,in,ph2}(T^{h,ph2,in}, A_i^{LV,p}, \ldots, E_i^{LV,p})$
- $p_i^{LV,h,out,ph2}(T^{h,ph2,out}, A_i^{LV,p}, \ldots, E_i^{LV,p})$
- $p_i^{LV,h,out,sc}(T^{h,sc,out}, A_i^{LV,p}, \ldots, E_i^{LV,p})$
- $p_k^{LV,c,in,sc}(T^{c,sc,in}, A_k^{LV,p}, \ldots, E_k^{LV,p})$
- $p_k^{LV,c,out,sc}(T^{c,sc,out}, A_k^{LV,p}, \ldots, E_k^{LV,p})$
- $p_k^{LV,c,out,ph2}(T^{c,ph2,out}, A_k^{LV,p}, \ldots, E_k^{LV,p})$
- $p_k^{LV,c,out,sh}(T^{c,sh,out}, A_k^{LV,p}, \ldots, E_k^{LV,p})$

Tab. B.45: Design variable specification.

Design Variable		Value	Engineering Unit
$F^{L,h,in,n,ph2}$	=	0.0	$\mathrm{mol\,s^{-1}}$
$F^{V,h,n,out,ph2}$	=	0.0	$\mathrm{mol\,s^{-1}}$
$F^{c,in,n,sc}$	=	5.2	$\mathrm{mol\,s^{-1}}$
$F^{h,in,n,sh}$	=	0.34	$\mathrm{mol\,s^{-1}}$

Design Variable		Value	Engineering Unit
$H^{L,total}$	=	0.05	m
H^{dim}	=	1.0	m
$T^{c,in,sc}$	=	298.15	K
$T^{h,in,sh}$	=	353.15	K
$\lambda_{wilson,i=1}$	=	95.68	K
$\lambda_{wilson,i=2}$	=	506.7	K
k^{ph2}	=	50.0	$\mathrm{W\,m^{-2}\,K^{-1}}$
k^{sc}	=	50.0	$\mathrm{W\,m^{-2}\,K^{-1}}$
k^{sh}	=	50.0	$\mathrm{W\,m^{-2}\,K^{-1}}$
$p^{c,in,sc}$	=	100000.0	Pa
$p^{c,out,ph2}$	=	100000.0	Pa
$p^{c,out,sc}$	=	100000.0	Pa
$p^{c,out,sh}$	=	100000.0	Pa
$p^{h,in,ph2}$	=	70000.0	Pa
$p^{h,out,ph2}$	=	70000.0	Pa
$p^{h,out,sc}$	=	70000.0	Pa
$p^{h,out,sh}$	=	70000.0	Pa
$v^{L}_{i=1}$	=	$5.869E-5$	$\mathrm{m^3\,mol^{-1}}$
$v^{L}_{i=2}$	=	$1.807E-5$	$\mathrm{m^3\,mol^{-1}}$
$x^{c,in,sc}_{k=1}$	=	1.0	$\mathrm{mol\,mol^{-1}}$
$y^{h,in,sh}_{i=1}$	=	0.412	$\mathrm{mol\,mol^{-1}}$
$y^{h,in,sh}_{i=2}$	=	0.588	$\mathrm{mol\,mol^{-1}}$

Tab. B.46: Parameter specification.

Parameter		Value	Engineering Unit
A_{cNP}	=	38.7993	−
$A^{cp}_{i=1}$	=	9.008	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$A^{L,n,rho}_{i=2}$	=	−13851.0	$\mathrm{mol\,m^{-3}}$
$A^{cp}_{i=2}$	=	32.22	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$A^{L,n,rho}_{k=1}$	=	−13851.0	$\mathrm{mol\,m^{-3}}$
$A^{cp}_{k=1}$	=	32.22	$\mathrm{J\,mol^{-1}}$
$A^{LV,p}_{i=1}$	=	73.304	−
$A^{LV,p}_{i=2}$	=	73.649	−
$A^{LV,p}_{k=1}$	=	73.649	−

Parameter		Value	Engineering Unit
$A_{i=1}^{L,n,rho}$	=	1628.8	$\mathrm{mol\,m^{-3}}$
$A_{i=1}^{LV,h}$	=	65831.0	$\mathrm{J\,mol^{-1}}$
$A_{i=2}^{LV,h}$	=	56600.0	$\mathrm{J\,mol^{-1}}$
$A_{k=1}^{LV,h}$	=	56600.0	$\mathrm{J\,mol^{-1}}$
B_{cNP}	=	7.2093	−
$B_{i=1}^{cp}$	=	0.2139	$\mathrm{J\,mol^{-1}\,K^{-2}}$
$B_{i=2}^{L,n,rho}$	=	640.38	$\mathrm{mol\,m^{-3}\,K^{-1}}$
$B_{i=2}^{cp}$	=	0.0019225	$\mathrm{J\,mol^{-1}\,K^{-2}}$
$B_{k=1}^{L,n,rho}$	=	640.38	$\mathrm{mol\,m^{-3}\,K^{-1}}$
$B_{k=1}^{cp}$	=	0.0019225	$\mathrm{J\,mol^{-1}\,K^{-2}}$
$B_{i=1}^{LV,p}$	=	−7122.3	K
$B_{i=2}^{LV,p}$	=	−7258.2	K
$B_{k=1}^{LV,p}$	=	−7258.2	K
$B_{i=1}^{L,n,rho}$	=	0.27469	−
$B_{i=1}^{LV,h}$	=	1.1905	−
$B_{i=2}^{LV,h}$	=	0.61204	−
$B_{k=1}^{LV,h}$	=	0.61204	−
C_{cNP}	=	1.1124	−
$C_{i=1}^{cp}$	=	$-8.3846E-5$	$\mathrm{J\,mol^{-1}\,K^{-3}}$
$C_{i=2}^{L,n,rho}$	=	−1.9124	$\mathrm{mol\,m^{-3}\,K^{-2}}$
$C_{i=2}^{cp}$	=	$1.0548E-5$	$\mathrm{J\,mol^{-1}\,K^{-3}}$
$C_{k=1}^{L,n,rho}$	=	−1.9124	$\mathrm{mol\,m^{-3}\,K^{-2}}$
$C_{k=1}^{cp}$	=	$1.0548E-5$	$\mathrm{J\,mol^{-1}\,K^{-3}}$
$C_{i=1}^{LV,p}$	=	−7.1424	−
$C_{i=2}^{LV,p}$	=	−7.3037	−
$C_{k=1}^{LV,p}$	=	−7.3037	−
$C_{i=1}^{L,n,rho}$	=	514.0	K
$C_{i=1}^{LV,h}$	=	−1.7666	−
$C_{i=2}^{LV,h}$	=	−0.6257	−
$C_{k=1}^{LV,h}$	=	−0.6257	−
D^{i}	=	0.298	m
$D_{i=1}^{cp}$	=	$1.3723E-9$	$\mathrm{J/mol/K^4}$
$D_{i=2}^{L,n,rho}$	=	0.0018211	$\mathrm{mol/m^3/K^3}$
$D_{i=2}^{cp}$	=	$-3.594E-9$	$\mathrm{J/mol/K^4}$
$D_{k=1}^{L,n,rho}$	=	0.0018211	$\mathrm{mol\,m^{-3}\,K^{-3}}$
$D_{k=1}^{cp}$	=	$-3.594E-9$	$\mathrm{J/mol/K^4}$

Parameter		Value	Engineering Unit
$D_{i=1}^{LV,p}$	=	$2.8853E-6$	$\mathrm{K}^{E_{\mathrm{i=1}}^{\mathrm{LV,p}}}$
$D_{i=2}^{LV,p}$	=	$4.1653E-6$	$\mathrm{K}^{E_{\mathrm{i=2}}^{\mathrm{LV,p}}}$
$D_{k=1}^{LV,p}$	=	$4.1653E-6$	$\mathrm{K}^{E_{\mathrm{k=1}}^{\mathrm{LV,p}}}$
$D_{i=1}^{L,n,rho}$	=	0.23178	–
$D_{i=1}^{LV,h}$	=	1.0012	–
$D_{i=2}^{LV,h}$	=	0.3988	–
$D_{k=1}^{LV,h}$	=	0.3988	–
$E_{i=1}^{cp}$	=	0.0	$\mathrm{J/mol/K^5}$
$E_{i=2}^{L,n,rho}$	=	0.0	$\mathrm{mol/m^3/K^4}$
$E_{i=2}^{cp}$	=	0.0	$\mathrm{J/mol/K^5}$
$E_{k=1}^{L,n,rho}$	=	0.0	$\mathrm{mol/m^3/K^4}$
$E_{k=1}^{cp}$	=	0.0	$\mathrm{J/mol/K^5}$
$E_{i=1}^{LV,p}$	=	2.0	–
$E_{i=2}^{LV,p}$	=	2.0	–
$E_{k=1}^{LV,p}$	=	2.0	–
$E_{i=1}^{LV,h}$	=	0.0	–
$E_{i=2}^{LV,h}$	=	0.0	–
$E_{k=1}^{LV,h}$	=	0.0	–
L	=	2.0	m
$N^{p,total}$	=	66.0	–
R	=	8.314	$\mathrm{J\,mol^{-1}\,K^{-1}}$
$T_{i=1}^{crit}$	=	514.0	K
$T_{i=1}^{h,o}$	=	298.15	K
$T_{i=2}^{crit}$	=	647.096	K
$T_{i=2}^{h,o}$	=	298.15	K
$T_{k=1}^{crit}$	=	647.096	K
$T_{k=1}^{h,o}$	=	298.15	K
π	=	3.14159265359	–
d^o	=	0.025	m
$h_{i=1}^o$	=	−234950.0	$\mathrm{J\,mol^{-1}}$
$h_{i=2}^o$	=	−241818.0	$\mathrm{J\,mol^{-1}}$
$h_{k=1}^o$	=	−241818.0	$\mathrm{J\,mol^{-1}}$

Tab. B.47: Physical solution.

Iteration Variable		Value	Engineering Unit
A	=	$1.037E1$	m^2
A^{V}	=	$1.008E1$	m^2
$A^{aux,sc}$	=	0.87594	m^2
$A^{aux,sh}$	=	0.9365	m^2
A^{aux}	=	$6.644E-1$	m^2
A^{ph2}	=	$1.005E1$	m^2
A^{sc}	=	$2.830E-1$	m^2
A^{sh}	=	$3.536E-2$	m^2
$F^{V,h,in,n,ph2}$	=	$3.400E-1$	$mol\,s^{-1}$
$F^{c,n,out,ph2}$	=	5.200	$mol\,s^{-1}$
$F^{c,n,out,sc}$	=	5.200	$mol\,s^{-1}$
$F^{c,n,out,sh}$	=	5.200	$mol\,s^{-1}$
$F^{h,in,n,sc}$	=	$3.400E-1$	$mol\,s^{-1}$
$F^{h,n,out,sc}$	=	$3.400E-1$	$mol\,s^{-1}$
$HU^{L,h,n,ph2}$	=	$1.679E2$	mol
$HU^{V,h,n,ph2}$	=	1.5087	mol
$HU^{h,n,sc}$	=	$1.470E2$	mol
$HU^{h,n,sh}$	=	$1.170E-1$	mol
$HU^{h,n,ph2}_{i=1}$	=	$7.013E1$	mol
$HU^{h,n,sc}_{i=1}$	=	$6.058E1$	mol
$HU^{h,n,sh}_{i=1}$	=	$4.823E-1$	mol
$HU^{h,n,ph2}_{i=2}$	=	$9.926E1$	mol
$HU^{h,n,sc}_{i=2}$	=	$8.646E1$	mol
$HU^{h,n,sh}_{i=2}$	=	$6.883E-1$	mol
$H^{sc,total}$	=	$1.942E-2$	m
$H^{sh,total}$	=	$9.449E-3$	m
$K^{h,in,ph2}_{i=1}$	=	4.800	–
$K^{h,out,ph2}_{i=1}$	=	1.548	–
$K^{h,in,ph2}_{i=2}$	=	0.6432	–
$K^{h,out,ph2}_{i=2}$	=	0.6162	–
$N^{L,p,ph2}$	=	5.798	–
$N^{p,sc}$	=	1.802	–
$N^{p,sh}$	=	$2.251E-1$	–
$T^{c,out,ph2}$	=	$3.353E2$	K

Iteration Variable		Value	Engineering Unit
$T^{c,out,sc}$	=	$2.995E2$	K
$T^{c,out,sh}$	=	$3.355E2$	K
$T^{h,in,ph2}$	=	$3.514E2$	K
$T^{h,in,sc}$	=	$3.442E2$	K
$T^{h,out,sc}$	=	$3.291E2$	K
$U^{h,ph2}$	=	$-4.699E4$	J
$U^{h,sc}$	=	$-4.108E4$	J
$U^{h,sh}$	=	$-2.733E1$	J
V	=	$1.395E-1$	m^3
$V^{L,h,ph2}$	=	$4.355E-3$	m^3
$V^{L,p,ph2}$	=	$5.692E-3$	m^3
$V^{L,ph2,total}$	=	$1.004E-2$	m^3
$V^{L,total}$	=	$1.559E-2$	m^3
V^{L}	=	$8.133E-3$	m^3
$V^{V,h,ph2}$	=	$5.711E-2$	m^3
$V^{V,h}$	=	$6.656E-2$	m^3
$V^{V,p,ph2}$	=	$5.711E-2$	m^3
$V^{V,ph2,total}$	=	$1.188E-1$	m^3
$V^{V,total}$	=	$1.239E-1$	m^3
$V^{h,sc}$	=	$3.777E-3$	m^3
$V^{h,sh}$	=	$4.886E-3$	m^3
$V^{p,sc}$	=	$1.769E-3$	m^3
$V^{p,sh}$	=	$2.221E-4$	m^3
$V^{sc,total}$	=	$5.546E-3$	m^3
$V^{sh,total}$	=	$5.107E-3$	m^3
$\alpha^{h,in,ph2}_{wilson,i=1}$	=	$2.345E-1$	–
$\alpha^{h,out,ph2}_{wilson,i=1}$	=	$2.332E-1$	–
$\alpha^{h,in,ph2}_{wilson,i=2}$	=	$7.680E-1$	–
$\alpha^{h,out,ph2}_{wilson,i=2}$	=	$7.452E-1$	–
$\gamma^{h,in,ph2}_{i=1}$	=	3.323	–
$\gamma^{h,out,ph2}_{i=1}$	=	1.433	–
$\gamma^{h,in,ph2}_{i=2}$	=	1.021	–
$\gamma^{h,out,ph2}_{i=2}$	=	1.322	–
$\rho^{L,h,n,out,ph2}$	=	$3.854E4$	$mol\,m^{-3}$
$\rho^{L,h,n,out,sc}$	=	$3.893E4$	$mol\,m^{-3}$
av^{ph2}	=	1.369	K

Iteration Variable		Value	Engineering Unit
av^{sc}	=	$3.574E1$	K
av^{sh}	=	$7.448E-2$	K
$h^{L,h,in,n,ph2}$	=	$-2.806E5$	$\mathrm{J\,mol^{-1}}$
$h^{V,h,in,n,ph2}$	=	$-2.364E5$	$\mathrm{J\,mol^{-1}}$
$h^{V,h,n,out,ph2}$	=	$-2.348E5$	$\mathrm{J\,mol^{-1}}$
$h^{c,in,n,sc}$	=	$-2.858E5$	$\mathrm{J\,mol^{-1}}$
$h^{c,n,out,ph2}$	=	$-2.830E5$	$\mathrm{J\,mol^{-1}}$
$h^{c,n,out,sc}$	=	$-2.857E5$	$\mathrm{J\,mol^{-1}}$
$h^{c,n,out,sh}$	=	$-2.830E5$	$\mathrm{J\,mol^{-1}}$
$h^{h,in,n,sc}$	=	$-2.778E5$	$\mathrm{J\,mol^{-1}}$
$h^{h,in,n,sh}$	=	$-2.363E5$	$\mathrm{J\,mol^{-1}}$
$h^{h,n,out,sc}$	=	$-2.793E5$	$\mathrm{J\,mol^{-1}}$
$x_{i=1}^{h,in,ph2}$	=	$8.584E-2$	$\mathrm{mol\,mol^{-1}}$
$x_{i=1}^{h,in,sc}$	=	$4.120E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{i=1}^{h,out,sc}$	=	$4.120E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{i=2}^{h,in,ph2}$	=	$9.142E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{i=2}^{h,in,sc}$	=	$5.880E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{i=2}^{h,out,sc}$	=	$5.880E-1$	$\mathrm{mol\,mol^{-1}}$
$x_{k=1}^{c,out,ph2}$	=	1.000	$\mathrm{mol\,mol^{-1}}$
$x_{k=1}^{c,out,sc}$	=	1.000	$\mathrm{mol\,mol^{-1}}$
$x_{k=1}^{c,out,sh}$	=	1.000	$\mathrm{mol\,mol^{-1}}$
$y_{i=1}^{h,in,ph2}$	=	$4.120E-1$	$\mathrm{mol\,mol^{-1}}$
$y_{i=1}^{h,out,ph2}$	=	$6.377E-1$	$\mathrm{mol\,mol^{-1}}$
$y_{i=2}^{h,in,ph2}$	=	$5.880E-1\ \mathrm{mol\,mol^{-1}}$	
$y_{i=2}^{h,out,ph2}$	=	$3.623E-1$	$\mathrm{mol\,mol^{-1}}$

B.7 Methanol and Water Column

A boiling, liquid mixture of $0.55\,\mathrm{mol\,mol^{-1}}$ methanol and $0.45\,\mathrm{mol\,mol^{-1}}$ water is separated by continuous distillation in this example. A total condenser and a partial reboiler are used. The column's trays are assumed to be well-mixed and the phases are in thermodynamic equilibrium. The equilibrium is approximated by Raoult's law. Enthalpies and vapor pressures are determined by the *Design*

***I**nstitute for **P**hysical **PR**operties* (DIPPR) equations (*DIPPR project 801: Physical and thermodynamic properties of pure chemicals, evaluated process design data* 1999) including the respective parameters. The reflux ratio and the reboiler's temperature are specified. Keeping the latter constant, four different numbers of trays (10, 20, 30, and 40) have been investigated regarding their change in process time with increasing size of the system. The dominating, complex subsystem of the NLE is rather large, e.g., 100 equations for the simulation with 10 trays. Nevertheless, compared to the other examples, it has a low nonlinearity ratio of 0.095. The equation system, its notation, variable and parameter specifications are shown next.

Tab. B.48: Notation, base names.

Base Name	**Description**	**Engineering Unit**
A	Parameter	various
B	Parameter	various
C	Parameter	various
D	Parameter	various
E	Parameter	various
F	Flow rate	$\mathrm{mol\,s^{-1}}$
K	Equilibrium constant	$-$
Q	Heat flux	$\mathrm{J\,s^{-1}}$
R	Reflux ratio	$-$
T	Temperature	K
X	Input parameter	various
Z	Output parameter	various
γ	Activity coefficient	$-$
φ	Fugacity coefficient	$-$
h	Molar enthalpy [	$\mathrm{J\,mol^{-1}}$
p	Pressure	Pa
Δp	Pressure drop	Pa
x	Liquid mole fraction	$\mathrm{mol\,mol^{-1}}$
y	Vapor mole fraction	$\mathrm{mol\,mol^{-1}}$

Tab. B.49: Notation, superscripts.

Superscript	Description
crit	Critical
f	Feed
L	Liquid
LV	Vapor liquid equilibrium
V	Vapor
h	Enthalpy
n	Molar
o	Reference
pLV	Vapor pressure
sca	Scaled

Tab. B.50: Notation, subscripts.

Subscript	Description
C	Condenser
R	Reboiler
*d*100	Dippr correlation 100
*d*101	Dippr correlation 101
*d*106	Dippr correlation 106

Tab. B.51: Notation, indices.

Index	Range	Description
i	$1 \ldots NC$	Component index
tr	$1 \ldots NTR$	Tray index

Tab. B.52: Component index.

Index	Component
$i = 1$	Methanol
$i = 2$	Water

Equation System - Tray Section

$$0 = F_{tr}^{f,L,n} \cdot x_{tr,i}^{f} + F_{tr+1}^{L,n} \cdot x_{tr+1,i} + F_{tr-1}^{V,n} \cdot y_{tr-1,i} - F_{tr}^{L,n} \cdot x_{tr,i} - F_{tr}^{V,n} \cdot y_{tr,i}$$

$$0 = \frac{\gamma_{tr,i} \cdot x_{tr,i} \cdot p_{tr,i}^{LV}}{p^{sca}} - \frac{\varphi_{tr,i}^{V} \cdot p_{tr} \cdot y_{tr,i}}{p^{sca}}$$

$$0 = \sum_{i=1}^{NC} x_{tr,i} - 1$$

$$0 = \sum_{i=1}^{NC} y_{tr,i} - 1$$

$$0 = \frac{F_{tr}^{f,L,n} \cdot h^{f,L,n} + F_{tr+1}^{L,n} \cdot h_{tr+1}^{L,n} + F_{tr-1}^{V,n} \cdot h_{tr-1}^{V,n} - F_{tr}^{L,n} \cdot h_{tr}^{L,n} - F_{tr}^{V,n} \cdot h_{tr}^{V,n}}{h^{sca}}$$

$$0 = \frac{\sum_{i=1}^{NC} \left(x_{tr,i} \cdot h_{tr,i}^{L,n} \right)}{h^{sca}} - \frac{h_{tr}^{L,n}}{h^{sca}}$$

$$0 = \frac{\sum_{i=1}^{NC} \left(y_{tr,i} \cdot \left(h_{tr,i}^{L,n} + h_{tr,i}^{LV,n} \right) \right)}{h^{sca}} - \frac{h_{tr}^{V,n}}{h^{sca}}$$

$$0 = \frac{\sum_{i=1}^{NC} x_{i}^{f} \cdot h_{i}^{f,L,n}}{h^{sca}} - \frac{h^{f,L,n}}{h^{sca}}$$

$$0 = p_{tr-1} - p_{tr} - \Delta p$$

Equation System - Partial Reboiler

$$0 = F_{tr=1}^{L,n} \cdot x_{tr=1,i} - F_{R}^{L,n} \cdot x_{R,i} - F_{tr=0}^{V,n} \cdot y_{tr=0,i}$$

$$0 = \frac{x_{R,i} \cdot p_{R,i}^{LV} \cdot \gamma_{R,i}}{p^{sca}} - \frac{y_{tr=0,i} \cdot p_{tr=0}}{p^{sca}}$$

$$0 = \sum_{i=1}^{NC} x_{R,i} - 1$$

$$0 = \sum_{i=1}^{NC} y_{tr=0,i} - 1$$

$$0 = \frac{F_{tr=1}^{L,n} \cdot h_{tr=1}^{L,n} - F_{R}^{L,n} \cdot h_{R}^{L,n} - F_{tr=0}^{V,n} \cdot h_{tr=0}^{V,n} + Q_{R}}{h^{sca}}$$

$$0 = \frac{\sum_{i=1}^{NC} x_{R,i} \cdot h_{R,i}^{L,n}}{h^{sca}} - \frac{h_{R}^{L,n}}{h^{sca}}$$

$$0 = \frac{\sum_{i=1}^{NC} y_{R,i} \cdot \left(h_{R,i}^{L,n} + h_{R,i}^{LV,n} \right)}{h^{sca}} - \frac{h_{tr=0}^{V,n}}{h^{sca}}$$

Equation System - Total Condenser

$$
\begin{aligned}
0 &= F^{V,n}_{tr=NTR} \cdot y_{tr=NTR,i} - (F^{L,n}_{C} + F^{L,n}_{tr=NTR+1}) \cdot x_{tr=NTR+1,i} \\
0 &= \frac{x_{tr=NTR+1,i} \cdot p^{LV}_{C,i} \cdot \gamma_{C,i}}{p^{sca}} - \frac{y_{C,i} \cdot p_{tr=NTR+1}}{p^{sca}} \\
0 &= R \cdot F^{L,n}_{C} - F^{L,n}_{tr=NTR+1} \\
0 &= \sum_{i=1}^{NC} x_{tr=NTR+1,i} - 1 \\
0 &= \sum_{i=1}^{NC} y_{C,i} - 1 \\
0 &= \frac{F^{V,n}_{tr=NTR} \cdot h^{V,n}_{tr=NTR} - (F^{L,n}_{C} + F^{L,n}_{tr=NTR+1}) \cdot h^{L,n}_{tr=NTR+1} + Q_C}{h^{sca}} \\
0 &= \frac{\sum_{i=1}^{NC} x_{C,i} \cdot h^{L,n}_{C,i}}{h^{sca}} - \frac{h^{L,n}_{C}}{h^{sca}}
\end{aligned}
$$

Functions

$$
Z(X) = \exp(A^{pLV}_{d101} + \frac{B^{pLV}_{d101}}{X} + C^{pLV}_{d101} \cdot \ln(X) + D^{pLV}_{d101} \cdot (X)^{E^{pLV}_{d101}})
$$

Applications:

- $p^{LV}_{C,i}(T_C, A^{pLV}_{d101,i}, \ldots, E^{pLV}_{d101,i})$
- $p^{LV}_{tr,i}(T_{tr}, A^{pLV}_{d101,i}, \ldots, E^{pLV}_{d101,i})$
- $p^{LV}_{R,i}(T_R, A^{pLV}_{d101,i}, \ldots, E^{pLV}_{d101,i})$

$$
\begin{aligned}
Z(X) = A^{L,h,n}_{d100} \cdot (X - T^{h,o}) + \frac{B^{L,h,n}_{d100}}{2} \cdot ((X)^2 - (T^{h,o})^2) + \frac{C^{L,h,n}_{d100}}{3} \cdot ((X)^3 - (T^{h,o})^3) \\
+ \frac{D^{L,h,n}_{d100}}{4} \cdot ((X)^4 - (T^{h,o})^4) + \frac{E^{L,h,n}_{d100}}{5} \cdot ((X)^5 - (T^{h,o})^5) + h^{L,o,n}
\end{aligned}
$$

Applications:

- $h^{L,n}_{C,i}(T_C), A^{L,h,n}_{d100,i}, \ldots, E^{L,h,n}_{d100,i})$

- $h_{tr,i}^{L,n}(T_{tr}), A_{d100,i}^{L,h,n}, \ldots, E_{d100,i}^{L,h,n})$
- $h_{R,i}^{L,n}(T_R), A_{d100,i}^{L,h,n}, \ldots, E_{d100,i}^{L,h,n})$

$$Z(X) = A_{d106}^{LV,h,n} \cdot (1 - \frac{X}{T^{crit}})^{B_{d106}^{LV,h,n} + C_{d106}^{LV,h,n} \cdot \frac{X}{T^{crit}} + D_{d106}^{LV,h,n} \cdot (\frac{X}{T^{crit}})^2 + E_{d106}^{LV,h,n} \cdot (\frac{X}{T^{crit}})^3}$$

Applications:

- $h_{tr,i}^{LV,n}(T_{tr}, A_{d106,i}^{LV,h,n}, \ldots, E_{d106,i}^{LV,h,n})$
- $h_{R,i}^{LV,n}(T_R), A_{d106,i}^{LV,h,n}, \ldots, E_{d106,i}^{LV,h,n})$

Because the number of design specifications and solution values are numerous, their presentation is omitted here. They are part of the evaluation (ID: 166690) in MOSAICmodeling and can be viewed in the software's simulation editor.

B.8 Heavies Column

The notation is the same as already introduced for the model *Partial Condenser* in table B.32 - B.36.

Equation System - Tray Section (reformulated)

$$0 = \sum_{s=tr}^{NST} (F_{tr=s}^{F} \cdot x_{F,tr=s,i}) + F_{tr-1}^{V} \cdot y_{tr-1,i} - F^{D} \cdot x_{tr=N+1,i} - F_{tr}^{L} \cdot x_{tr,i}$$

$$0 = K_{tr,i} \cdot x_{tr,i} - y_{tr,i}$$

$$0 = \sum_{i=1}^{NC} x_{tr,i} - 1$$

$$0 = \sum_{i=1}^{NC} y_{tr,i} - 1$$

$$0 = \frac{\sum_{s=tr}^{NST} (F_{tr=s}^{F} \cdot h_{tr=s}^{F}) - F^{D} \cdot h_{tr=N+1}^{L} + F_{tr-1}^{V} \cdot h_{tr-1}^{V} - F_{tr}^{L} \cdot h_{tr}^{L} + Q + Q^{C}}{h^{sca}}$$

$$0 = \frac{p_{tr-1}}{p^{sca}} - \frac{p_{tr}}{p^{sca}} - \frac{\Delta p}{p^{sca}}$$

Equation System - Tray Section (classical)
The equation system equals the reformulated version except for the component balances referring to the first equation and the energy balance referring to the fifth equation that are replaced by

$$0 = F^F_{tr} \cdot x_{F,tr,i} + F^V_{tr-1} \cdot y_{tr-1,i} + F^L_{tr+1} \cdot x_{tr+1,i} - F^V_{tr} \cdot y_{tr,i} - F^L_{tr} \cdot x_{tr,i}$$
$$0 = \frac{F^F_{tr} \cdot h^L_{F,tr} + F^V_{tr-1} \cdot h^V_{tr-1} + F^L_{tr+1} \cdot h^L_{tr+1} - F^V_{tr} \cdot h^V_{tr} - F^L_{tr} \cdot h^L_{tr} + Q}{h^{sca}}$$

Equation System - Partial Reboiler (reformulated)

$$0 = F^B \cdot ((y_{tr=0,i} - x^R_i) - (R^R + 1) \cdot (y_{tr=0,i} - x_{tr=0,i}))$$
$$0 = K^R_i \cdot x^R_i - y_{tr=0,i}$$
$$0 = \sum_{i=1}^{NC} x^R_i - 1$$
$$0 = \sum_{i=1}^{NC} y_{tr=0,i} - 1$$
$$0 = \frac{F^B \cdot ((h^V_{tr=0} - h^{R,L}) - (R^R + 1) \cdot (h^V_{tr=0} - h^L_{tr=1})) + Q^R}{h^{sca}}$$

Equation System - Partial Reboiler (classical)
The equation system equals the reformulated version except for the component balances referring to the first equation and the energy balance referring to the fifth equation that are replaced by

$$0 = F^L_{tr=1} \cdot x_{tr=1,i} - F^B \cdot x_{R,i} - F^V_{tr=0} \cdot y_{tr=0,i}$$
$$0 = \frac{F^L_{tr=1} \cdot h^L_{tr=1} - F^B \cdot h^{R,L} - F^V_{tr=0} \cdot h^V_{tr=0} + Q^R}{h^{sca}}$$

Equation System - Total Condenser (reformulated)
Component balances and reflux ratio are applied in functions

$$0 = K_i^C \cdot x_{tr=NTR+1,i} - y_i^C$$
$$0 = \sum_{i=1}^{NC} y_i^C - 1$$
$$0 = \frac{F_{tr=NTR}^V \cdot (h_{tr=NTR}^V - h_{tr=NTR+1}^L) + Q^C}{h^{sca}}$$

Equation System - Total Condenser (classical)

$$0 = F_{tr=NTR}^V \cdot y_{tr=NTR,i} - (F^D + F_{tr=NTR+1}^{L,n}) \cdot x_{tr=NTR+1,i}$$
$$0 = K_i^C \cdot x_{tr=NTR+1,i} - y_i^C$$
$$0 = \sum_{i=1}^{NC} y_{C,i} - 1$$
$$0 = \sum_{i=1}^{NC} x_{tr=NTR+1,i} - 1$$
$$0 = \frac{F_{tr=NTR}^V \cdot (h_{tr=NTR}^V - h_{tr=NTR+1}^L) + Q^C}{h^{sca}}$$

Enthalpies and the equilibrium constants are equally determined as in *Partial Condenser*. Hence, the related equations and functions are not listed here.

Additional Functions (reformulated)
Next to thermophysical properties calculations, functions are also used in the reformulated equation system to explicitly calculate the connecting streams between tray section and reboiler and tray section and condenser. The reason is described in section 4.5.

$$y(x) = x$$

Applications:

– $y_{tr=NTR,i}(x_{tr=NTR+1,i})$

$$A(B,R) = B \cdot (1 + R)$$

Applications:

– $F^{V}_{tr=NTR}(F^{D},R^{C})$

$$A(B,R) = B \cdot R$$

Applications:

– $F^{L}_{tr=NTR+1}(F^{D},R^{C})$

– $F^{V}_{tr=0}(F^{B},R^{B})$

Because the number of design specifications and solution values are numerous, their presentation is omitted here. They are part of the evaluation (ID: 164598) in MOSAICmodeling and can be viewed in the software's simulation editor.

Appendix C

Computational Experiments

C.1 Contraction

Tab. C.1: Results combined contraction methods: HC4revise (hc), Interval Newton (n) and Bnormal with tighten_bounds (bctb), initialized with $\boldsymbol{x}^{(0)} = [-10^9, 10^9]^{n_{var}}$. The following settings were applied: resolution = 8, $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$.

Contraction /	**hc_n_bctb**		**hc_n**	
NLE	ε^{RADL}	**CPU (s)**	$\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**CPU (s)**
CSTR	1.61×10^{-8}	0.30	1.14×10^{-4}	0.19
Flash Unit	4.40×10^{-1}	0.85	0.0	0.72
Total Condenser	1.58×10^{-1}	27.42	1.82×10^{0}	4.16
Heavies Column	7.08×10^{-1}	449.50	2.98×10^{-10}	83.09

Tab. C.2: Results combined contraction methods: HC4revise (hc) and Bnormal with tighten_bounds (bctb) and without (bc), initialized with $\boldsymbol{x}^{(0)} = [-10^9, 10^9]^{n_{var}}$. The following settings were applied: resolution = 8, $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$.

Contraction / NLE	**hc_bctb** $\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**hc_bctb CPU (s)**	**hc_bc** $\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**hc_bc CPU (s)**
CSTR	0.0	0.27	0.0	0.26
Flash Unit	0.0	0.80	0.0	0.76
Total Condenser	-2.97×10^{-11}	26.67	4.23×10^{-5}	10.59
Heavies Column	0.0	423.59	0.0	98.46

Tab. C.3: Results combined contraction methods: Interval Newton (n) and Bnormal with tighten_bounds (bctb) and without (bc), initialized with $\boldsymbol{x}^{(0)} = [-10^9, 10^9]^{n_{var}}$. The following settings were applied: resolution = 8, $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$.

Contraction / NLE	**n_bctb** $\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**n_bctb CPU (s)**	**n_bc** $\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**n_bc CPU (s)**
CSTR	3.08×10^{-3}	1.31	3.08×10^{-3}	0.22
Flash Unit	9.52×10^{-9}	9.22	9.52×10^{-9}	6.65
Total Condenser	1.18×10^{0}	391.37	1.20×10^{0}	140.74
Heavies Column	3.80×10^{-1}	302.75	3.80×10^{-1}	76.84

Tab. C.4: Results single contraction methods: Interval Newton (n) and Bnormal with tighten_bounds (bctb), initialized with $\boldsymbol{x}^{(0)} = [-10^9, 10^9]^{n_{var}}$. The following settings were applied: resolution = 8, $\varepsilon^{Rel} = 10^{-3}$, $\varepsilon^{Abs} = 10^{-8}$.

Contraction / NLE	**n** ε^{RADL}	**n CPU (s)**	**bctb** $\frac{\varepsilon^{RADL} - \varepsilon^{RADL}_{hc\_n\_bctb}}{\varepsilon^{RADL}_{hc\_n\_bctb}}$	**bctb CPU (s)**
CSTR	1.0	0.21	3.08×10^{-3}	1.15
Flash Unit	1.0	0.28	9.52×10^{-3}	6.57
Total Condenser	1.0	2.75	1.18×10^{0}	332.44
Heavies Column	1.0	59.91	3.80×10^{-1}	278.66

C.2 Global Solver

Tab. C.5: Adjusted initialization of NLEs.

NLE	Iteration Variables	Initial Interval	Unit
Reactive	Activity coefficients	$[0, 10^9]$	–
Flash	Enthalpies	$[0,10^9]$	$\mathrm{J\,mol^{-1}}$
Unit	Equilibrium constant	$[0, 10^9]$	–
	Flow rates	$[0, 10^9]$	$\mathrm{mol\,s^{-1}}$
	Heat flow rate	$[0, 10^9]$	$\mathrm{J\,s^{-1}}$
	Mole fractions	$[0, 1]$	–
	Reaction constant and rate	$[0, 10^9]$	$\mathrm{s^{-1}}$
Total	Temperatures	[298.15, 353.15]	K
Condenser	Temperature differences	[0.01, 100]	K
	Activity coefficients	$[0, 10^9]$	–
	Wilson's coefficients	$[0, 10^9]$	–
	Equilibrium constants	$[0, 10^9]$	–
	Flow rates	$[0, 10^9]$	$\mathrm{mol\,s^{-1}}$
	Internal Energies	$[-10^9,0]$	MJ
	Enthalpies	$[-10^6,0]$	$\mathrm{J\,mol^{-1}}$
	Densities	$[0, 10^9]$	$\mathrm{mol\,m^{-3}}$
	Mole fractions	$[0, 1]$	–
	Molar hold ups	$[0, 10^9]$	mol
	Heights, Areas, Volumes	$[0, 10^9]$	m, $\mathrm{m^2}$,$\mathrm{m^3}$
Partial	a,b of cubic equation	$[0, 10]$	$\mathrm{Pam^6/mol^2}$,
Condenser			$\mathrm{mol\,m^{-3}}$
	Auxiliary variables	$[10^{-6}, 10^9]$	–
	Compressibility factors	$[0, 1]$	–
	Equilibrium constants	$[0, 10^9]$	–

NLE	Iteration Variables	Initial Interval	Unit
	Flow rates	$[0, 75]$	$\mathrm{mol\,s^{-1}}$
	Heat flux	$[-100, 100]$	$\mathrm{J\,s^{-1}}$
	Mole fractions	$[0, 1]$	–
	SRK coefficients	$[-1, 10]$	–
Methanol	Condenser duty	$[-10^6, 0]$	$\mathrm{J\,s^{-1}}$
Water/	Enthalpies	$[-10^6,0]$	$\mathrm{J\,mol^{-1}}$
Column	Flow rates	$[0, 10]$	$\mathrm{mol\,s^{-1}}$
	Mole fractions	$[0, 1]$	–
	Pressures	$[10^3,10^6]$	Pa
	Reboiler duty	$[0, 10^6]$	$\mathrm{J\,s^{-1}}$
	Temperatures	$[240, 400]$	K
Heavies	a of cubic equation	$[0, 10]$	$\mathrm{Pam^6/mol^2}$
Column	a_{EoS} of cubic equation	$[-1, 0]$	–
	Auxiliary variables	$[10^{-6}, 10^9]$	–
	b of cubic equation	$[0\ 1]$	$\mathrm{mol\,m^{-3}}$
	Condenser duty	$[-1, 0]$	$\mathrm{MJ\,s^{-1}}$
	Equilibrium constants	$[0, 10]$	–
	Flow rates	$[0, 30]$	$\mathrm{mol\,s^{-1}}$
	Integrated heat capacities	$[0, 0.5]$	$\mathrm{MJ\,mol^{-1}}$
	Mole fractions	$[0, 1]$	–
	Pressures	[0.0,10]	bar
	Reboiler duty	$[0, 1]$	$\mathrm{MJ\,s^{-1}}$
	Temperatures	[350.0, 400.0]	K
	θ of cubic equation	$[0, 0.3]$	–

C.3 Local Solver

Tab. C.6: Initial value selection for *Heavies Column* that result in feasible initial point. If not explicitly stated it is used for all tray and component indices..

Iteration Variables	**Initial Value**	**Unit**
a of cubic equation	1.1	$\mathrm{Pam^6/mol^2}$
a_{EoS} of cubic equation	-0.001	–
Auxiliary variable	0.5	–
b of cubic equation	0.0001	$\mathrm{mol\,m^{-3}}$
Condenser duty	-1.0	$\mathrm{MJ\,s^{-1}}$
Equilibrium constants	$K_{i=1} = K_{i=2} = 0.5$	–
	$K_{i=3} = K_{i=4} = K_{i=5} = 1.1$	–
Flow rates	1.5	$\mathrm{mol\,s^{-1}}$
Integrated heat capacities	0.05	$\mathrm{MJ\,mol^{-1}}$
Mole fractions	0.33	–
Pressures	1.5	bar
Reboiler duty	1.0	$\mathrm{MJ\,s^{-1}}$
Temperatures	$T_{tr} = 371 + (NTR - tr)$	K
	$T^C = 370$, $T^R = 376$	K
θ of cubic equation	0.1	–

Appendix D

Conclusion

D.1 Application area

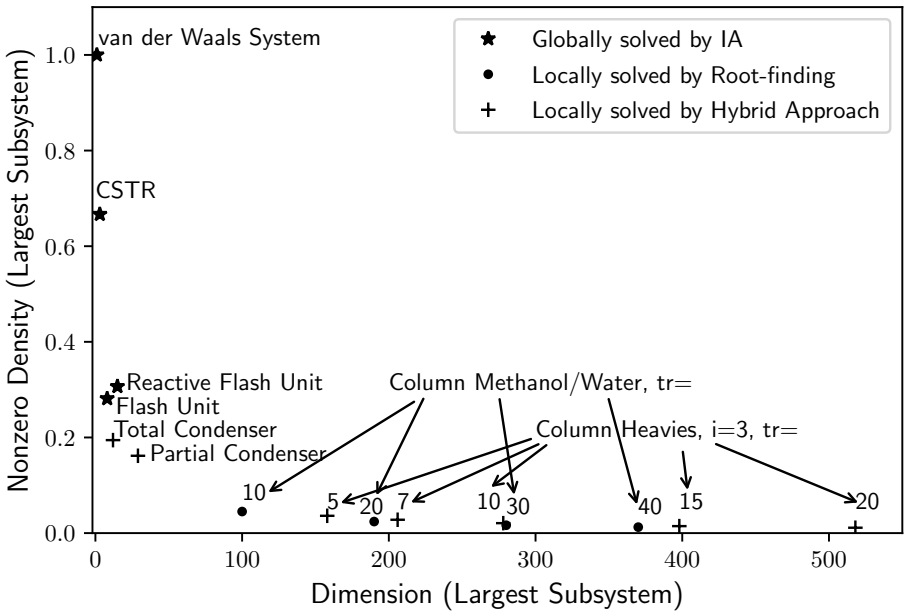


Fig. D.1: Nonzero ratio versus dimension of largest subsystem for the tested examples.

D.2 Guidelines for initialization

Tab. D.1: Recommended initialization for geometric and hydraulic quantities.

Variable	**Initial range**	**Condition**
Diameter, length, height	≥ 0	$-$
Partial area	$[0,\ A^{total}]$	Known total area A^{total}
Partial holdup	$[0,\ HU^{total}]$	Known total holdup HU^{total}
Partial volume	$[0,\ V^{total}]$	Known total volume V^{total}
Mole, volume, weight fractions	[0,1]	$-$
Molar, mass outlet flow rates	$[0,\ \sum_{st=1}^{NST} F_{st}^{in}]$	Steady-state, flow-driven process known NST inlet flow rates F_{st}^{in}

Tab. D.2: Recommended initialization for MESH related quantities.

Variable	**Initial range**	**Condition**
Mole, volume, weight fractions	$[0,1]$	$-$
Molar, mass outlet flow rates	$[0,\ \sum_{st=1}^{NST} F_{st}^{in}]$	Steady-state, flow-driven process, known NST inlet flow rates F_{st}^{in}
Distillation, internal molar, mass flow rates (liquid)	$[0,\ \frac{\sum_{st=1}^{NST} F_{st}^{in}}{R^C}]$	Steady-state, known inlet flow rates, known reflux ratio
Distillation, internal molar, mass flow rates (vapor)	$[0,\ \frac{\sum_{st=1}^{NST} F_{st}^{in}}{R^R}]$	Steady-state, known inlet flow rates, known boil-up ratio
Heat rates	≤ 0	Exothermic processes
Heat rates	≥ 0	Endothermic processes

Tab. D.3: Recommended initialization for thermophysical properties.

Variable	**Initial range**	**Condition**
Molar enthalpies	$[\min(h_{0,i}), \max(h_{0,i})]$	Ideal mixture
Molar volumes (liquid)	$[\min(v_{0,i}), \max(v_{0,i})]$	Ideal mixture
Molar volumes (gas, vapor)	$[\frac{R \cdot \underline{T}}{\overline{p}}, \frac{R \cdot \overline{T}}{\underline{p}}]$	Ideal gas, known, estimated $\underline{\overline{T}}$, $\underline{\overline{p}}$
VLE, temperatures	$[\min(T_{0,i}^{LV}), \max(T_{0,i}^{LV})]$	Zeotropic mixture
VLE, pressures	$[\min(p_{0,i}^{LV}), \max(p_{0,i}^{LV})]$	Zeotropic mixture
VLE, equilibrium constants	$[0, 1]$	Heavy-boiling component
VLE, equilibrium constants	≥ 1	Light-boiling component
VLE, equilibrium constants	≥ 0	Medium-boiling component, unkown boiling temperature

D.3 Error in the Solution Algorithm

An error in the solution algorithm can be found systematically via successive debugging of the code. In small systems this can be done efficiently via the debug mode of Python editors such as spyder from Raybaut (2009). However, this kind of debugging is slower than the normal execution of the program and thus extremely time-consuming for large systems. Hence, a debug function is provided in the hybrid approach, which prints parameters describing the state of the program to the console during program execution. For example, the ID of the variable being currently reduced. In this way, unwanted infinite loops could be identified and removed from the program, which caused the program to hang during certain variable reductions.

D.4 Things that did not work out

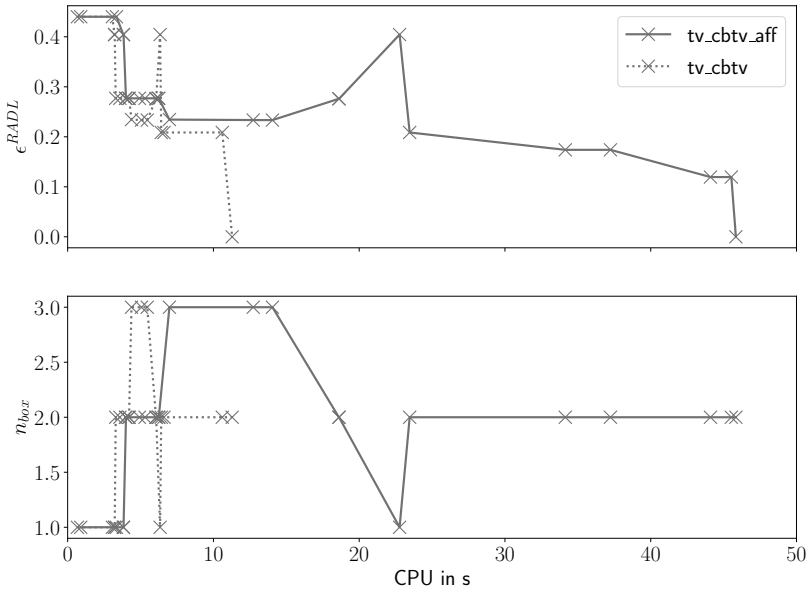


Fig. D.2: Comparison of hybrid approach applied on Flash Unit with and without affine arithmetic.

List of References

Alefeld, G., J. Herzberger (1970): ALGOL-60 Algorithmen zur Auflösung linearer Gleichungssysteme mit Fehlererfassung. *Computing* 6 (1), 28–34. ISSN: 1436-5057. DOI: 10.1007/BF02241730 (cit. on p. 46).

Mc-Allester, D., P. van Hentenryck, D. Kapur (1995): *Three Cuts for Accelerated Interval Propagation*. Ed. by MIT. URL: http://dspace.mit.edu/bitstream/handle/1721.1/6642/AIM-1542.pdf?sequence=2 (last access 01/05/2023) (cit. on p. 53).

Amarger, R. J., L. T. Biegler, I. E. Grossmann (1992): An automated modelling and reformulation system for design optimization. *Computers & Chemical Engineering* 16 (7), 623–636. ISSN: 00981354. DOI: 10.1016/0098-1354(92)80011-W (cit. on p. 3).

Anderson, D. G. (1965): Iterative Procedures for Nonlinear Integral Equations. *Journal of the ACM* 12 (4), 547–560. ISSN: 0004-5411. DOI: 10.1145/321296.321305 (cit. on p. 17).

Andersson, J. A. E., J. Gillis, G. Horn, J. B. Rawlings, M. Diehl (2019): CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation* 11 (1), 1–36. DOI: 10.1007/s12532-018-0139-4 (cit. on p. 83).

Armijo, L. (1966): Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics* 16 (1), 1–3. ISSN: 0030-8730. DOI: 10.2140/pjm.1966.16.1 (cit. on p. 22).

Asaithambi, N. S., R. E. Moore, S. Zuhe (1982): On computing the range of values. *Computing* 28 (3), 225–237. ISSN: 1436-5057. DOI: 10.1007/BF02241750 (cit. on p. 55).

Asprion, N., M. Bortz (2018): Process Modeling, Simulation and Optimization: From Single Solutions to a Multitude of Solutions to Support Decision Making. *Chemie Ingenieur Technik* 90 (11), 1727–1738. ISSN: 0009286X. DOI: 10.1002/cite.201800051 (cit. on pp. 1, 2).

Baharev, A. (2016): A robust approach for finding all well-separated solutions of sparse systems of nonlinear equations. *Numerical algorithms*, 1–27. ISSN: 1572-9265. DOI: 10.1007/s11075-016-0249-x (cit. on pp. 5, 37).

Baharev, A., L. Kolev, E. Rév (2011): Computing multiple steady states in homogeneous azeotropic and ideal two-product distillation. *AIChE Journal* 57 (6), 1485–1495. ISSN: 00011541. DOI: 10.1002/aic.12362 (cit. on p. 6).

Benhamou, F., J.-F. Puget (1999): Revising hull and box consistency. In: *Logic Programming: Proceedings of the 1999 International Conference on Logic Programming*. Ed. by D. de Schreye. MIT Press, 230–244. ISBN: 9780262291118. DOI: 10.7551/mitpress/4304.003.0024 (cit. on pp. 51–53, 60).

Biegler, L. T. (2010): *Nonlinear programming: Concepts, algorithms, and applications to chemical processes*. MOS-SIAM Series on optimization. Philadelphia: SIAM. ISBN: 9780898717020. DOI: 10.1137/1.9780898719383 (cit. on pp. 13, 14, 26).

Biegler, L. T. (2014): Recent Advances in Chemical Process Optimization. *Chemie Ingenieur Technik* 86 (7), 943–952. ISSN: 0009286X. DOI: 10.1002/cite.201400033 (cit. on p. 1).

Biegler, L. T., I. E. Grossmann, A. W. Westerberg (1999): *Systematic Methods of Chemical Process Design*. Physical and Chemical Engineering Sciences. New Jersey: Prentice Hall. ISBN: 0134924223 (cit. on p. 4).

Biegler, L. T., D. C. Miller, C. O. Okoli (2022): Don't search—Solve! Process optimization modeling with IDAES. In: *Simulation and Optimization in Process Engineering: The Benefit of Mathematical Methods in Applications of the Chemical Industry*. Elsevier, 33–55. ISBN: 9780323850438. DOI: 10.1016/B978-0-323-85043-8.00005-2 (cit. on p. 6).

Bliek, C. (1992): Computer methods for design automation. *Ph.D. Thesis*. Boston: MIT. URL: http://hdl.handle.net/1721.1/35361 (last access 08/22/2021) (cit. on p. 46).

Bortz, M., N. Asprion (2022): *Simulation and Optimization in Process Engineering: The Benefit of Mathematical Methods in Applications of the Chemical Industry*. San Diego: Elsevier. ISBN: 9780323850445 (cit. on p. 1).

Boston, J. F., S. L. Sullivan (1974): A New Class of Solution Methods for Multicomponent, Multistage Separation Processes. *The Canadian Journal of Chemical Engineering* 52 (1), 52–63. ISSN: 00084034. DOI: 10.1002/cjce.5450520108 (cit. on p. 5).

Boyd, S., L. Vandenberghe (2013): *Convex Optimization*. Cambridge University Press. ISBN: 9780521833783. DOI: 10.1017/CBO9780511804441 (cit. on p. 35).

Bröcker, S., R. Benfer, M. Bortz, S. Engell, C. Knösche, A. Kröner (2021): Process Simulation - Fit for the future? Position paper of the ProcessNet working Commitee Process Simulation, Process Synthesis and Knowledge Processing (cit. on p. 1).

Broyden, C. G. (1965): A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation* 19 (92), 577–593. ISSN: 1088-6842. DOI: 10.1090/S0025-5718-1965-0198670-6 (cit. on p. 18).

Bublitz, S., E. Esche, J.-U. Repke (2021b): Automatic Initial Value Generation with Interval Arithmetic for Nonlinear Process Models. *Computers & Chemical Engineering* 151, 107342. DOI: 10.1016/j.compchemeng.2021.107342 (cit. on pp. 70, 151, 155).

Bublitz, S., E. Esche, G. Tolksdorf, V. Mehrmann, J.-U. Repke (2017a): Analysis and Decomposition for Improved Convergence of Nonlinear Process Models in Chemical Engineering. *Chemie Ingenieur Technik* 89 (11), 1503–1514. ISSN: 0009286X. DOI: 10.1002/cite.201700041 (cit. on pp. 5, 35–37, 84, 144).

Collavizza, H., F. Delobel, M. Rueher (1998): Comparing Partial Consistencies. *Developments in Reliable Computing, International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics*. Ed. by T. Csendes. Szeged: Springer, 213–228. DOI: 10.1023/A:1009922003700 (cit. on p. 51).

Conn, A. R., N. I. M. Gould, P. L. Toint (2000): *Trust-region Methods*. MOS-SIAM Series on optimization. SIAM. ISBN: 978-0-89871-460-9. DOI: 10.1137/1.9780898719857 (cit. on p. 22).

Curtis, A., J. K. Reid (1972): On the Automatic Scaling of Matrices for Gaussian Elimination. *IMA Journal of Applied Mathematics* 10 (1), 118–124. ISSN: 0272-4960. DOI: 10.1093/imamat/10.1.118 (cit. on p. 32).

Dahmen, W., A. Reusken (2008): *Numerik für Ingenieure und Naturwissenschaftler*. 2. ed. Springer-Lehrbuch. Berlin, Heidelberg: Springer. ISBN: 9783540764939. DOI: 10.1007/978-3-540-76493-9 (cit. on pp. 18, 26–28, 107).

Dennis, J. E., R. B. Schnabel (1996): *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM. ISBN: 978-0-89871-364-0. DOI: 10.1137/1.9781611971200 (cit. on pp. 3, 18, 21).

Desrochers, B. (2015): *pyIbex*. URL: https://pypi.org/project/pyibex (last access 09/16/2021) (cit. on p. 62).

DIPPR project 801: Physical and thermodynamic properties of pure chemicals, evaluated process design data (1999). Washington: Taylor & Francis (cit. on p. 227).

Duff, I. S., A. M. Erisman, J. K. Reid (2017a): *Direct Methods for Sparse Matrices*. 2. ed. Numerical mathematics and scientific computation. Oxford: Oxford University Press. ISBN: 9780191746420. DOI: 10.1093/acprof:oso/9780198508380.001.0001 (cit. on pp. 4, 27, 33, 35).

Duff, I. S., A. M. Erisman, J. K. Reid (2017b): The Hellerman-Rarick algorithm. *Rutherford Appleton Laboratory Technical Reports*. DOI: 10.5286/raltr.2017003 (cit. on pp. 36, 37).

Dulmage, A. L., N. S. Mendelsohn (1958): Coverings of Bipartite Graphs. *Canadian Journal of Mathematics* 10, 517–534. ISSN: 0008-414X. DOI: 10.4153/CJM-1958-052-0 (cit. on p. 33).

Dutton, R. A., W. Mao, J. Chen, W. Watson (2008): Parallel Job Scheduling with Overhead: A Benchmark Study. *International Conference on Networking, Architecture, and Storage*. IEEE, 326–333. ISBN: 978-0-7695-3187-8. DOI: 10.1109/NAS.2008.26 (cit. on p. 102).

Ebert, F. (2021): Untersuchung verschiedener Intervall-Arithmetik basierter Methoden zur Reduktion des Lösungsraumes von nichtlinear algebraischen Gle-

ichungssystemen. *Master Thesis*. Berlin: Technische Universität Berlin (cit. on pp. 73, 84, 85, 87, 103).

Erisman, A. M., R. G. Grimes, J. G. Lewis, J. W. G. Poole (1985): A Structurally Stable Modification of Hellerman–Rarick's P4 Algorithm for Reordering Unsymmetric Sparse Matrices. *Journal on Numerical Analysis* 22 (2), 369–385. ISSN: 0036-1429. DOI: 10.1137/0722022 (cit. on p. 36).

Figueiredo, L. H. d., R. J. van Iwaarden, J. Stolfi (1997): Fast Interval Branch-And-Bound Methods For Unconstrained Global Optimization With Affine Arithmetic (cit. on p. 6).

Fourer, R., D. M. Gay, B. W. Kernighan (2009): *AMPL: A mMdeling Language for Mathematical Programming*. 2. ed., 5. print. Belmont: DuxburyPr. ISBN: 0534388094 (cit. on p. 38).

Friday, J. R., B. D. Smith (1964): An analysis of the equilibrium stage separations problem—formulation and convergence. *AIChE Journal* 10 (5), 698–707. ISSN: 00011541. DOI: 10.1002/aic.690100524 (cit. on pp. 5, 160).

Gander, W., M. J. Gander, F. Kwok (2014): *Scientific computing: An introduction using Maple and Matlab*. Vol. 11. Texts in computational science and engineering. Cham: Springer. ISBN: 9783319043258. DOI: 10.1007/978-3-319-04325-8 (cit. on p. 29).

Gmehling, J., B. Kolbe (1992): *Thermodynamik*. 2. ed. Weinheim: VCH. ISBN: 3527285474 (cit. on pp. 111, 112).

Grossmann, I. E., A. W. Westerberg (2000): Research Challenges in Process Systems Engineering. *AIChE Journal* 46 (9), 1700–1703. ISSN: 00011541. DOI: 10.1002/aic.690460902 (cit. on p. 3).

Guddat, J., F. Guerra Vazquez, H. T. Jongen (1990): *Parametric optimization: Singularities, pathfollowing and jumps*. Stuttgart: Teubner. ISBN: 3519021129 (cit. on p. 4).

Hammersley, J. M., D. C. Handscomb (1964): *Monte Carlo Methods*. Dordrecht: Springer. ISBN: 978-94-009-5821-0. DOI: 10.1007/978-94-009-5819-7 (cit. on p. 150).

Hangos, K. M., I. T. Cameron (2001): *Process modelling and model analysis*. Vol. 4. Process Systems Engineering. San Diego: Academic Press. ISBN: 9780121569310 (cit. on p. 155).

Hansen, E. R. (1992): Bounding the Solution of Interval Linear Equations. *Journal on Numerical Analysis* 29 (5), 1493–1503. DOI: 10.1137/0729086 (cit. on p. 46).

Hansen, E. R. (2000): The Hull of Preconditioned Interval Linear Equations. *Reliable Computing* 6 (2), 95–103. ISSN: 1573-1340. DOI: 10.1023/A:1009962903365 (cit. on p. 46).

Hansen, E. R., R. I. Greenberg (1983): An interval Newton method. *Applied Mathematics and Computation* 12 (2-3), 89–98. ISSN: 0096-3003. DOI: 10.1016/0096-3003(83)90001-2 (cit. on pp. 55, 56).

Hansen, E. R., S. Sengupta (1981): Bounding solutions of systems of equations using interval analysis. *BIT Numerical Mathematics* 21 (2), 203–211. DOI: 10.1007/BF01933165 (cit. on p. 46).

Hansen, E. R., R. Smith (1967): Interval Arithmetic in Matrix Computations, Part II. *Journal on Numerical Analysis* 4 (1), 1–9. DOI: 10.1137/0704001 (cit. on p. 46).

Hansen, E. R., G. W. Walster (2003): *Global Optimization Using Interval Analysis: Revised And Expanded*. 2nd ed. Hoboken: Taylor and Francis. ISBN: 9780824740597 (cit. on pp. 4, 12, 46, 49, 53, 54, 77, 80).

Hansen, N. (2016): *The CMA Evolution Strategy: A Tutorial*. DOI: 10.48550/arXiv.1604.00772 (cit. on p. 150).

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. Del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant (2020): Array programming with NumPy. *Nature* 585 (7825), 357–362. DOI: 10.1038/s41586-020-2649-2 (cit. on p. 77).

Heath, M. T. (2002): *Scientific computing: An introductory survey*. 2. ed. Boston: McGraw-Hill. ISBN: 0072399104 (cit. on p. 17).

Hellerman, E., D. Rarick (1971): Reinversion with the Preassigned Pivot Procedure. *Mathematical Programming* 1 (1), 195–216. ISSN: 0025-5610. DOI: 10.1007/BF01584086 (cit. on p. 36).

Henley, E. J., J. D. Seader (1981): *Equilibrium-Stage Separation Operations in Chemical Engineering*. New York: John Wiley & Sons, Inc. ISBN: 0471371084 (cit. on p. 116).

HSL (2022): *Fortran: A collection of Fortran Codes for Large Scale Scientific Computation*. URL: http://www.hsl.rl.ac.uk (last access 01/13/2022) (cit. on p. 85).

Huss, R. S., F. Chen, M. F. Malone, M. F. Doherty (2003): Reactive distillation for methyl acetate production. *Computers & Chemical Engineering* 27 (12), 1855–1866. ISSN: 00981354. DOI: 10.1016/S0098-1354(03)00156-X (cit. on p. 191).

Johansson, F. (2010): *mpmath: a Python library for arbitrary-precision floating-point arithmetic*. URL: https://mpmath.org/ (last access 08/22/2021) (cit. on p. 39).

Kearfott, R. B. (1990): Preconditioners for the Interval Gauss–Seidel Method. *Journal on Numerical Analysis* 27 (3), 804–822. DOI: 10.1137/0727047. (Last access 08/27/2021) (cit. on p. 48).

Knight, P. A., D. Ruiz, B. Uçar (2014): A Symmetry Preserving Algorithm for Matrix Scaling. *Journal on Matrix Analysis and Applications* 35 (3), 931–955. ISSN: 0895-4798. DOI: 10.1137/110825753 (cit. on p. 32).

Kraft, D. (1988): *A Software Package for Sequential Quadratic Programming*. Oberpfaffenhofen (cit. on p. 83).

Krawczyk, R. (1969): Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing* 4 (3), 187–201. ISSN: 1436-5057. DOI: 10.1007/BF02234767 (cit. on p. 47).

Levenberg, K. (1944): A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics* 2 (2), 164–168. ISSN: 1552-4485. DOI: 10.1090/qam/10666 (cit. on p. 20).

Lhomme, O. (1993): Consistency Techniques for Numeric CSPs. 13th International Joint Conference on Artifical Intelligence. San Francisco: Morgan Kaufmann Publishers, 232–238 (cit. on pp. 49, 55, 56).

Liu, S. (2017): Chapter 15 - Sustainability and Stability. In: *Bioprocess Engineering*. Ed. by S. Liu. 2. ed. Elsevier, 871–947. ISBN: 978-0-444-63783-3. DOI: 10.1016/B978-0-444-63783-3.00015-0 (cit. on pp. 109, 110).

Marquardt, D. W. (1963): An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics* 11 (2), 431–441. ISSN: 0368-4245. DOI: 10.1137/0111030 (cit. on p. 20).

Martin, M., R. Gani, I. M. Mujtaba (2022): Sustainable process synthesis, design, and analysis: Challenges and opportunities. *Sustainable Production and Consumption* 30, 686–705. ISSN: 23525509. DOI: 10.1016/j.spc.2022.01.002 (cit. on p. 1).

Mazumder, S., ed. (2016): *Numerical Methods for Partial Differential Equations*. Academic Press. ISBN: 978-0-12-849894-1 (cit. on p. 3).

McKay, M. D., R. J. Beckman, W. J. Conover (1979): A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* 21 (2), 239. ISSN: 00401706. DOI: 10.2307/1268522 (cit. on p. 150).

McKerns, M. M., L. Strand, T. Sullivan, A. Fang, M. A. G. Aivazis (2012): *Building a Framework for Predictive Science*. DOI: 10.48550/arXiv.1202.1056 (cit. on p. 102).

Moore, R. E. (1979): *Methods and Applications of Interval Analysis*. SIAM. ISBN: 978-0-89871-161-5. DOI: 10.1137/1.9781611970906 (cit. on p. 56).

Moore, R. E., R. Baker Kearfott, M. J. Cloud (2009): *Introduction to Interval Analysis*. Philadelphia: SIAM. ISBN: 0898716691. DOI: 10.1137/1.9780898717716 (cit. on pp. 38, 39, 43, 44, 56, 139).

Neumaier, A. (1990): *Interval methods for systems of equations*. Vol. 37. Encyclopedia of Mathematics and its Applications. Cambridge and New York: Cambridge

University Press. ISBN: 0511526474. DOI: 10.1017/CBO9780511526473 (cit. on pp. 45, 47).

Neumaier, A. (1999): A simple Derivation of the Hansen-Bliek-Rohn-Ning-Kearfott Enclosure for Linear Interval Equations. *Reliable Computing* 5 (2), 131–136. ISSN: 1573-1340. DOI: 10.1023/A:1009997221089 (cit. on p. 46).

Neumaier, A. (2000): Erratum to: A Simple Derivation of the Hansen-Bliek-Rohn-Ning-Kearfott Enclosure for Linear Interval Equations (Reliable Computing 5(2) (1999)). *Reliable Computing* 6 (2), 227. ISSN: 1573-1340. DOI: 10.1023/A:1009933709726 (cit. on p. 46).

Neumaier, A., O. Shcherbina, W. Huyer, T. Vinkó (2005): A comparison of complete global optimization solvers. *Mathematical Programming* 103 (2), 335–356. ISSN: 0025-5610. DOI: 10.1007/s10107-005-0585-4 (cit. on p. 4).

Ning, S., R. B. Kearfott (1997): A Comparison of Some Methods for Solving Linear Interval Equations. *Journal on Numerical Analysis* 34 (4), 1289–1305 (cit. on p. 46).

Nocedal, J., A. Wächter, R. A. Waltz (2009): Adaptive Barrier Update Strategies for Nonlinear Interior Methods. *Journal on Optimization* 19 (4), 1674–1693. ISSN: 1052-6234. DOI: 10.1137/060649513 (cit. on p. 26).

Nocedal, J., S. Wright (2005): *Numerical Optimization*. 2. ed. Operations Research and Financial Engineering. Berlin: Springer. ISBN: 0387303030 (cit. on pp. 21, 25, 26).

Orbach, O., C. M. Crowe (1971): Convergence promotion in the simulation of chemical processes with recycle-the dominant eigenvalue method. *The Canadian Journal of Chemical Engineering* 49 (4), 509–513. ISSN: 00084034. DOI: 10.1002/cjce.5450490414 (cit. on p. 17).

Pattison, R. C., M. Baldea (2014): Equation-oriented flowsheet simulation and optimization using pseudo-transient models. *AIChE Journal* 60 (12), 4104–4123. ISSN: 00011541. DOI: 10.1002/aic.14567 (cit. on p. 4).

Petković, M. S., L. D. Petković (1998): *Complex interval arithmetic and its applications*. 1. ed. Vol. 105. Mathematical Research. Berlin: Wiley-VCH. ISBN: 3527401342 (cit. on p. 42).

Powell, M. J. D. (1970): A Hybrid Method for Nonlinear Equations. In: *Numerical Methods for Nonlinear Algebraic Equations*. London: Gordon and Breach, 87–114 (cit. on pp. 22, 83).

Rajes, S. (2020): Modellierung und Modellvalidierung des Hydroalkylierungsprozesses. *Master Thesis*. Berlin: Technische Universität Berlin (cit. on pp. 116, 154, 198).

Rao, Y. V. C. (1997): *Chemical Engineering Thermodynamics*. London: Sangam Books. ISBN: 0863116884 (cit. on pp. 116, 178, 198).

Ratschek, H., J. Rokne (1988): *New computer methods for global optimization*. Mathematics and its Applications. Chichester: Horwood. ISBN: 0745801390. URL: https://pages.cpsc.ucalgary.ca/~rokne/global_book.pdf (last access 11/17/2021) (cit. on p. 55).

Ratschek, H., J. Rokne (2009): Interval Global Optimization. In: *Encyclopedia of Optimization*. Ed. by C. A. Floudas, P. M. Pardalos. Boston: Springer, 1739–1757. ISBN: 978-0-387-74758-3. DOI: 10.1007/978-0-387-74759-0 (cit. on p. 55).

Ratz, D., T. Csendes (1995): On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization* 7 (2), 183–207. ISSN: 1573-2916. DOI: 10.1007/BF01097060 (cit. on p. 56).

Raybaut, P. (2009): Spyder-documentation. URL: https://docs.spyder-ide.org (last access 01/06/2023) (cit. on p. 245).

Reum, A. (2022): Efficiency Improvement of an Interval-Arithmetic-based Solution Method applied on a stationary Flash Model for Separating a Multi-Component System. *Bachelor Thesis*. Berlin: Technische Universität Berlin (cit. on pp. 118, 138).

Rohn, J. (1993): Cheap and Tight Bounds: The Recent Result by E. Hansen Can Be Made More Efficient. *Interval Computations* 4, 13–21. URL: http://uivtx.cs.cas.cz/~rohn/publist/69.pdf (last access 01/05/2023) (cit. on p. 46).

Ruiz, D. (2001): *A Scaling Algorithm to Equilibrate Both Rows and Columns Norms in Matrices*. URL: https://cerfacs.fr/wp-content/uploads/2017/06/14_DanielRuiz.pdf (last access 01/05/2023) (cit. on p. 33).

Schewe, L., M. Schmidt (2019): *Optimierung von Versorgungsnetzen: Mathematische Modellierung und Lösungstechniken*. Springer eBook Collection. Berlin: Springer. ISBN: 9783662585399. DOI: 10.1007/978-3-662-58539-9 (cit. on p. 14).

Schnepper, C. A., M. A. Stadtherr (1996): Robust process simulation using interval methods. *Computers & Chemical Engineering* 20 (2), 187–199. ISSN: 00981354. DOI: 10.1016/0098-1354(95)00014-S (cit. on pp. 4, 44, 46, 49).

Seo, K., C. Tsay, B. Hong, T. F. Edgar, M. A. Stadtherr, M. Baldea (2020): Rate-Based Process Optimization and Sensitivity Analysis for Ionic-Liquid-Based Post-Combustion Carbon Capture. *Sustainable Chemistry & Engineering* 8 (27), 10242–10258. DOI: 10.1021/acssuschemeng.0c03061 (cit. on p. 4).

Shcherbina, O., A. Neumaier, D. Sam-Haroud, X.-H. Vu, T.-V. Nguyen (2003): Benchmarking Global Optimization and Constraint Satisfaction Codes. In: *Global Optimization and Constraint Satisfaction*. Ed. by G. Goos, J. Hartmanis, J. van Leeuwen, C. Bliek, C. Jermann, A. Neumaier. Vol. 2861. Lecture Notes in Computer Science. Berlin and Heidelberg: Springer, 211–222. ISBN: 978-3-540-20463-3. DOI: 10.1007/978-3-540-39901-8{\textunderscore}16. URL: https://arnold-neumaier.at/glopt/coconut/index.html (cit. on p. 155).

Sobol, I. (1967): On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics* 7 (4), 86–112. ISSN: 00415553. DOI: 10.1016/0041-5553(67)90144-9 (cit. on p. 150).

Song, W., G. Venimadhavan, J. M. Manning, M. F. Malone, M. F. Doherty (1998): Measurement of Residue Curve Maps and Heterogeneous Kinetics in Methyl

Acetate Synthesis. *Industrial & Engineering Chemistry Research* 37 (5), 1917–1928. ISSN: 0888-5885. DOI: 10.1021/ie9708790 (cit. on p. 191).

Steffensen, J. F. (1933): Remarks on iteration. *Scandinavian Actuarial Journal* 1933 (1), 64–72. ISSN: 0346-1238. DOI: 10.1080/03461238.1933.10419209 (cit. on p. 17).

Thomas, L. H. (1949): *Elliptic Problems in Linear Difference Equations over a Network*. Ed. by I. W. Laboratority. New York (cit. on pp. 5, 158).

Tolksdorf, G., E. Esche, G. Wozny, J.-U. Repke (2019): Customized Code Generation based on User Specifications for Simulation and Optimization. *Computers & Chemical Engineering* 121, 670–684. ISSN: 00981354. DOI: 10.1016/j.compchemeng.2018.12.006 (cit. on p. 103).

Tsuboka, T., T. Katayamak (1976): General design algorithm based on pseudo-equilibrium concept for multistage multi-component liquid-liquid separation processes. *Journal Of Chemical Engineering Of Japan* 9 (1), 40–45. ISSN: 0021-9592. DOI: 10.1252/jcej.9.40 (cit. on p. 5).

van der Waals, J. D. (1873): Over de Continuiteit van den Gas- en Vloeistoftoestand. *Ph.D. Thesis*. Leiden (cit. on p. 178).

van Iwaarden, R. J., W. Lodwick (1996): An Improved Unconstrained Global Optimization Algorithm. *phdthesis*. USA (cit. on p. 6).

Wächter, A., L. T. Biegler (2006): On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* 106 (1), 25–57. ISSN: 0025-5610. DOI: 10.1007/s10107-004-0559-y (cit. on p. 83).

Wang, J. C., G. E. Henke (1966): Tridiagonal Matrix for Distillation. *Hydrocarbon Processing* 45 (8), 155–163 (cit. on pp. 157, 159).

Wang, L., X. Sun, L. Xia, J. Wang, S. Xiang (2020): Inside–Out Method for Simulating a Reactive Distillation Process. *Processes* 8 (5), 604. DOI: 10.3390/pr8050604 (cit. on p. 5).

Watkins, D. S. (2002): *Fundamentals of Matrix Computations*. Hoboken, NJ, USA: John Wiley & Sons, Inc. ISBN: 9780471249719. DOI: 10.1002/0471249718 (cit. on p. 29).

Wegstein, J. H. (1958): Accelerating Convergence of Iterative Processes. *Communications of the ACM* 1 (6), 9–13. ISSN: 0001-0782. DOI: 10.1145/368861.368871 (cit. on p. 17).

Wilkinson, J. H. (1961): Error Analysis of Direct Methods of Matrix Inversion. *Journal of ACM* 8 (3), 281–330. DOI: 10.1145/321075.321076 (cit. on p. 46).

Index