NTNU
Norwegian University of Science and Technology

Design and Implementation of Modular Subroutines for Simulation of LNG Plants

Matias Vikse

Master of Energy and Environmental Engineering

Submission date:	June 2016
Supervisor:	Truls Gundersen, EPT
Co-supervisor:	Harry Alexander James Watson, Massachusetts Institute of Technology Donghoi Kim, EPT

Norwegian University of Science and Technology
Department of Energy and Process Engineering

NTNU
Norwegian University
of Science and Technology

Department of Energy
and Process Engineering

EPT-M-2016-150

MASTER THESIS

for

Student Matias Vikse

Spring 2016

Design and Implementation of Modular Subroutines for Simulation of LNG Plants

Design og implementering av modulære subrutiner for simulering av LNG-anlegg

Background and objective

As part of a collaboration project between MIT and NTNU partially funded by Statoil, a new tool is under development for simulation and optimization of different LNG process concepts. The tool should be based on recent developments in the area of Global Optimization in Paul Barton's group at MIT, and the scope is primarily to handle the liquefaction part of LNG plants.

Equation based approaches have the advantage of high efficiency but low robustness, while Sequential Modular approaches have higher robustness but low efficiency in computing time. Based on experience from this collaboration project and available literature, the equation based approaches have problems dealing with more complex thermodynamic models in conjunction with flash calculations (i.e. Vapor Liquid Equilibrium – VLE). When solving simulation problems in general and optimization problems in particular, there is a need for a very large number of VLE calculations. Based on this, it is of interest to study whether these parts of the simulation/optimization tool could use modular subroutines, while the flowsheet is solved using equation oriented techniques, or whether the entire simulation/optimization should be modularized.

Recent work at MIT using nonsmooth methods are compatible with modularization (partly or fully) of simulation/optimization problems as mentioned above. A chain rule and an implicit function theorem for Lexicographic-Directional (LD) derivatives that allow calculation of the necessary sensitivities of the subroutines. Work is also developing on a nonsmooth version of the well-known inside-out algorithm for handling non-ideal thermodynamics, which could be used in either of the two proposed simulation frameworks.

The ***main objective*** of this master thesis is to design and implement modular subroutines in the simulation code to increase the robustness and allow for more complex thermodynamic models and LNG processes.

The following tasks are to be considered:

1. Study LD-derivatives more deeply; in particular to understand the chain rule and implicit function theorem (including implementation), and why these results are necessary for modular flowsheet simulation and optimization. This is the main mathematical theory that will underlay the work.
2. Adapt the current simulation code to work in both modes mentioned above, i.e. modularized VLE calculations (only), and a fully Sequential Modular approach. This can be done at first with ideal thermodynamics for simplicity. Compare the performance of the two frameworks on various processes, including those that were studied in the Specialization Project during the fall semester 2015.
3. Implement a robust flash calculation module for nonideal thermodynamics.
4. Generally identify areas for improvement moving forward, and make recommendations for the future of the software.

-- " --

Within 14 days of receiving the written text on the master thesis, the candidate shall submit a research plan for his project to the department.

When the thesis is evaluated, emphasis is put on processing of the results, and that they are presented in tabular and/or graphic form in a clear manner, and that they are analyzed carefully.

The thesis should be formulated as a research report with summary both in English and Norwegian, conclusion, literature references, table of contents etc. During the preparation of the text, the candidate should make an effort to produce a well-structured and easily readable report. In order to ease the evaluation of the thesis, it is important that the cross-references are correct. In the making of the report, strong emphasis should be placed on both a thorough discussion of the results and an orderly presentation.

The candidate is requested to initiate and keep close contact with his/her academic supervisor(s) throughout the working period. The candidate must follow the rules and regulations of NTNU as well as passive directions given by the Department of Energy and Process Engineering.

Risk assessment of the candidate's work shall be carried out according to the department's procedures. The risk assessment must be documented and included as part of the final report. Events related to the candidate's work adversely affecting the health, safety or security, must be documented and included as part of the final report. If the documentation on risk assessment represents a large number of pages, the full version is to be submitted electronically to the supervisor and an excerpt is included in the report.

Pursuant to "Regulations concerning the supplementary provisions to the technology study program/Master of Science" at NTNU §20, the Department reserves the permission to utilize all the results and data for teaching and research purposes as well as in future publications.

The final report is to be submitted digitally in DAIM. An executive summary of the thesis including title, student's name, supervisor's name, year, department name, and NTNU's logo and name, shall be submitted to the department as a separate pdf file. Based on an agreement with the supervisor, the final report and other material and documents may be given to the supervisor in digital format.

☐ Work to be done in lab (Water power lab, Fluids engineering lab, Thermal engineering lab)

☐ Field work

Department of Energy and Process Engineering, 18 January 2016

[illegible]

Olav Bolland
Department Head

[illegible]

Truls Gundersen
Academic Supervisor

Co-Supervisors:	Harry Watson, PhD Student, Department of Chemical Engineering, Massachusetts Institute of Technology (MIT), USA E-mail: hwatson@mit.edu
	Donghoi Kim, PhD Student, Department of Energy and Process Engineering, NTNU, E-mail: Donghoi.Kim@ntnu.no

Abstract

A nonsmooth equation-oriented multistream heat exchanger (MHEX) model has been developed by the Process Systems Engineering Laboratory at Massachusetts Institute of Technology that is intended to be a part of a rigorous optimization and simulation tool for liquefied natural gas (LNG) processes. The model was successfully used to simulate the poly refrigerant integrated cycle operations (PRICO®) process for LNG production, though it suffered from convergence difficulties in more complex single mixed refrigerant processes. The primary challenge was flash calculations, which frequently failed to converge with a Newton solver even for initial guesses close to the solution. Equation-oriented simulation models have the advantage of high efficiency, but are generally less robust than the sequential-modular approach, such that improved performance may be achieved by using a different simulation framework.

This master thesis studies two alternative model structures. First, the equation-oriented framework is replaced with a hybrid solution, in which vapour-liquid equilibrium calculations are included as nested subroutines and solved sequentially. Next, a fully sequential-modular approach is considered. The models are tested for different single mixed refrigerant processes, and are solved with a nonsmooth Newton-type solver using Clarke Jacobian elements as generalized derivatives. The implicit function theorem for lexicographically smooth functions is used for computing analytical derivatives in the subroutines.

Results showed that the hybrid models were considerably more robust than the original equation-oriented models. In addition, they required fewer iterations to converge. However, as expected, they suffered a loss in efficiency. About half the computing time in the hybrid PRICO model was spent on the vapour-liquid equilibrium modules, which was primarily due to derivatives calculations. As a consequence the time per iteration was between 4 and 5 times longer for the processes studied, and even with fewer required iterations, the models were normally 1.5-3 times slower. On the other hand, the sequential-modular framework turned out to be unsuitable for simulating the LNG models as it was both significantly less robust and efficient than the other approaches. The observed drop in robustness is against the general theory, however, and it was concluded that the convergence problems were caused by modularizing the MHEX model.

Sammendrag

En ikke-glatt likningsbasert multistrøms varmeveksler (MHEX) modell, har blitt utviklet av Process Systems Engineering Laboratoriet ved Massachusetts Institute of Technology. Dette har vært en del av et større prosjekt for å lage et robust simulerings- og optimeringsverktøy for flytende naturgass (LNG) produksjon. Modellen har blitt brukt til å simulere PRICO- prosessen, men den viste seg å ha vansker med å konvergere for større og mer kompliserte modeller. Hovedutfordringen var flash-beregningene, som ofte mislyktes å konvergere i Newtonløseren selv for initialverdier nærme løsningen. Likningsbaserte modeller har en fordel av økt effektivitet, men er normalt mindre robuste enn modulbaserte simuleringsverktøy. Av denne grunn kan det være fordelaktig å studere alternative modellstrukturer.

Masteroppgaven ser nærmere på to ulike modellformer. Det ene er et hybrid-alternativ hvor flashberegningene er modulisert og ligger innbakt i den lignings-baserte modellen. Det andre er en fullstendig modulbasert modellform. De to alternativene er testet for forskjellige single mixed refrigerant (SMR) modeller, og er løst ved bruk av en Newton metode for ikke-glatte funksjoner der Clarke Jacobian elementer brukes som gradienter. For å beregne gradienter i modulene, benyttes et implisitt funksjonsteorem for lexicografisk-glatte funksjoner.

Resultatene viser at den hybride metoden var betydelig mer robust enn den ligningsbaserte modellen. I tillegg krevde modellformen mindre iterasjoner for å konvergere. Derimot opplevde de hybride modellene redusert effektivitet, som er i samsvar med teorien. Det viste seg at omtrent halvparten av den totale stimuleringstiden for PRICO modellen ble brukt til å løse flashberegningene, noe som primært skyldes tidkrevende beregninger av gradienter. Dette førte imidlertid til at iterasjonstiden for de hybride modellene var mellom 4 og 5 ganger lengre, mens den totale stimuleringstiden normalt var 1.5-3 ganger lengre. På en annen side, viste den fullstendig modulbaserte modellformen seg å være ugunstig for simulering av LNG- modeller, ettersom den var både langt saktere og betydelig mindre robust. Det siste skrider mot generell teori på området, og det ble konkludert med at det hele skyldes en modulisering av varmevekslermodellen.

Acknowledgements

This master thesis is written for the Department of Energy and Process Engineering at the Norwegian University of Science and Technology (NTNU) as a part of a Statoil funded research collaboration with Massachusetts Institute of Technology. The thesis was written during the spring semester 2016 in the Process Systems Engineering Laboratory at MIT. Throughout my stay in the United States, I was financially supported by the Fulbright Foundation.

I would like to thank my supervisor Professor Truls Gundersen for all the help he has given me on this master thesis, and for letting me take part in the research collaboration between MIT, NTNU and Statoil. I also wish to thank Professor Paul I. Barton for organizing a place for me in the lab, and for giving me academic advice during my stay here. In addition, I want to thank my co-supervisors, PhD students Harry A. J. Watson and Donghoi Kim, for their help and support during the semester. Lastly, I want to thank Dr. Peter Stechlinski for helpful academic discussions in the lab.

Boston 2016

Matias Vikse

Contents

List of Figures

List of Tables

Acronyms

AD	Automatic differentiation
DMR	Dual Mixed Refrigerant
EO	Equation-oriented
HP	High-pressure
L	Lexicographic
LD	Lexicografic directional
LNG	Liquefied natural gas
LP	Low-pressure
MHEX	Multistream heat exchanger
NG	Natural gas
NGL	Natural gas liquids
NLP	Nonlinear programming
SM	Sequential-modular
SMR	Single mixed refrigerant

Nomenclature

Symbol	Unit	Description
α	–	Vapour fraction
ΔT	[K]	Temperature difference
ϵ	–	Stopping criterion/convergence tolerance
EBP	[kW]	Enthalpies of the extended composite curves
F	[kmol/s]	Feed flowrate
h	[kJ/kmol]	Molar enthalpy
H	[kW]	Enthalpy
K	–	Equilibrium constant
L	[kmol/s]	Liquid flowrate
mC_p	[kW/K]	Heat capacity flowrate
n_C	–	Number of components
n_2p	–	Number of discrete flash stages
p	[MPa]	Pressure
$\mathbf{p}$	–	Parameters
Q	[kW]	Heat transfer
RSD	[%]	Relative standard deviation
$\mathbf{t}\left(\mathbf{x}, \mathbf{w}\right)$	–	Tear equations

Symbol	Unit	Description
T	[K]	Temperature
UA	[MW/K]	Overall heat transfer coefficient
V	[kmol/s]	Vapour flowrate
$\mathbf{w}$	–	Tear variables
$\mathbf{z}$	–	Feed composition

Subscripts and superscripts

Symbol	Description
2p	Two-phase region
C	Cold stream
c	Critical temperature
f	Feed
H	Hot stream
l	Liquid
LM	Log mean
sub	Subcooled liquid
sup	Superheated vapour
TP	Triple point
v	Vapour

List of Symbols

This section lists mathematical symbols and operators in the report.

Symbol	Description
a	Lower case letters denote variables
A	Upper case letters denote sets
$\mathbf{A}$	The directions matrix for the inner variables
$\lvert x \rvert$	Absolute value of x
$\leftarrow$	Assignment operator
$\mathbf{b}$	Bold lower case letters denote vectors
$\mathbf{B}$	Bold upper case letters denote matrices
$\mathbf{f} \circ \mathbf{g}\left(\mathbf{x}\right)$	Composite function $\mathbf{f}\left(\mathbf{g}\left(\mathbf{x}\right)\right)$
$\mathrm{conv}(A)$	Convex hull of A
$\mathrm{plen}(A)$	Plenary hull of A
$\equiv$	"Defined as"
$\det\mathbf{A}$	Determinant of a matrix $\mathbf{A}$
$\mathbf{f}^{n}_{\mathbf{x},\mathbf{M}}\left(\mathbf{m}_n\right)$	n-th order directional derivative at $\mathbf{x}$ in direction $\mathbf{m}_n$
$\in$	"Element of"
$\lVert\cdot\rVert$	Euclidean norm
$\exists$	"There exists"

Symbol	**Description**
$\forall$	"For all"
$\mathbf{G}$	Generalized derivative element
$\mathbf{g}(\mathbf{x})$	Implicit function
$\mathbf{I}$	Identity matrix
$\mathbf{A}^{-1}$	Inverse of $\mathbf{A}$
$f : \mathbb{R}^n \to \mathbb{R}$	Mapping from $\mathbb{R}^n$ to $\mathbb{R}$
$\mathbf{J}f$	Jacobian of f
$\mathbf{M}$	Directions matrix
$\mathbf{m}_k$	Column k of the directions matrix
$\mathbf{d}$	Direction vector
$\mathbb{N}$	Set of all natural numbers
$\mathbf{J}_\mathrm{L}\mathbf{f}(\mathbf{x};\mathbf{M})$	Lexicographic derivative of $\mathbf{f}$ at $\mathbf{x}$ in directions $\mathbf{M}$
$\mathbf{f}'(\mathbf{x};\mathbf{M})$	Lexicographic directional derivative of $\mathbf{f}$ at $\mathbf{x}$ in directions $\mathbf{M}$
$\frac{\partial}{\partial x}$	Partial derivative with respect to x
∇f	Gradient of function f
$\pi_n\mathbf{f}(\mathbf{x})$	Projection of $\mathbf{f}$ with respect to the n-th argument
$\mathbb{R}$	Set of all real numbers
$:$	"Such that"
$\subset$	Subset
$\partial_\mathrm{B}\mathbf{f}(\mathbf{x})$	B-subdifferential of $\mathbf{f}$ at $\mathbf{x}$
$\partial\mathbf{f}(\mathbf{x})$	Clarke Jacobian of $\mathbf{f}$ at $\mathbf{x}$
$\partial_\mathrm{L}\mathbf{f}(\mathbf{x})$	Lexicographic subdifferential of $\mathbf{f}$ at $\mathbf{x}$
$\mathbf{A}^\mathrm{T}$	Transpose of matrix (vector) $\mathbf{A}$

Chapter 1

Introduction

Liquefied natural gas (LNG) plays an important role in the shift towards green energy sources. It is considered a cleaner alternative to oil and coal – with a low sulfur content and no particle emissions – and at the same time avoids the transportation difficulties associated with traditional pipeline gas. However, natural gas liquefaction is a very energy intensive process that requires cooling to circa -162°C. Investments in expensive, custom and proprietary technology like cryogenic heat exchangers and rotating equipment are necessary, and along with the high operating costs, liquefaction accounts for about 30-45% of the cost in the LNG chain [27]. Moreover, natural gas is often used to power the LNG processes, which results in significant emissions. There exist, therefore, both economic and environmental incentives for improving process designs.

1.1 Motivation

The high operating and capital costs of natural gas liquefaction, means a thorough and reliable analysis of the system should be conducted before making a commitment. Unfortunately, state-of-the-art simulation tools like Aspen Plus® that are available for LNG process simulation offer only local optimization software [2], which has clear limitations in systems design. Therefore, an equation-oriented model for multistream heat exchangers (MHEX) is currently being developed by the Process Systems Engineering Laboratory at Massachusetts Institute of Technology under supervision of Professor Barton, for use in a global optimization algorithm. The MHEX model has already been used to successfully simulate the Poly Refrigerant Integrated Cycle Operations (PRICO®) process, and was tested in the specialization project for more complex liquefaction systems [40].

Results showed that the MHEX model struggled for the larger processes. Additional variables made initialization challenging, especially for the discrete flash calculations used to approximate the heating/cooling curves in the two-phase region. The challenge with flash calculations became pronounced when using a cubic equation of state, which caused a sharp increase in the number of two-phase variables from 2 to 16 per stream segment. Besides from the convergence issues, the model exceeded the memory limit of (dense) CPLEX®. Although, the limit can be raised by exploiting sparsity (see [40]), another possibility would be to use a sequential-modular approach. The sequential-modular framework partitions the models into smaller subroutines that are solved separately, thus reducing the size of the system and generally making it more robust. This comes at the expense of lower efficiency, however, mainly due to the additional cost of solving and computing derivatives for the modules. In order to make the models more robust, it is interesting to study alternative frameworks. For instance, either a fully sequential-modular framework where each module is solved separately, or by using a partially modular approach where flash calculations are nested and solved sequentially.

1.2 Master thesis structure

This master thesis is divided into seven chapters. Chapter 2 deals with modeling of liquefaction processes for natural gas, and more specifically the multistream heat exchanger model by Watson et al. Next, Chapter 3 presents different frameworks for process simulation, as well as a method for calculating analytical derivatives of modular subroutines by invoking the implicit function theorem. The same chapter also briefly discusses nonsmoothness in the MHEX model. Chapter 4 describes key concepts in nonsmooth analysis before presenting an implicit function theorem for lexicographically smooth functions. Analogously to the procedure presented in Chapter 3, this theorem is then used in Chapter 5 for implementing a sensitivities calculation procedure for nonsmooth modular subroutines. The next two chapters look at two alternative frameworks for the MHEX model. First, in Chapter 6 a hybrid model is created, in which the vapour-liquid equilibrium (VLE) calculations are modularized and solved sequentially. The hybrid framework is then tested and compared with the original model by studying the LNG processes from the specialication project. Chapter 7 includes the MHEX model in a fully sequential-modular framework, which likewise to Chapter 6, is compared with the partial-modular and the equation-oriented models. Finally, an LNG process that did not converge in the specialization project is revisited in Chapter 8.

1.3 Deviations from the original project description

The scope of this project is to design and implement modular subroutines such that the MHEX model can be tested for alternative frameworks. In agreement with supervisor, the thesis focused on the first two suggested tasks, especially testing the hybrid and sequential-modular models. All the simulations were with ideal properties, as robust vapour-liquid equilibrium calculations had yet to be implemented for nonideal thermodynamics. Also, work has been initiated on the third task, with the rest left for future work along with testing the frameworks with a cubic equation of state.

Chapter 2

LNG process modeling

2.1 LNG liquefaction plants

Natural gas liquefaction plants are cryogenic processes that cool the gas from ambient temperature down to circa 111K. At this temperature, the natural gas is liquid at atmospheric conditions, which is suitable for energy storage or transport over long distances. There currently exist three main types of LNG plants [27]:

- Base-load plants
- Peak-shaving plants
- Small-scale plants

Base-load plants are based on a field developement and are the main facilities for processing and shipping of LNG. They typically have a production capacity of more than 3 mtpa (million tons per annum) [27]. Peak-shaving plants are smaller liquefaction processes that are connected to an existing gas network. Natural gas is liquefied and stored when demand is low, so that it can be sold during days of high demand. Finally, small-scale plants are LNG plants with a production capacity less than 500 000 tpa (tons per annum) [27]. They are connected to an existing gas network and distribute LNG to small customers. The type of LNG plant and trade-offs between capital and operating costs determines the design of liquefaction processes. Simpler processes like the PRICO (Figure 2.1) are normally only considered for small-scale and peak-shaving plants. Base-load production, on the other hand, often employ more complex processes (e.g. the dual mixed refrigerant (DMR) process in Figure 2.2) in order to reduce the operating costs.

For base-load plants that handle unprocessed feed gas, a key decision is whether

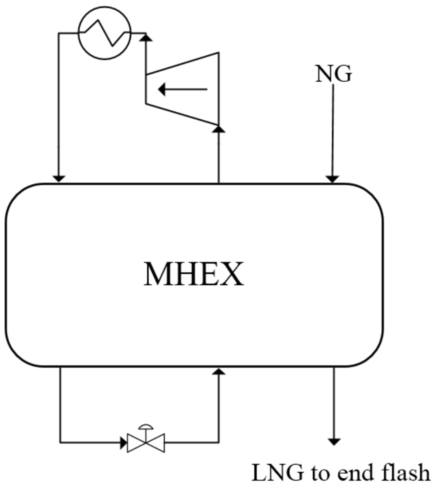


Figure 2.1: The PRICO process.

to have integrated or upstream natural gas liquids (NGL) extraction. Heavier hydrocarbons freeze out at cold temperatures, which can cause plugging of process equipment [29]. The LNG is also subject to quality constraints that may require heating value adjustments by removing heavier components [29]. In addition, liquefied petroleum gas (LPG), i.e. propane and butane [14], is a valuable commodity and is therefore normally sold separately. With integrated NGL extraction, the cold in the liquefaction process is used for liquefying and separating the heavy components. The disadvantage is that the natural gas has to be below the critical pressure p_c in order for separation to occur. At lower pressures, the work required for cooling the gas will increase, resulting in additional costs [28]. There also exists a trade-off between hydrocarbon recovery and energy consumption, which is determined by the temperature of the scrub-column. For upstream extraction, on the other hand, the NGLs are removed in a separate process prior to liquefaction. Normally, a turbo-expander process is employed that expands, cools, liquefies and separates NGLs before recompressing the gas [29]. High recovery is possible, and unlike LNG processes with integrated NGL extraction, the liquefaction pressure may be above p_c. However, additional power is needed for recompression.

Liquefaction processes typically exhibit tight designs in the low-temperature region where thermodynamic losses from irreversible heat transfer are significant (see Figure 2.3) [40]. Temperature differences in the heat exchangers are minimized by using cascades and/or refrigerant mixture compositions, as well as splitting of streams to provide cooling at different temperature levels. As a consequence, multistream heat exchangers are used, serving as the core of the liquefaction

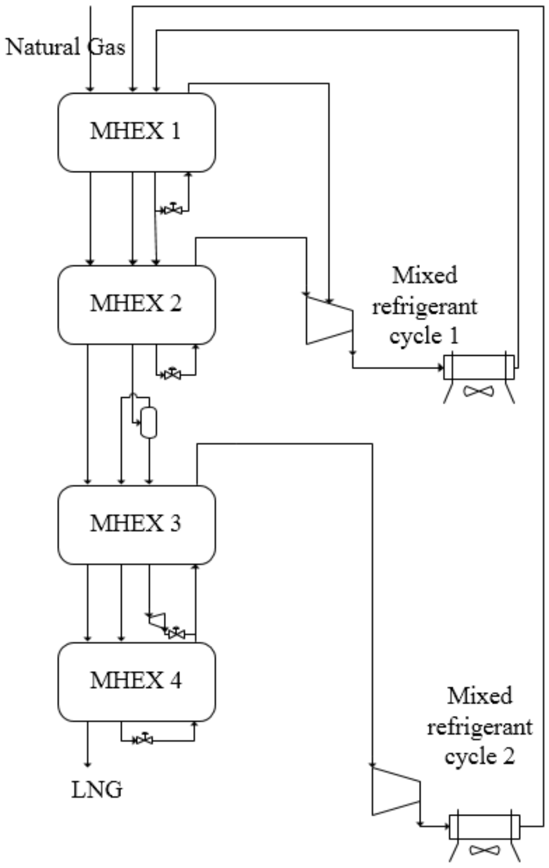


Figure 2.2: The DMR process.

process. The most commonly used MHEXs in LNG production are plate-fin and spiral-wound heat exchangers [29]. Because of the tightness in design, especially at cold temperatures, accurate modeling of these heat exchangers is necessary to correctly describe the liquefaction process.

2.2 Multistream heat exchanger model

The multistream heat exchanger model developed by Watson et al. [42] is a generalization of the two-stream countercurrent heat exchanger in Figure 2.4. The heat exchanger is completely characterized by Equations (2.1)-(2.3), which are the energy balance, as well as the equations for the minimum temperature difference

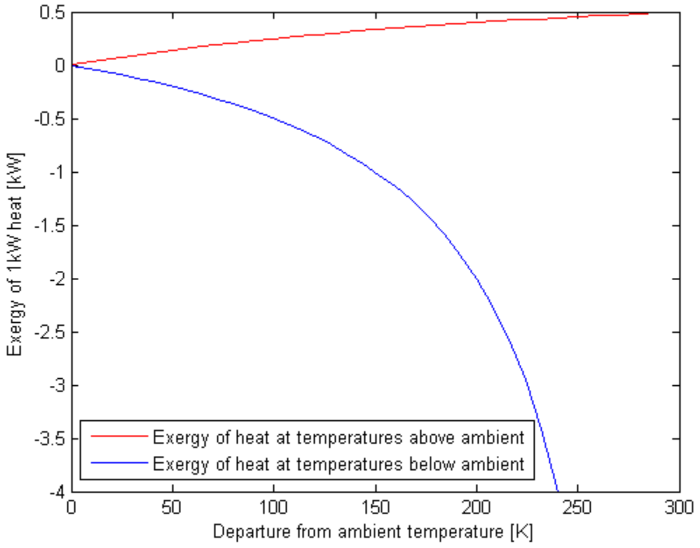


Figure 2.3: Graph showing the exergy-change for a system at different temperatures as it is heated 1kW [40].

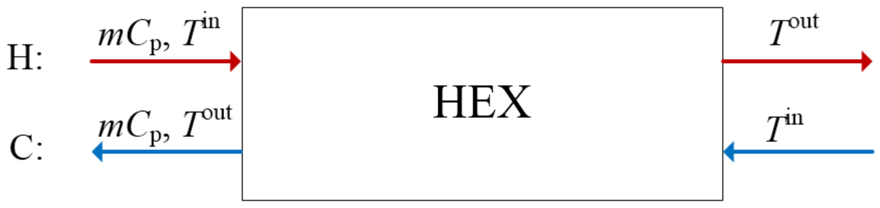


Figure 2.4: The two-stream countercurrent heat exchanger.

ΔT_{min} and the overall heat transfer coefficient.

$$mC_{\text{p,H}}\left(T_{\text{H}}^{\text{in}} - T_{\text{H}}^{\text{out}}\right) = mC_{\text{p,C}}\left(T_{\text{C}}^{\text{out}} - T_{\text{C}}^{\text{in}}\right), \tag{2.1}$$

$$\Delta T_{\text{min}} = \min\left\{T_{\text{H}}^{\text{in}} - T_{\text{C}}^{\text{out}}, T_{\text{H}}^{\text{out}} - T_{\text{C}}^{\text{in}}\right\}, \tag{2.2}$$

$$UA = \frac{Q}{\Delta T_{\text{LM}}}. \tag{2.3}$$

The mC_{p} in Equation (2.1) is the heat capacity flowrate for the hot and cold streams, and is assumed constant in the heat exchanger. For Equation (2.3), U is the overall heat transfer coefficient, A is the heat transfer area and ΔT_{LM} is the log-mean temperature difference.

Watson et al. [42] extended these equations to the case for multiple hot and cold stream inlets (see Figure 2.5).

$$\sum_{i=1}^{n_{\mathrm{H}}} mCp_{\mathrm{H},i} \left(T_{\mathrm{H},i}^{\mathrm{in}} - T_{\mathrm{H},i}^{\mathrm{out}}\right) = \sum_{j=1}^{n_{\mathrm{C}}} mCp_{\mathrm{C},j} \left(T_{\mathrm{C},j}^{\mathrm{in}} - T_{\mathrm{C},j}^{\mathrm{out}}\right), \tag{2.4}$$

$$UA = \sum_{k=1}^{K-1} \frac{\Delta Q^k}{\Delta T_{\mathrm{LM}}^k}. \tag{2.5}$$

The n_{H} and n_{C} in Equation (2.4) are the total number of hot and cold streams in the MHEX. The energy balance for the MHEX is otherwise analogous to the two-stream case. An equation for heat transfer area (2.5) on the other hand, is found by only allowing for vertical heat exchange between the composite curves. Theoretically, this is done by partitioning the composite curves into enthalpy intervals k with endpoints defined by kinks in either the hot or cold composite curve. Here, K is used to denote the total number of interval temperatures.

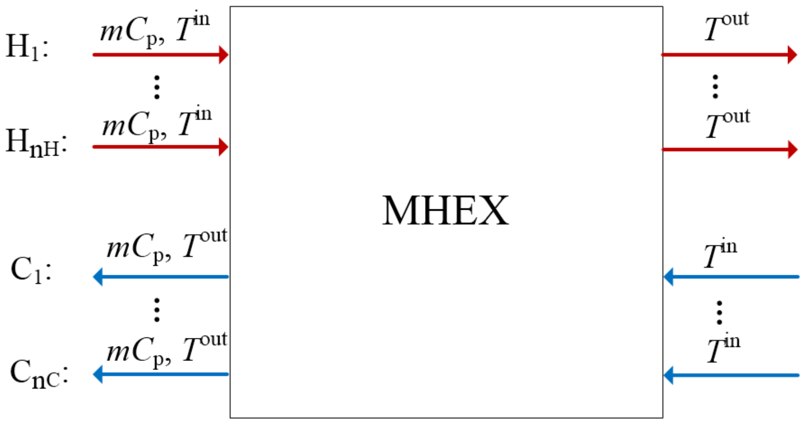


Figure 2.5: A multistream heat exchanger with n_{H} hot streams and n_{C} cold streams.

For two-stream heat exchangers, the pinch point is always located at one of the stream inlets/outlets. This is not the case for MHEXs, however, where pinching may occur at intermediate temperatures. Consequently, (2.2) cannot readily be extended to the general case. Instead, to ensure the only solution will be the one corresponding to a minimum temperature difference equal to ΔT_{min} between the composite curves, Watson et al. [42] included the following additional constraint:

$$\min_{p\in P}\{EBP_{\mathrm{C}}^{p} - EBP_{\mathrm{H}}^{p}\} = 0, \tag{2.6}$$

where EBP_{H}^{p} and EBP_{C}^{p} are the enthalpies of the extended composite curves at given pinch point candidate p, evaluated from the following equations:

$$\begin{aligned} EBP_{\mathrm{H}}^{p} = \sum_{i=1}^{n_{\mathrm{H}}} mCp_{\mathrm{H},i} \big[&\max\left\{0, T^{p} - T_{\mathrm{H},i}^{\mathrm{out}}\right\} - \max\left\{0, T^{p} - T_{\mathrm{H},i}^{\mathrm{in}}\right\} \\ &-\max\left\{0, T_{\mathrm{H}}^{\min} - T^{p}\right\} + \max\left\{0, T^{p} - T_{\mathrm{H}}^{\max}\right\}\big], \quad \forall p \in P, \end{aligned} \tag{2.7}$$

$$\begin{aligned} EBP_{\mathrm{C}}^{p} = \sum_{j=1}^{n_{\mathrm{C}}} mCp_{\mathrm{C},j} \big[&\max\left\{0, (T^{p} - \Delta T_{\min}) - T_{\mathrm{C},j}^{\mathrm{in}}\right\} \\ -\max\left\{0, (T^{p} - \Delta T_{\min}) - T_{\mathrm{C},j}^{\mathrm{out}}\right\} &+ \max\left\{0, (T^{p} - \Delta T_{\min}) - T_{\mathrm{C}}^{\max}\right\} \\ &-\max\left\{0, T_{\mathrm{C}}^{\min} - (T^{p} - \Delta T_{\min})\right\}\big], \quad \forall p \in P. \end{aligned} \tag{2.8}$$

The P in (2.7) and (2.8) defines the set of pinch candidates, which according to the pinch method consists of all the stream supply temperatures. The only difference between regular and extended composite curves is that the latter extrapolates the endpoints to the maximum and minimum temperatures in the process. This is to ensure that there always exist a corresponding hot or cold temperature even for pinch candidates at the endpoints of the composite curves. The slope of each extension is made as low as possible so that it will not change the pinch location. In the MHEX model by Watson et al., all pinch point candidates T^{p} were expressed by their hot stream temperature [42]

$$T^{p} = \begin{cases} T_{\mathrm{H},i}^{\mathrm{in}}, & \forall p = i \in n_{\mathrm{H}}, \\ T_{\mathrm{C},j}^{\mathrm{in}} + \Delta T_{\min}, & \forall p = j \in n_{\mathrm{C}}. \end{cases}$$

According to Equation (2.6), the pinch point will be at the point where the horizontal distance between the two extended composite curves is equal to zero. $\Delta T_{\min}$ violations happen whenever the expression $EBP_{\mathrm{C}}^{p} - EBP_{\mathrm{H}}^{p}$ is negative.

The assumption of constant mC_{p}s is only valid for small temperature changes. Therefore, to account for the wide temperature range in LNG liquefaction, the composite curves are approximated using affine segments with constant mC_{p}s over an enthalpy interval (for a definition of affine functions, see Section 4.1). The accuracy of the representation is determined by the number of stream segments, which was set to three for the subcooled and superheated region and to five for the two-phase region in most of the simulations.

In summary, the MHEX model is composed of the following three equations:

$$\sum_{i=1}^{n_{\mathrm{H}}} mCp_{\mathrm{H},i}\left(T_{\mathrm{H},i}^{\mathrm{in}} - T_{\mathrm{H},i}^{\mathrm{out}}\right) = \sum_{j=1}^{n_{\mathrm{C}}} mCp_{\mathrm{C},j}\left(T_{\mathrm{C},j}^{\mathrm{in}} - T_{\mathrm{C},j}^{\mathrm{out}}\right),$$

$$UA = \sum_{k=1}^{K-1} \frac{\Delta Q^k}{\Delta T_{\mathrm{LM}}^k},$$

$$\min_{p \in P}\{EBP_{\mathrm{C}}^p - EBP_{\mathrm{H}}^p\} = 0.$$

With three model equations, the MHEX model can be solved for three unknown variables.

2.3 Vapour-liquid equilibrium calculations in the multistream heat exchanger model

The refrigerant(s) and natural gas traverse one or more phase regions in the LNG process that need to be captured by the MHEX model. In the single phase regions the fluid composition will remain constant and heat capacities will depend only on temperature (provided ideal gas properties are used). On the other hand, compositions vary in the two-phase region as components evaporate/condense while heating/cooling. During phase transitions, therefore, heat capacitites will be dependent on factors such as the vapour fraction (α) and the vapour/liquid compositions ($\mathbf{y}$ and $\mathbf{x}$).

The MHEX model approximates the two-phase region as a series of discrete pQ-flash calculations (see Figure 2.6) [41]. The pQ-flash is a vapour-liquid equilibrium (VLE) problem for a fixed heat load Q and pressure p. The feed composition $\mathbf{z}$, flowrate F, temperature T_{in}, and molar feed enthalpy h_{f} are known quantities in the problem, thus solving for the temperature T_{out}, vapour or liquid flowrates V/L, and the vapour and liquid compositions $\mathbf{y}$ and $\mathbf{x}$. For an N component system,

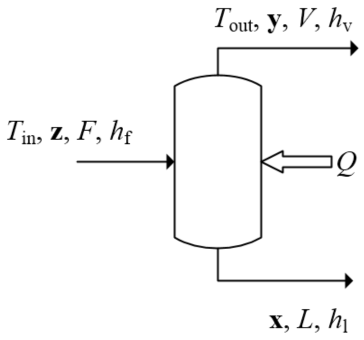

T_{out}, $\mathbf{y}$, V, h_{v}
T_{in}, $\mathbf{z}$, F, h_{f}
Q
$\mathbf{x}$, L, h_{l}

Figure 2.6: The pQ flash problem.

each stage is completely characterized by the following $2N + 3$ equations [11]:

$$L + V = F, \tag{2.9}$$

Total material balance

$$x_i L + y_i V = z_i F, \qquad \forall i = 1, 2, \ldots, N, \tag{2.10}$$

Component material balance

$$y_i = K_i x_i, \qquad \forall i = 1, 2, \ldots, N, \tag{2.11}$$

Phase equilibrium

$$h_\mathrm{l} L + h_\mathrm{v} V = h_\mathrm{f} F + Q, \tag{2.12}$$

Energy balance

$$\sum_i y_i - \sum_i x_i = 0, \tag{2.13}$$

Consitutive equation

where the heating/cooling terms Q are equally distributed between the different stages and sum up to the total enthalpy difference of the two-phase region.

Rachford and Rice [33] provided an alternative formulation of the flash equations

$$\sum_i^{n_\mathrm{C}} \frac{z_i (K_i - 1)}{1 + \alpha (K_i - 1)} = 0, \tag{2.14}$$

which is monotonically decreasing with respect to the vapour fraction α

$$\alpha \equiv \frac{V}{F}. \tag{2.15}$$

Equation (2.14) is derived from component material balances, phase equilibrium and the consitutive equation, and can be solved iteratively using the Newton-Raphson method. Since the function is monotonically decreasing, it contains no false α-roots, i.e. nonphysical roots, and thus no false solutions for the Newton-method. The full derivation of (2.14) is given in Appendix D.

The phase regions occurring in the MHEX as well as their location, are often not known a priori. Instead, phase transitions in the problem are usually approached by solving an MINLP with phase detection handled through binary variables [16, 17, 18]. Alternative methods involving complementary constraints have also been suggested [20]. All these solution strategies include solving nonconvex optimization problems that are computationally expensive.

Watson and Barton [41] propose an alternative model formulation that detects phase changes using a nonsmooth mid-function:

$$\operatorname{mid}\left\{\alpha, \alpha-1, -\sum_{i=1}^{n_{\mathrm{C}}} \frac{z_i\left(K_i-1\right)}{1+\alpha\left(K_i-1\right)}\right\}=0. \tag{2.16}$$

The mid-function is a composite function of the piecewise continuously differentiable ($\mathcal{PC}^1$) max and min functions (for definition of $\mathcal{PC}^1$, see Section 4.1):

$$\operatorname{mid}\{x, y, z\} \equiv \max\{\min(x, y), \min(\max(x, y), z)\}, \tag{2.17}$$

and is thus $\mathcal{PC}^1$. It can be solved using a nonsmooth Newton-type solver, e.g. the nonsmooth Newton-type method by Qi and Sun [32]:

$$\mathbf{G}\left(\mathbf{x}^k\right)\left(\mathbf{x}^{(k+1)}-\mathbf{x}^k\right)=-\mathbf{f}\left(\mathbf{x}^k\right), \tag{2.18}$$

avoiding the binary phase detection variables altogether. In Equation (2.18), $\mathbf{G}\left(\mathbf{x}^k\right)$ is a generalized derivative element, which is discussed in more detail in Chapters 3 and 4.

The nonsmooth formulation works as follows. For an all vapour outlet, $\alpha = 1$ and the Rachford-Rice term will be positive (see [35]). Hence, the third argument, which is the negative of the Rachford-Rice term, will be less than zero, reducing (2.16) to the middle term. For this case, the nonsmooth equation solver will set the vapour flowrate V equal to the feed flowrate F, corresponding to the superheated region. The argument for an all liquid outlet is analogous. Likewise, in the two phase region, the Rachford-Rice term will be zero from (2.14) and $0 \leq \alpha \leq 1$. In this case, the mid-function evaluates the third term to zero [41].

Equations (2.9)-(2.13) are general, in the sense that they do not depend on the thermodynamic models used. Equation (2.16) should thus be compatible with any fluid package. Nevertheless, solving the mid-function directly turns out to be quite challenging with the Peng-Robinson equation of state, as the nonsmooth Newton solver frequently fails to converge even for initial guesses close to the solution (Watson, Massachusetts Institute of Technology, Cambridge MA, personal communication, 2015). As a consequence, a more robust flash formulation is needed before implementing a cubic equation of state in the MHEX model. For this thesis, however, all simulations are done with ideal properties and hence use the formulation in (2.16).

2.4 Inside-out algorithm for pQ-flash calculations

Boston and Britt [11] came up with an algorithm for solving the vapour-liquid equilibrium (VLE) problem in Equations (2.9)-(2.13) using nested loops. The algorithm has proven to be very dependable, and is used in state-of-the-art process simulation tools like Aspen Plus® [26]. Rather than dealing with the problem directly by either substitution or a Newton method, it estimates a solution first by using simple models for calculating thermodynamic properties. Then, model parameters are updated in an outer loop using more rigorous relations. Overall convergence of the algorithm is achieved when the relative change in parameter values are within a given tolerance ϵ. With a calculation procedure going from an inner to an outer loop, Boston and Britt's algorithm is commonly known as the "inside-out" algorithm.

Instead of solving the $2N + 3$ VLE equations simultaneously, the inner loop is reduced to a single variable problem by introducing variables R and r_i [11]:

$$R \equiv \frac{K_r V}{K_r V + K_r^0 L}, \tag{2.19}$$

where K_r is a reference equilibrium constant taken as the weighted average of the individual K_i values

$$\ln\left(K_r\right) \equiv \sum_i w_i \ln\left(K_i\right), \tag{2.20}$$

and K_r^0 is merely introduced to avoid numerical difficulties whenever K_r gets very large or very small. The weights w_i in (2.20) are derived by Boston and Britt [11]:

$$w_i = \frac{t_i}{\sum_j t_j}, \tag{2.21}$$

where

$$t_i \equiv \frac{y_i}{1 + \frac{V}{F}\left(K_i - 1\right)}. \tag{2.22}$$

The other variables r_i introduced by Boston and Britt are defined as

$$r_i \equiv \frac{x_i L}{1 - R}. \tag{2.23}$$

Both the liquid flowrate L and molar composition x_i are unknown quantities, and are thus substituted in order to retain the single variable problem. By introducing a "volatility parameter"

$$\phi_i \equiv \ln\left(K_i / K_r\right), \tag{2.24}$$

the molar gas fractions y_i can be expressed in terms of x_i and K_r:

$$y_i = K_r e^{\phi_i} x_i. \tag{2.25}$$

Combining Equation (2.25) with the component material balances yields a more convenient formulation of the variables r_i:

$$r_i = \frac{z_i F}{1 - R + K_r^0 R e^{\phi_i}}. \tag{2.26}$$

See Appendix B for the full derivation of (2.26). Equation (2.26) expresses the inner variables r_i as a function of known quantities (z_i, F, ϕ_i, and K_r^0) and the variable R only.

From (2.26), another expression for K_r can be found, along with relations for the composition and flowrates [11]:

$$K_r = \frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i}, \tag{2.27}$$

$$x_i = \frac{r_i}{\sum_j r_j}, \tag{2.28}$$

$$y_i = \frac{\mathrm{e}^{\phi_i} r_i}{\sum_j \mathrm{e}^{\phi_j} r_j}, \tag{2.29}$$

$$L = (1 - R) \sum_i r_i. \tag{2.30}$$

All four equations are derived in Appendix B.

In the inner convergence problem, the total energy balance is used as a residual

$$\Psi = h_{\mathrm{f}} F + Q - L\left(h_{\mathrm{l}} - h_{\mathrm{v}}\right) - F h_{\mathrm{v}} = 0, \tag{2.31}$$

where R is implicitly defined through the variables h_{v}, h_{l}, and V.

Departure functions that are assumed affinely dependent on temperature are used for finding the enthalpies in (2.31), where system temperature is calculated from the reference equilibrium constant.

$$\ln\left(K_r\right) = A + B\left(\frac{1}{T} - \frac{1}{T^*}\right), \tag{2.32}$$

$$\Delta h_{\mathrm{v}} = C + D\left(T - T^*\right), \tag{2.33}$$

$$\Delta h_{\mathrm{l}} = E + F\left(T - T^*\right). \tag{2.34}$$

The variable T^* is a reference temperature set by the user. Defining enthalpies in terms of departure functions requires calculated ideal gas enthalpies. For ideal gas, however, enthalpy is a function of temperature alone, and does not require large additional computational cost to evaluate.

Equations (2.32)-(2.34) are designed to be as simple as possible to limit the computational complexity. Rather, the model parameters A-F and ϕ_i are updated in an outer loop by more rigorous models to ensure the system approaches a solution. Calculating thermodynamic properties is quite time consuming, making up most of the CPU time in computer simulations [24, 26, 44]. Avoiding such computations at every iteration by delegating it to the outer loop of the algorithm therefore presents clear benefits to model efficiency.

Originally, Boston and Britt proposed a more complicated relationship for Δh_{v}, involving computing pseudo-critical temperatures and pressures, as well as reduced temperatures [11]. The simpler temperature dependence shown in (2.33) is adopted by Parekh and Mathias in [26].

To summarize, the pQ flash algorithm by Boston and Britt takes the following form [11]:

Algorithm 1 Boston and Britt's algorithm for pQ flash calculations

```
1: Initialize $\mathbf{U} = [\phi, A, B, C, D, E, F]$, where $\phi$ is a row vector of the volatility parameters.
2: Initialize $R$
3: while $\left\|\mathbf{U} - \hat{\mathbf{U}}\right\|_{\infty} > \epsilon_{\mathrm{U}}$ do
4:     Set $\mathbf{U} \leftarrow \hat{\mathbf{U}}$
5:     while $|\Psi| > \epsilon_{\Psi}$ do
6:         Calculate $\mathbf{x}, \mathbf{y}, L$ and $\mathbf{r}$ using Equations (2.28)-(2.26).
7:         Find $K_r$ using (2.27) and use this to calculate the temperature in (2.32).
8:         Calculate the enthalpy departures using Equations (2.33) and (2.34).
9:         Calculate $\Psi$ and update $R$.
10:    end while
11:    Calculate $\mathbf{K}$ and $\Delta h_{\mathrm{l/v}}$ using rigorous thermodynamic models.
12:    Calculate $\hat{\mathbf{U}} = \left[\hat{\phi}, \hat{A}, \hat{B}, \hat{C}, \hat{D}, \hat{E}, \hat{F}\right]$ using Equations (2.32)-(2.34) and updated values for $\mathbf{K}$ and $\Delta h_{\mathrm{l/v}}$.
13: end while
```

Here $\|\mathbf{x}\|_{\infty}$ is the infinity norm of $\mathbf{x} \in \mathbb{R}^n$, which is defined as:

$$\|\mathbf{x}\|_{\infty} \equiv \max\{|x_1|, \dots, |x_n|\}. \tag{2.35}$$

Chapter 3

Simulation frameworks

Flowsheet models are sets of equations (e.g. the conservation laws and thermodynamic relations) that accurately describe the behaviour of processes [8], thus providing a fundamental tool for design and optimization. How these equations are organized and solved by the program normally determines its efficiency and robustness. Therefore, deciding on a simulation framework is an important step in process modeling. This chapter presents the two main approaches for flowsheet simulation, namely sequential-modular (SM) and equation-oriented (EO) frameworks. It also mentions a third, hybrid strategy, before presenting a method for calculating sensitivities of modular subroutines. In Section 3.4, issues with nonsmoothness in the MHEX model are elaborated. For a detailed comparison between the EO and SM frameworks, see the specialization project [40].

3.1 Sequential-modular framework

Sequential-modular flowsheets organize the model into smaller sub-processes (e.g. separation, expansion valve, reactor etc.) along with their corresponding equations. In each module, the output stream variables are computed based on input streams and design specifications [12]. Hence, it will be influenced by upstream processes, and the solver proceeds in the same direction as the information flow in the flowsheet.

Recycles represent a reflux of information that complicates the input/output relationships in the model. In such cases, a stream will not only be influenced by upstream processes, but by all units in the same loop. Necessarily, one pass through the flowsheet is generally not sufficient in this case, instead requiring the use of an

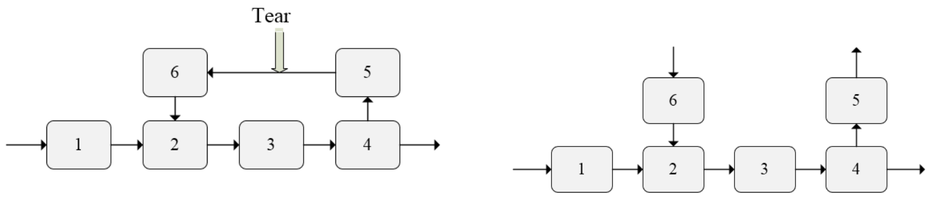


Figure 3.1: Tearing recycle streams to yield an acyclic network [40].

iterative approach for the simulation problem. Practically, this is done by "tearing" streams in each recycle to generate an acyclic network, which is solved by successive passes until the tear stream variables (input variables to module 6 and output variables from module 5 in Figure 3.1) converge.

Each calculation pass corresponds to solving a set of equations

$$\mathbf{w} = \mathbf{t}\left(\mathbf{x}, \mathbf{w}\right), \tag{3.1}$$

for the tear variables $\mathbf{w}$. Here $\mathbf{x}$ is used for other inputs to the simulation model. The tear equations (3.1) are not defined explicitly, but result from sequential solution of the modules between ends of the torn stream. They may be solved using any iterative method [6].

The number of tear variables needed for each tear stream is found by applying Duhem's theorem, which says that the state of a closed system at equilibrium with known masses for the components can be completely determined by fixing two independent variables [37]. Thus, for a system with C components, a total of $C+2$ independent variables need to be fixed. Although originally developed for closed systems, the theorem is also applicable to streams at steady state [6], such that each tear stream needs $C+2$ tear variables.

In process optimization, tear equations may be included directly in the optimization problem as equality constraints yielding a nonlinear program (NLP) of the form [8]:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{w}} \quad & f\left(\mathbf{x},\mathbf{w}\right) \\ \text{subject to:} \quad & \mathbf{h}\left(\mathbf{x},\mathbf{w}\right) = \mathbf{0} \\ & \mathbf{g}\left(\mathbf{x},\mathbf{w}\right) \leq \mathbf{0} \\ & \mathbf{w} - \mathbf{t}\left(\mathbf{x},\mathbf{w}\right) = \mathbf{0} \\ \mathbf{x} \in X \subset \mathbb{R}^{n_x}, \quad & \mathbf{w} \in W \subset \mathbb{R}^{n_w} \end{aligned} \tag{3.2}$$

where $\mathbf{x}$ are the decision variables, and $\mathbf{w}$ are the tear variables in the problem. The functions $\mathbf{h}$ and $\mathbf{g}$ are constraints in the model, for instance quality specifications or raw material supply. The program formulation in (3.2) is known as the infeasible path approach and can be solved effectively for a local solution with a Newton-based NLP solver like sequential quadratic programming (SQP) [9, 10]. With this approach, flowsheet convergence is not necessary at each intermediate step, but is instead achieved automatically at the optimum [9].

3.2 Equation-oriented framework

The equation-oriented strategy works as follows. Rather than partitioning the flowsheet into smaller subroutines, it gathers all the model equations and rewrites them in the form

$$\mathbf{f}\left(\mathbf{x}\right) = \mathbf{0}, \tag{3.3}$$

which can be solved simultaneously with a root-finding algorithm (e.g. a Newton-type method) [6]. Equation (3.3) is typically highly nonlinear, and finding initial guesses in the region of local convergence can be quite challenging, especially for large models [6]. Also, by pooling all the equations together, EO models eliminate the possibility of using specialized procedures for initializing and solving particular subroutines, which is why they tend to be less robust than their SM counterparts [6]. On the other hand, stream tearing and repeated acyclic flowsheet passes are no longer necessary, making them generally more efficient.

The equation-oriented model can be implemented in an optimization program by including (3.3) as equality constraints:

$$\begin{aligned} \min_{\mathbf{x}} \quad & f_{\mathrm{o}}\left(\mathbf{x}\right) \\ \text{subject to:} \quad & \mathbf{g}\left(\mathbf{x}\right) \leq \mathbf{0} \\ & \mathbf{h}\left(\mathbf{x}\right) = \mathbf{0} \\ & \mathbf{f}\left(\mathbf{x}\right) = \mathbf{0} \\ & \mathbf{x} \in X \subset \mathbb{R}^{n_x} \end{aligned} \tag{3.4}$$

where f_{o} here is used to denote the objective function, and $\mathbf{f}$ the model equations.

3.2.1 Hybrid framework

Hybrid frameworks combine the robustness of the sequential-modular approach with the efficiency of the equation-oriented method by modularizing particularly challenging sub-processes. Instead of solving the entire flowsheet simultaneously, certain parts are handled sequentially at every Newton-step resulting in greater robustness. No stream tearing and acyclic flowsheet simulation is necessary in hybrid models, thus preserving the main advantage of using an EO approach. Furthermore, custom initialization and solver algorithms can readily be implemented for the nested subroutines. On the other hand, by including subroutines in an equation-oriented framework that is solved with a Newton-type method, input-output sensitivities (i.e., derivative info) for the modules are required. Calculating these sensitivities can be quite time-consuming, especially for nonsmooth modules or if finite differencing is used (see Section 3.3).

3.3 Acquiring analytical sensitivities

Correct derivative information is needed for the objective function and constraints in order to optimize flowsheet models. However, unlike the equation-oriented approach where relations are expressed analytically, sequential-modular and hybrid models are defined (fully or partially) by the implicit input-output relationships in modular subroutines. As a consequence, derivative information for the modules must be found before these two frameworks can be implemented in an optimization procedure. Furthermore, like the equation-oriented approach, hybrid models normally employ a Newton-method for simulating the flowsheet. This requires accurate derivatives to be defined for all the variables that occur in the model, including those that are output variables from nested subroutines.

Though derivatives can be approximated numerically using finite differences, i.e.,

$$\mathbf{Jf}\left(\mathbf{x}\right)\mathbf{d} \approx \mathbf{f}\left(\mathbf{x}+\mathbf{d}\right) - \mathbf{f}\left(\mathbf{x}\right) \tag{3.5}$$

this approach introduces a truncation error $O\left(\mathbf{d}\right)$, which may lead to poor search directions and early termination of the NLP algorithm [8]. Wolbert et al. [44, 45] also found that analytical derivatives leads to considerable time savings in an optimization framework. Instead of using finite differences therefore, sensitivities ought to be calculated analytically by exploiting the implicit function theorem [44, 45].

Every module i in a modular flowsheet is regarded as its own sub-process described by a system of equations $\mathbf{f}_i : W \subset \mathbb{R}^{m+n} \rightarrow \mathbb{R}^n$

$$\mathbf{f}_i\left(\mathbf{x},\mathbf{y}\right) = \mathbf{0} \tag{3.6}$$

that can be solved for the output variables $\mathbf{y} \in \mathbb{R}^n$ with a Newton type method. Here W is open, and $\mathbf{x} \in \mathbb{R}^m$ is used collectively for the inputs to the unit. Analytical derivatives for $\mathbf{y}$ in Equation (3.6) follows directly from the implicit function theorem.

Theorem 3.1. *(Rudin [34, Theorem 9.27]). Let $\mathbf{f}_i$ be continuously differentiable $\left(\mathcal{C}^1\right)$ in W and $\mathbf{f}_i\left(\hat{\mathbf{x}},\hat{\mathbf{y}}\right) = \mathbf{0}$ for some $\left(\hat{\mathbf{x}},\hat{\mathbf{y}}\right) \in W$. Provided the partial Jacobian matrix*

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{y}}\left(\hat{\mathbf{x}},\hat{\mathbf{y}}\right) \tag{3.7}$$

is nonsingular, then for each $\mathbf{x}$ *in a neighborhood* N *of* $\hat{\mathbf{x}}$ *there corresponds a unique* $\mathbf{y}$ *near* $\hat{\mathbf{y}}$ *that also satisfies* (3.6)*. That is, there exists a* $\mathcal{C}^1$ *function*

$$\mathbf{g} : N \subset \mathbb{R}^m \to \mathbb{R}^n \tag{3.8}$$

such that

$$\mathbf{f}_i\left(\mathbf{x}, \mathbf{g}\left(\mathbf{x}\right)\right) = \mathbf{0},\ \forall \mathbf{x} \in N. \tag{3.9}$$

Moreover,

$$\frac{d\mathbf{g}}{d\mathbf{x}}\left(\hat{\mathbf{x}}\right) = -\left[\frac{\partial \mathbf{f}_i}{\partial \mathbf{y}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right)\right]^{-1} \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right). \tag{3.10}$$

Expression (3.10) in Theorem 3.1 is produced by differentiating (3.9) using the classical chain rule:

$$\frac{d\mathbf{f}_i}{d\mathbf{x}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right) = \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right) + \frac{\partial \mathbf{f}_i}{\partial \mathbf{y}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right) \frac{d\mathbf{g}}{d\mathbf{x}}\left(\hat{\mathbf{x}}\right) = \mathbf{0}, \tag{3.11}$$

implying that

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{y}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right) \frac{d\mathbf{g}}{d\mathbf{x}}\left(\hat{\mathbf{x}}\right) = -\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right), \tag{3.12}$$

which gives

$$\frac{d\mathbf{g}}{d\mathbf{x}}\left(\hat{\mathbf{x}}\right) = -\left[\frac{\partial \mathbf{f}_i}{\partial \mathbf{y}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right)\right]^{-1} \frac{\partial \mathbf{f}_i}{\partial \mathbf{x}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right). \tag{3.13}$$

It provides a method for computing analytical derivatives for modules whenever the partial Jacobian $\frac{\partial \mathbf{f}_i}{\partial \mathbf{y}}\left(\hat{\mathbf{x}}, \mathbf{g}\left(\hat{\mathbf{x}}\right)\right)$ is nonsingular (i.e. it has an inverse) and $\mathbf{f}_i$ is continuously differentiable.

Definition 3.1. *(From Rudin [34]). Let* $D \subset \mathbb{R}^n$ *be open and* $f : D \to \mathbb{R}$*. Then,* f *is continuously differentiable on* D *if the partial derivatives*

$$\nabla f\left(\mathbf{x}\right) = \begin{bmatrix} \frac{\partial f}{\partial x_1}\left(\mathbf{x}\right) \\ \vdots \\ \frac{\partial f}{\partial x_n}\left(\mathbf{x}\right) \end{bmatrix}$$

exist and are continuous on D*.*

Continuously differentiable functions are often denoted by $\mathcal{C}^n$, where n represents an order at which the partial derivatives are defined and continuous.

Example 3.1. *Given a function*

$$f : \mathbb{R} \to \mathbb{R} : x \to x^2,$$

Then, f *will be* C^∞ *as all its higher-order derivatives exist and are continuous on* X*:*

$$\frac{df}{dx}\left(x\right) = 2x,$$

$$\frac{d^2 f}{dx^2} = 2,$$

and

$$\frac{d^k f}{dx^k} = 0$$

for all positive integers k greater than or equal to three.

On the other hand, the function

$$F : D \subset \mathbb{R} \to \mathbb{R} : x \to \int_0^x |u-1| du$$

is only C^1 on the domain $D = (0, 2)$ as its derivative

$$\frac{dF}{dx}(x) = |x-1|$$

is nondifferentiable at $x = 1$.

3.4 Issues with nonsmoothness

The assumption of continuously differentiable functions cannot be extended to the MHEX model. Equations (2.4), (2.6) and (2.16) are all nonsmooth, meaning that the derivatives are not defined everywhere on their domain. The nonsmoothness is due to max- and min-functions, which have undefined derivatives whenever the arguments are equal.
Example 3.2. *Let $f : \mathbb{R} \to \mathbb{R}$ be the function*

$$f(x) = max\{0, x\}.$$

Then f will be nondifferentiable at $x = 0$ as

$$\frac{df}{dx}(x) = \begin{cases} 0, & for\, x < 0, \\ 1, & for\, x > 0. \end{cases}$$

That is, the derivative of f does not exist at $x = 0$, and the function is said to have a kink at this point.

Max-functions occur, for instance, in the expressions for the enthalpies of the extended composite curves (Equations (2.7) and (2.8)), thereby causing (2.6) to become nonsmooth. Also, to account for the possibility of a stream entering or

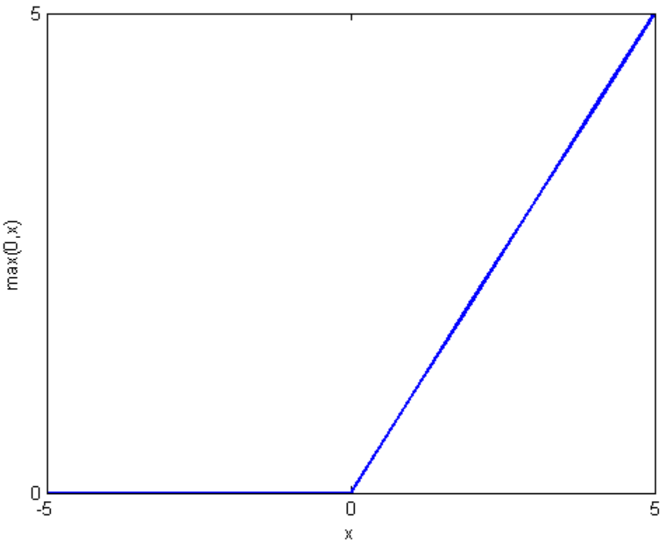


Figure 3.2: The max function.

exiting the MHEX in any phase, the following max, min, and mid expressions are used for the inlet and outlet temperatures [41]:

$$\begin{aligned} T_{\text{sup}}^{\text{in}} &= \max\left(T_{\text{DP}}, T^{\text{in}}\right), \\ T_{\text{2p}}^{\text{in}} &= \text{mid}\left(T_{\text{DP}}, T_{\text{BP}}, T^{\text{in}}\right), \\ T_{\text{sub}}^{\text{in}} &= \min\left(T_{\text{BP}}, T^{\text{in}}\right), \end{aligned} \tag{3.14}$$

and:

$$\begin{aligned} T_{\text{sup}}^{\text{out}} &= \max\left(T_{\text{DP}}, T^{\text{out}}\right), \\ T_{\text{2p}}^{\text{out}} &= \text{mid}\left(T_{\text{DP}}, T_{\text{BP}}, T^{\text{out}}\right), \\ T_{\text{sub}}^{\text{out}} &= \min\left(T_{\text{BP}}, T^{\text{out}}\right), \end{aligned} \tag{3.15}$$

where BP and DP are the bubble- and dew-point, and sup, 2p and sub denote the superheated, two-phase and subcooled regions, respectively. The variables $T^{\text{in/out}}$ are the stream inlet/outlet temperatures to the MHEX.

Generally, nonsmoothness causes issues for Newton solvers and derivative-based optimization techniques. To cope with points of nondifferentiability, smoothing-approximations are frequently used that approximate the kink as a $\mathcal{C}^1$ function.

One such approximation for the max function is

$$\max\{0, f(x)\} \approx \frac{\left(\sqrt{f(x)^2 + \beta^2} + f(x)\right)}{2}, \tag{3.16}$$

where β is a user-defined parameter [5].

Alternatively, nonsmooth Newton-type solvers (e.g. the nonsmooth Newton method by Qi and Sun in (2.18)) and optimization algorithms (e.g bundle methods [19]) have been developed that use generalized derivatives in place of conventional derivatives. Generalized derivatives are defined as extensions of the concept of derivatives to some classes of nondifferentiable functions, e.g. piecewise continuously differentiable ($\mathcal{PC}^1$) functions, which include max and min. Importantly, generalized derivatives are reduced to conventional derivatives for continuously differentiable functions. Recent developments in nonsmooth analysis by the Process Systems Engineering Laboratory at Massachusetts Institute of Technology are centered around calculating generalized derivatives for nonsmooth systems and is discussed in more detail in the next chapter.

Chapter 4

Mathematical theory

This chapter presents a methodology for calculating the generalized derivatives needed to simulate or optimize the nonsmooth MHEX model. The procedure focuses on one type of generalized derivatives, namely the Clarke Jacobian. However, computing elements of the Clarke Jacobian directly is difficult, and thus this chapter presents an alternate route through lexicographic derivatives and the lexicographic-directional derivative. Unlike the Clarke Jacobian in general, lexicographic-directional derivatives follow sharp calculus rules such that they are computationally tractable. This chapter also presents a nonsmooth implicit function theorem that can be used to compute lexicographic-directional derivatives for piecewise continuously differentiable subroutines. Lastly, Section 4.6 focuses on a numerical procedure for calculating the lexicographic-directional derivatives by exploiting the procedural operations in a computer.

4.1 $\mathcal{PC}^1$ functions

Definition 4.1. *(From Scholtes [36]). Given a function $\mathbf{f} : X \to \mathbb{R}^m$ for an open set $X \subset \mathbb{R}^n$, and some $\mathbf{x} \in X$. Then $\mathbf{f}$ is piecewise continuously differentiable $\left(\mathcal{PC}^1\right)$ at $\mathbf{x}$ if there exist a neighborhood $N \subset X$ of $\mathbf{x}$ and a finite collection $\mathcal{F}_\mathbf{f}(\mathbf{x})$ of $\mathcal{C}^1$ selection functions mapping N into $\mathbb{R}^m$, where $\mathbf{f}$ is continuous and*

$$\mathbf{f}(\mathbf{y}) \in \{\phi(\mathbf{y}) : \phi \in \mathcal{F}_\mathbf{f}(\mathbf{x})\}, \quad \forall \mathbf{y} \in N.$$

Furthermore, if each selection function $\phi \in \mathcal{F}_\mathbf{f}(\mathbf{x})$ is linear, then function $\mathbf{f}$ is piecewise linear $(\mathcal{PL})$ at $\mathbf{x}$. If each selection function $\phi \in \mathcal{F}_\mathbf{f}(\mathbf{x})$ is affine, then *function* $\mathbf{f}$ *is piecewise affine* ($\mathcal{PA}$) *at* $\mathbf{x}$*. If* $\mathbf{f}$ *is* $\mathcal{PC}^1$ *at* $\mathbf{x}$ *for each* $\mathbf{x} \in X$ *then* $\mathbf{f}$ *is* $\mathcal{PC}^1$ *on* X*.*

In Definition 4.1, the term selection function is used for elements of the set of $\mathcal{C}^1$ functions that makes up the $\mathcal{PC}^1$ function.
Example 4.1. *Consider the function*

$$f : \mathbb{R} \to \mathbb{R} : x \to |x|.$$

For $x = 0$*,* $\mathcal{F}_f(0)$ *consists of two different selection functions:*

$$\begin{aligned} \phi_1(x) &= -x, \ \ \forall x \in (-\infty, 0] \\ \phi_2(x) &= x, \ \ \forall x \in [0, \infty); \end{aligned}$$

f *is* $\mathcal{PC}^1$ *on* $\mathbb{R}$*.*

Definition 4.1 distinguishes between linear and affine functions. The first is any mapping that satisfies the vector addition and scalar multiplication properties:

$$f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}), \tag{4.1}$$

$$f(a\mathbf{x}) = a \cdot f(\mathbf{x}), \quad \forall a \in \mathbb{R}. \tag{4.2}$$

Linear functions mapping $\mathbb{R}^n$ to $\mathbb{R}$ take the form:

$$f(\mathbf{x}) = \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{4.3}$$

for some $\mathbf{c} \in \mathbb{R}^n$.

Affinity, on the other hand, originates from convex analysis and is defined as functions that are both concave and convex.
Definition 4.2. *(From Barton [7, Definition 3.1]). A set* $C \subset \mathbb{R}^n$ *is convex if the line connecting any two points* $\mathbf{x}, \mathbf{y} \in C$ *lies entirely in* C*. Thus, for any two points* $\mathbf{x}, \mathbf{y} \in C$*, the points* $\mathbf{z}$ *defined by*

$$\mathbf{z} \equiv \lambda\mathbf{x} + (1 - \lambda)\mathbf{y}, \ \forall \lambda \in [0, 1],$$

must also be in C*.*
Definition 4.3. *(From Barton [7, Definition 3.16 and Definition 3.19].) Let* $C \subset \mathbb{R}^n$ *be a non-empty convex set. Then a function* $f : C \to \mathbb{R}$ *is convex if the following inequality holds for each* $\mathbf{x}, \mathbf{y} \in C$*:*

$$f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}), \quad \forall \lambda \in (0, 1).$$

A function is concave if instead the inequality

$$f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y}) \geq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}), \quad \forall \lambda \in (0, 1),$$

is satisfied for all $\mathbf{x}, \mathbf{y} \in C$ *[7].*

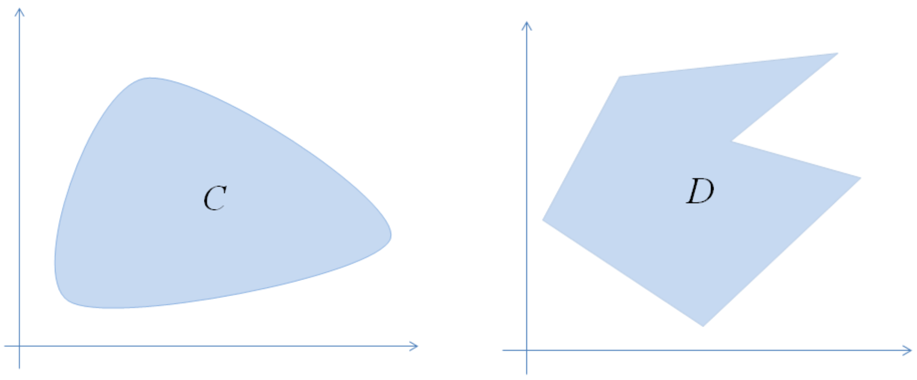


Figure 4.1: A convex set C and a nonconvex set D.

Clearly, affine functions must satisfy both inequalities by equality, which is only true for functions of the form:

$$f(\mathbf{x}) = \mathbf{c}^{\mathrm{T}}\mathbf{x} + b. \tag{4.4}$$

Example 4.2. *Let $f : C \subset \mathbb{R}^n \to \mathbb{R}$ be an affine function*

$$f(\mathbf{x}) = \mathbf{c}^{T}\mathbf{x} + b.$$

Then neither the vector addition property (4.1)*:*

$$\begin{aligned} f(\mathbf{x}+\mathbf{y}) &= \mathbf{c}^{T}(\mathbf{x}+\mathbf{y}) + b, \\ &= \mathbf{c}^{T}\mathbf{x} + \mathbf{c}^{T}\mathbf{y} + b, \\ &\neq f(\mathbf{x}) + f(\mathbf{y}), \end{aligned}$$

nor the scalar multiplication property:

$$\begin{aligned} f(a\mathbf{x}) &= \mathbf{c}^{T}(a\mathbf{x}) + b, \\ &= a \cdot \mathbf{c}^{T}\mathbf{x} + b, \\ &\neq a \cdot f(\mathbf{x}), \end{aligned}$$

will be satisfied. Instead, these properties are only true for affine functions when $b = 0$, i.e. the linear function form.

4.2 Clarke's Jacobian and plenary hulls

Before presenting a tractable way of computing generalized derivatives for $\mathcal{PC}^1$ functions, the concepts of convex hulls, Lipschitz continuity, and B-subdifferential are introduced.

The convex hull of a set D, denoted by $\operatorname{conv}(D)$, is the smallest convex superset of D [7] (see Figure 4.2). In the case of D being convex, the convex hull is equal to D itself.

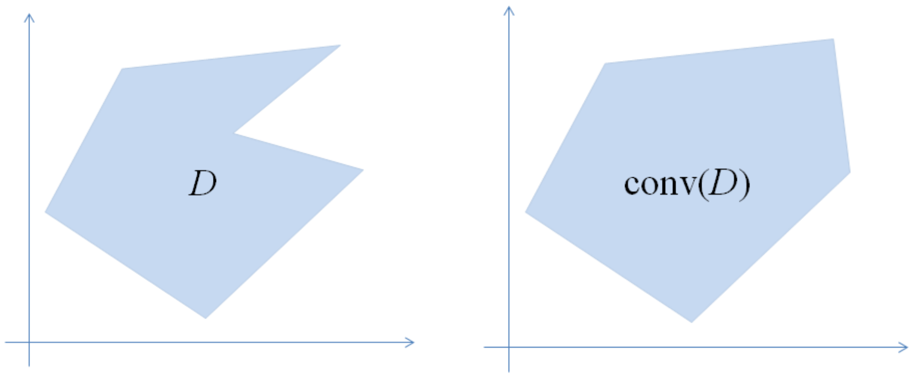


Figure 4.2: The convex hull of set D

Definition 4.4. *A function $\mathbf{f} : X \subset \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz continuous on X if there exists an $L \geq 0$ such that*

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|, \ \forall \mathbf{x}, \mathbf{y} \in X.$$

Lipschitz continuity is a strong form of uniform continuity, in that the function is limited in how fast it can change. Its rate of change is bounded on X by L, also known as the Lipschitz constant for the function.
Definition 4.5. *A function $\mathbf{f} : X \subset \mathbb{R}^n \to \mathbb{R}^m$ is locally Lipschitz continuous on X, if for each $\mathbf{x} \in X$, there exists a neighbourhood N of $\mathbf{x}$ such that $\mathbf{f}$ is Lipschitz continuous on N.*

For example, the function

$$f(x) = \frac{1}{\sqrt{x}}, \ \forall x \in (0, +\infty), \tag{4.5}$$

will not be Lipschitz continuous on the interval $(0, +\infty)$ as its slope approaches $(-\infty)$ when x goes to zero (see Figure 4.3). However, f is locally Lipschitz continuous on X as each neighbourhood N can be chosen infinitesimally small such that the function's gradient will be bounded on that interval. On the other hand, the function will not be locally Lipschitz continuous on $[0, +\infty)$, as f is nondifferentiable at $x = 0$.

Piecewise continuously differentiable functions consist of a finite set of $\mathcal{C}^1$ selection functions $\phi(\mathbf{x})$ that have bounded derivatives everywhere. Therefore, such functions will be locally Lipschitz continuous on their domains.

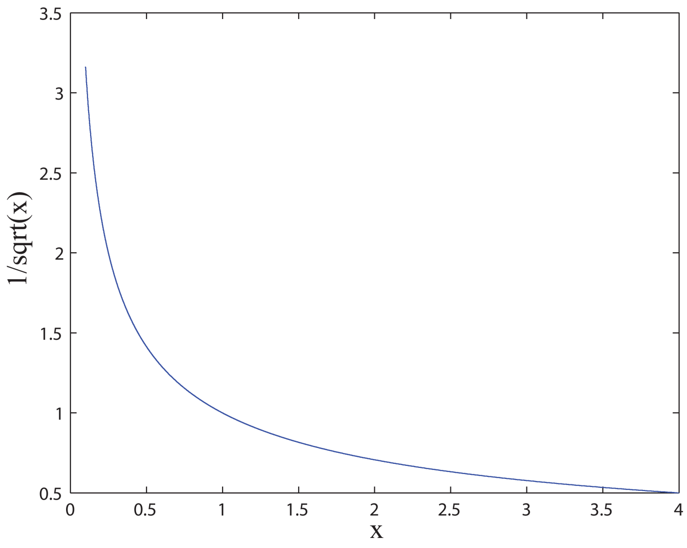


Figure 4.3: The graph of $1/\sqrt{x}$.

Definition 4.6. *(From [13] and [31]). Given a locally Lipschitz continuous function $\mathbf{f} : X \to \mathbb{R}^m$ on an open set $X \subset \mathbb{R}^n$, let $Y \subset X$ be the set where $\mathbf{f}$ is differentiable. Then, the B-subdifferential of $\mathbf{f}$ at $\mathbf{x} \in X$ is defined as:*

$$\partial_B \mathbf{f}(\mathbf{x}) \equiv \left\{ \mathbf{A} \in \mathbb{R}^{m\times n} : \mathbf{A} = \lim_{j\to\infty} \mathbf{Jf}(\mathbf{x}_j), \quad \mathbf{x} = \lim_{j\to\infty} \mathbf{x}_j, \quad \mathbf{x}_j \in Y, \quad \forall j \in \mathbb{N} \right\}.$$

In other words, the B-subdifferential at a point $\mathbf{x} \in X$ is a set of matrices $\partial_{\mathrm{B}} \mathbf{f}(\mathbf{x}) \subset \mathbb{R}^{m\times n}$, containing the Jacobians of $\mathbf{f}$ as $\mathbf{x}$ is approached from any direction of differentiability.

Example 4.3. *Let $f(x) = |x|$, a $\mathcal{PC}^1$ function on $\mathbb{R}$ whose graph is shown in Figure 4.4.*

Considering points $\{-2, 0, 2\}$ in Figure 4.4, then the B-subdifferential for Point 1 will be the set

$$\partial_B f(-2) = \{-1\},$$

as the slope remains the same when the point is approached from either direction.

Similarly, the B-subdifferential at $x = 2$ is

$$\partial_B f(2) = \{1\}.$$

At the origin, however, the Jacobians change depending on whether we approach the point from the left or from the right. There exists a nondifferentiable point, and

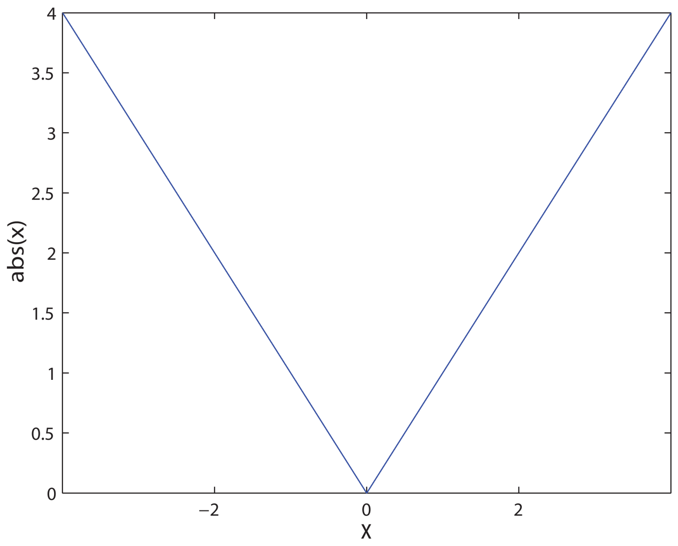


Figure 4.4: The absolute function.

the B-subdifferential will contain all the Jacobians in its neighbourhood:

$$\partial_B f(0) = \{-1, 1\}.$$

One type of generalized derivatives are the elements of the Clarke Jacobian ($\partial \mathbf{f}(\mathbf{x})$), which is defined as the convex hull of the B-subdifferential. Its disadvantage is that it satisfies the classical calculus rules only as inclusions rather than equations [13], which makes it difficult to evaluate its elements. Therefore, this chapter presents an alternative method that instead calculates objects of the plenary hull of the Clarke Jacobian using lexicographic (L-) derivatives.

The plenary hull of a set $A \subset \mathbb{R}^{m \times n}$ is defined as [38]:

$$\mathrm{plen}(A) \equiv \left\{\mathbf{H} \in \mathbb{R}^{m\times n} : \forall \mathbf{d} \in \mathbb{R}^n,\ \exists \mathbf{A} \in A\ s.t.\ \mathbf{Hd} = \mathbf{Ad}\right\}. \tag{4.6}$$

Moreover, a set is plenary if it equals its plenary hull. Given $\mathbf{G} \in \partial \mathbf{f}(\mathbf{x}) \subset \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$, solving the equation system

$$\mathbf{Gy} = \mathbf{b}, \tag{4.7}$$

is equvalent to solving the equation system

$$\mathbf{Hy} = \mathbf{b},\ \mathbf{H} \in \mathrm{plen}(\partial \mathbf{f}(\mathbf{x})). \tag{4.8}$$

Nonsmooth Newton-type methods are of the form:

$$\mathbf{G}\left(\mathbf{x}^k\right)\left(\mathbf{x}^{(k+1)} - \mathbf{x}^k\right) = -\mathbf{f}\left(\mathbf{x}^k\right), \tag{4.9}$$

where $\mathbf{G}(\mathbf{x})$ is a generalized derivative element, e.g. an element of $\partial\mathbf{f}(\mathbf{x})$. Equations (4.7)-(4.8) guarantee there exist matrices $\mathbf{H} \in \text{plen}\left(\partial\mathbf{f}\left(\mathbf{x}^k\right)\right)$, also referred to as the plenary Jacobian [38], that will produce an image identical to $\mathbf{G}\left(\mathbf{x}^k\right)\mathbf{x}^k$ and $\mathbf{G}\left(\mathbf{x}^k\right)\mathbf{x}^{(k+1)}$. As a result, any element of the plenary Jacobian is just as useful as a Clarke Jacobian element for nonsmooth equation solving [23]. Moreover, as the plenary hull is a superset of the Clarke Jacobian, its elements are more easily obtainable.

4.3 Lexicographic derivatives

Lexicographic derivatives (L-derivatives) were first introduced by Nesterov [25], and were later proved by Khan and Barton [21] to be elements of plen $(\partial\mathbf{f}(\mathbf{x}))$ whenever they exist. Lexicographic means "in the order it would appear in a dictionary" and is used here to describe the hierarchical order in which directional derivatives are computed. Analogous to conventional derivatives, the L-derivative for a function $\mathbf{f} : X \to \mathbb{R}^m$ only exist when $\mathbf{f}$ is lexicographically smooth [23, 25].

Definition 4.7. *Let $\mathbf{f} : X \subset \mathbb{R}^n \to \mathbb{R}^m$, where X is open. The directional derivative at $\mathbf{x} \in X$ in direction $\mathbf{d} \in \mathbb{R}^n$ is the rate of change of $\mathbf{f}(\mathbf{x})$ as it is perturbed in said direction.*

$$\mathbf{f}'(\mathbf{x};\mathbf{d}) \equiv \lim_{h\to 0^+} \frac{\mathbf{f}(\mathbf{x}+h\mathbf{d}) - \mathbf{f}(\mathbf{x})}{h}.$$

Definition 4.8. *A locally Lipschitz continuous function $\mathbf{f} : X \subset \mathbb{R}^n \to \mathbb{R}^m$, where X is open, is said to be lexicographically smooth (L-smooth) at a point $\mathbf{x} \in X$ if it is directionally differentiable at $\mathbf{x}$ and its higher-order directional derivatives are well-defined for any $\mathbf{M} = [\mathbf{m}_1 \ldots \mathbf{m}_k] \in \mathbb{R}^{n\times k}$ and $k \in \mathbb{N}$:*

$$\begin{aligned}
\mathbf{f}^0_{\mathbf{x},\mathbf{M}} &: \mathbb{R}^n \to \mathbb{R}^m : \mathbf{d} \to \mathbf{f}'(\mathbf{x};\mathbf{d}) \\
\mathbf{f}^1_{\mathbf{x},\mathbf{M}} &: \mathbb{R}^n \to \mathbb{R}^m : \mathbf{d} \to \left[\mathbf{f}^0_{\mathbf{x},\mathbf{M}}\right]'(\mathbf{m}_1;\mathbf{d}) \\
&\vdots \\
\mathbf{f}^k_{\mathbf{x},\mathbf{M}} &: \mathbb{R}^n \to \mathbb{R}^m : \mathbf{d} \to \left[\mathbf{f}^{k-1}_{\mathbf{x},\mathbf{M}}\right]'(\mathbf{m}_k;\mathbf{d})
\end{aligned}$$

Lexicograpical smoothness is possible even at nondifferentiable points, provided the directional derivatives are defined.

Example 4.4. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a $\mathcal{PC}^1$ function:*

$$f(x_1, x_2) = max(x_1, x_2),$$

whose graph is presented in Figure 4.5.

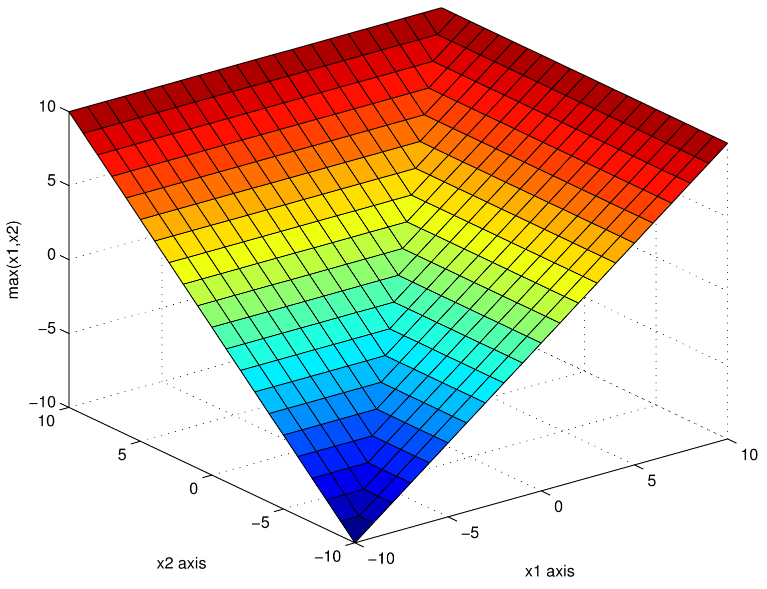


Figure 4.5: The maximum value function.

For each point $\mathbf{x} \in \mathbb{R}^2$, the directional derivative from perturbing $\mathbf{x}$ in direction:

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix},$$

is:

$$f'(x_1, x_2; \mathbf{d}) = \lim_{h \to 0^+} \frac{f(x_1 + hd_1, x_2 + hd_2) - f(x_1, x_2)}{h}.$$

For points $x_1 \neq x_2$, f is continuously differentiable, which implies it is also lexicographically smooth [25]. At $x_1 = x_2$, on the other hand, the derivative is undefined as there exists a kink in the graph. Fortunately, the directional derivative can still *be computed from Definition 4.7:*

$$\begin{aligned} f'(x_1, x_2; \mathbf{d}) &= \lim_{h\to 0^+} \frac{f(x_1 + hd_1, x_2 + hd_2) - f(x_1, x_2)}{h} \\ &= \lim_{h\to 0^+} \frac{max(x_1 + hd_1, x_2 + hd_2) - max(x_1, x_2)}{h} \\ &= \lim_{h\to 0^+} \frac{h \cdot max(d_1, d_2)}{h} = max(d_1, d_2). \end{aligned}$$

Obviously, the new mapping $f^0_{\mathbf{x},\mathbf{M}}(d_1, d_2) = max(d_1, d_2)$ is also directionally differentiable everywhere; higher-order directional derivatives exist, and the function is L-smooth according to Definition 4.8.

Lexicographic derivatives are best explained as generalizations of standard derivatives. More specifically, they are defined as the Jacobian of the mapping $\mathbf{f}^n_{\mathbf{x},\mathbf{M}} : \mathbb{R}^n \to \mathbb{R}^m$ evaluated at $\mathbf{0}_n$, which is continuously differentiable provided the directions matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ is nonsingular [25]. Lexicographic derivatives are denoted as $\mathbf{J}_\mathrm{L}\mathbf{f}(\mathbf{x}; \mathbf{M})$ and, unlike elements of the Clarke Jacobian, obey sharp calculus rules.

Example 4.5. *Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ in Example 4.4 evaluated at $\mathbf{x} = \mathbf{0}$. For an arbitrary nonsingular directions matrix $\mathbf{M}$*

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix},$$

the first-order directional derivative is (see Example 4.4)

$$f^0_{\mathbf{x},\mathbf{M}}(\mathbf{m}_1) = max(m_{11}, m_{21}),$$

which is the function we started with. If m_{11} is equal to m_{21}, then $m_{12} \neq m_{22}$ since $\mathbf{M}$ is nonsingular, and the calculation procedure is repeated. On the other hand, if $m_{11} \neq m_{21}$ the term $f^0_{\mathbf{x},\mathbf{M}}(\mathbf{m}_1)$ reduces to either m_{11} or m_{21}, depending on which is largest. Since $\mathbf{M}$ is assumed to be nonsingular, at least one of the columns $\mathbf{m}_k$ will have different entries. Let us say here that $m_{11} > m_{21}$ for illustration. Then, the second-order directional derivative will be

$$\begin{aligned} f^1_{\mathbf{x},\mathbf{M}}(\mathbf{m}_2) &= \left[f^0_{\mathbf{x},\mathbf{M}}\right]'(\mathbf{m}_1; \mathbf{m}_2) \\ &= \lim_{h\to 0^+} \frac{f^0_{\mathbf{x},\mathbf{M}}(m_{11} + h\cdot m_{12}, m_{21} + h \cdot m_{22}) - f^0_{\mathbf{x},\mathbf{M}}(m_{11}, m_{21})}{h} \\ &= \lim_{h\to 0^+} \frac{m_{11} + h\cdot m_{12} - m_{11}}{h} = m_{12}. \end{aligned}$$

Similarly

$$\begin{aligned} f^2_{\mathbf{x},\mathbf{M}}(\mathbf{d}) &= \left[f^1_{\mathbf{x},\mathbf{M}}\right]'(\mathbf{m}_1;\mathbf{d}) \\ &= \lim_{h\to 0^+} \frac{f^1_{\mathbf{x},\mathbf{M}}(m_{12}+hd_1, m_{22}+hd_2) - f^1_{\mathbf{x},\mathbf{M}}(m_{12},m_{22})}{h} \\ &= d_1, \end{aligned}$$

and the lexicographic derivative of f at $\mathbf{x}=\mathbf{0}$ is

$$\mathbf{J}f^2_{\mathbf{x},\mathbf{M}}(\mathbf{0}) = \begin{bmatrix} \dfrac{\partial f^2_{\mathbf{x},\mathbf{M}}}{\partial d_1}(\mathbf{0}) & \dfrac{\partial f^2_{\mathbf{x},\mathbf{M}}}{\partial d_2}(\mathbf{0}) \end{bmatrix} = [1 \;\; 0].$$

The set of lexicographic derivatives $\mathbf{J}_{\mathrm{L}}\mathbf{f}(\mathbf{x};\mathbf{M})$ at $\mathbf{x}$ for nonsingular directions matrices $\mathbf{M} \in \mathbb{R}^{n\times n}$ is known as the lexicographic subdifferential $\partial_{\mathrm{L}}\mathbf{f}(\mathbf{x})$:
Definition 4.9. *(From Nesterov [25]).*

$$\partial_L \mathbf{f}(\mathbf{x}) \equiv \left\{\mathbf{J}_L\mathbf{f}(\mathbf{x};\mathbf{M}) \;:\; \mathbf{M}\in\mathbb{R}^{n\times n}, det\mathbf{M} \neq 0\right\}.$$

The lexicographic subdifferential is a subset of the plenary hull of the Clarke Jacobian [21], and any element of $\partial_{\mathrm{L}}\mathbf{f}(\mathbf{x})$ can therefore substitute for $\mathbf{G}$ in Equation (4.9). In case of piecewise differentiable functions, Khan and Barton found the following to be true [23]:

$$\partial_{\mathrm{L}}\mathbf{f}(\mathbf{x}) \subset \partial_{\mathrm{B}}\mathbf{f}(\mathbf{x}) \subset \partial\mathbf{f}(\mathbf{x}). \tag{4.10}$$

That is, lexicographic derivatives are elements of the Clarke Jacobian itself rather than the plenary Jacobian. Nonsmooth Newton-type methods that employ B-subdifferential elements as generalized derivative information benefit from local quadratic convergence properties [23].

4.4 Lexicographic directional derivatives

Lexicographic derivatives are not computed directly, but are instead calculated by first finding a lexicographic directional (LD-)derivative. For an L-smooth function $\mathbf{f}: X \subset \mathbb{R}^n \to \mathbb{R}^m$, the LD-derivative of $\mathbf{f}$ at $\mathbf{x}\in X$ in directions $\mathbf{M}\in\mathbb{R}^{n\times k}$, $k\in\mathbb{N}$ is defined as [23]:

$$\mathbf{f}'(\mathbf{x};\mathbf{M}) \equiv \left[\mathbf{f}^0_{\mathbf{x},\mathbf{M}}(\mathbf{m}_1) \;\; \mathbf{f}^1_{\mathbf{x},\mathbf{M}}(\mathbf{m}_2) \;\; \cdots \;\; \mathbf{f}^{k-1}_{\mathbf{x},\mathbf{M}}(\mathbf{m}_k)\right]. \tag{4.11}$$

Equation (4.11) is used for computing the LD-derivatives:

Example 4.6. *The higher-order directional derivatives for the maximum value function at* $\mathbf{x} = \mathbf{0}$ *were computed in Example 4.5:*

$$\begin{aligned} f^0_{\mathbf{x},\mathbf{M}}(\mathbf{m}_1) &= m_{11} \\ f^1_{\mathbf{x},\mathbf{M}}(\mathbf{m}_2) &= m_{12}. \end{aligned}$$

Therefore, the lexicographic directional derivative is in this case the row vector

$$f'(\mathbf{0};\mathbf{M}) = \left[m_{11} \;\; m_{12}\right].$$

Where the L-derivative is a generalization of the conventional derivative, the LD-derivative is analogous to the standard directional derivative for smooth functions [23]. Provided the directions matrix $\mathbf{M}$ is nonsingular and square, the following relationship between the LD-derivative and L-derivative holds:

$$\mathbf{f}'(\mathbf{x};\mathbf{M}) = \mathbf{J}_{\mathrm{L}}\mathbf{f}(\mathbf{x};\mathbf{M})\,\mathbf{M}. \tag{4.12}$$

Example 4.7. *Considering the same maximum value function as in Examples 4.4-4.6 for a point* $\mathbf{x} = \mathbf{0}$*. For an arbitrary* 2×2 *nonsingular directions matrix* $\mathbf{M}$*, the LD-derivative was found in Example 4.6 to be:*

$$\mathbf{f}'(\mathbf{0};\mathbf{M}) = \left[m_{11} \;\; m_{12}\right].$$

Since $\mathbf{M}$ *is nonsingular,* $det\mathbf{M} \neq 0$ *and its inverse is given as*

$$\begin{aligned} \mathbf{M}^{-1} &= \frac{1}{det\mathbf{M}}\begin{bmatrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{bmatrix}, \\ &= \frac{1}{m_{11}m_{22} - m_{12}m_{21}}\begin{bmatrix} m_{22} & -m_{12} \\ -m_{21} & m_{11} \end{bmatrix}, \\ &= \begin{bmatrix} \dfrac{m_{22}}{m_{11}m_{22} - m_{12}m_{21}} & \dfrac{-m_{12}}{m_{11}m_{22} - m_{12}m_{21}} \\ \dfrac{-m_{21}}{m_{11}m_{22} - m_{12}m_{21}} & \dfrac{m_{11}}{m_{11}m_{22} - m_{12}m_{21}} \end{bmatrix}. \end{aligned}$$

The L-derivative can then be computed directly from Equation (4.12)*:*

$$\mathbf{J}_L\mathbf{f}(\mathbf{0};\mathbf{M}) = \mathbf{f}'(\mathbf{0};\mathbf{M})\,\mathbf{M}^{-1}.$$

Inserting for the inverted matrix:

$$\mathbf{J}_L\mathbf{f}\,(\mathbf{0};\mathbf{M}) = [m_{11}\;\; m_{12}] \begin{bmatrix} \frac{m_{22}}{m_{11}m_{22}-m_{12}m_{21}} & \frac{-m_{12}}{m_{11}m_{22}-m_{12}m_{21}} \\ \frac{-m_{21}}{m_{11}m_{22}-m_{12}m_{21}} & \frac{m_{11}}{m_{11}m_{22}-m_{12}m_{21}} \end{bmatrix},$$

$$= \begin{bmatrix} \frac{m_{11}m_{22}-m_{12}m_{21}}{m_{11}m_{22}-m_{12}m_{21}} & \frac{m_{11}m_{12}-m_{1}m_{12}}{m_{11}m_{22}-m_{12}m_{21}} \end{bmatrix} = [1\;\; 0]\,,$$

which is the same as in Example 4.5.

For a square and nonsingular directions matrix $\mathbf{M}$, $\partial_{\mathrm{L}}\mathbf{f}\,(\mathbf{x})$ is also guaranteed to be a part of the plenary hull of the Clarke Jacobian (see Section 4.3). Equation (4.12) represents a convenient way of finding L-derivatives for a given LD-derivative. Another advantage with using LD-derivatives is that they follow the strict chain rule such that LD-derivatives for composite functions can be found in the same fashion.

Theorem 4.1. *(From Khan and Barton [23]) Given open sets $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$, and locally Lipchitz continuous functions $\mathbf{f} : Y \to \mathbb{R}^q$ and $\mathbf{g} : X \to Y$ that are L-smooth at $\mathbf{g}\,(\mathbf{x}) \in Y$ and $\mathbf{x} \in X$. Then, the composition $\mathbf{f} \circ \mathbf{g}$ is L-smooth at $\mathbf{x}$, and for any directions matrix $\mathbf{M} \in \mathbb{R}^{n\times k}$, the following chain rule for LD-derivatives is satisfied:*

$$[\mathbf{f} \circ \mathbf{g}]'\,(\mathbf{x};\mathbf{M}) = \mathbf{f}'\,(\mathbf{g}\,(\mathbf{x})\,;\mathbf{g}'\,(\mathbf{x};\mathbf{M}))\,. \tag{4.13}$$

4.5 Implicit function theorem for lexicographically smooth functions

In Section 3.3, analytical solutions for the input-output sensitivities were formulated for $\mathcal{C}^1$ functions by means of the implicit function theorem. Fortunately, there exists an analogous theorem presented by Khan and Barton that is applicable to lexicographically smooth functions [22]. Before the theorem can be presented, however, the projection of the Clarke Jacobian must be defined.

Definition 4.10. *(From [13]) Let $\mathbf{f} : X \times Y \to \mathbb{R}^p$ for open sets $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$, and let it be locally Lipschitz continuous in a neighborhood of a point $(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \in X \times Y$. Then, the Clarke Jacobian projection of $\mathbf{f}$ at $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ with respect to $\mathbf{y}$ is defined as the set*

$$\pi_2\partial\mathbf{f}\,(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \equiv \left\{\mathbf{W}_2 \in \mathbb{R}^{p\times m} :\; \exists\mathbf{W}_1 \in \mathbb{R}^{p\times n} \quad s.t. \quad [\mathbf{W}_1 \quad \mathbf{W}_2] \in \partial\mathbf{f}\,(\hat{\mathbf{x}}, \hat{\mathbf{y}})\right\}.$$

Yunt [46, Theorem 2.6.12] also detailed the property $\partial_2\mathbf{f}\,(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \subset \pi_2\partial\mathbf{f}\,(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, where $\partial_2\mathbf{f}\,(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is the partial Clarke Jacobian with respect to $\mathbf{y}$ evaluated at $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$.

Theorem 4.2. *(From [22, Theorem 2]) Let $W \subset \mathbb{R}^{m+n}$ be open and $\mathbf{f} : W \to \mathbb{R}^n$ be L-smooth at a point $(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \in W$ that satisfies*

$$\mathbf{f}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = \mathbf{0}.$$

In addition, assume the projection $\pi_2 \partial \mathbf{f}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ contains no singular matrices. Then, there exist a neighbourhood $N \subset \mathbb{R}^m$ of $\hat{\mathbf{x}}$, and a Lipschitz continuous function $\mathbf{g} : N \to \mathbb{R}^n$ such that $\hat{\mathbf{y}} = \mathbf{g}(\hat{\mathbf{x}})$ and

$$\mathbf{f}(\mathbf{x}, \mathbf{g}(\mathbf{x})) = \mathbf{0},$$

is satisfied for all $\mathbf{x} \in N$. Furthermore, $\mathbf{g}$ is L-smooth at $\hat{\mathbf{x}}$; for every $p \in \mathbb{N}$ and $\mathbf{M} \in \mathbb{R}^{m \times p}$, $\mathbf{g}'(\hat{\mathbf{x}}; \mathbf{M})$ is the unique solution to the equation system

$$\mathbf{f}'(\hat{\mathbf{x}}, \hat{\mathbf{y}};\ (\mathbf{M}, \mathbf{g}'(\hat{\mathbf{x}}; \mathbf{M}))) = \mathbf{0}.$$

The last equation system in Theorem 4.2 is found by applying the chain rule in (4.13). Solving it with respect to $\mathbf{g}'(\hat{\mathbf{x}}; \mathbf{M})$ will yield the LD-derivatives for the sensitivities. In the case that $\mathbf{f}$ is $\mathcal{C}^1$ at $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, the projection

$$\pi_2 \partial \mathbf{f}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = \left\{ \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \right\}. \tag{4.14}$$

Consequently, the assumption regarding $\pi_2 \partial \mathbf{f}(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ containing no singular matrices is analogous to requiring the partial Jacobian to be nonsingular in the $\mathcal{C}^1$ case.

4.6 Forward automatic differentiation for evaluating derivatives

Automatic differentiation (AD) is a method for computing derivatives numerically by exploiting the procedural operations that take place in a computer when evaluating functions. A sequence of elemental operations are executed, each returning a temporary variable that is then used in other operations. Elemental operations in this context refer to simple operations like addition, subtraction, multiplication, division as well as simple functions (sine, exponential function, max/min function etc.). By virtue of the chain rule, such stepwise computation is also possible for differentiation, and may be done in parallel to finding the function value. This is the foundation of the forward mode of automatic differentiation, for which derivative values are carried forward along with the numerical values themselves [15]. There also exists a reverse mode, though it will not be discussed in this report. For more information on the reverse mode, check [15]. The principles of automatic differentiation is best illustrated with an example.

Example 4.8. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a $\mathcal{C}^1$ function such that*

$$f(x_1, x_2) = x_1 x_2 + x_1^2 + e^{x_2}. \tag{4.15}$$

Assume that a program evaluates this function at the point $(x_1, x_2) = (0.5, 1.0)$, then a possible sequence of elemental operations would be as in Table 4.1.

Table 4.1: Illustration of the evaluation sequence in a program.

u_{-1}	$=$	x_1	$=$	0.5000		
u_0	$=$	x_2	$=$	1.0000		
u_1	$=$	$u_{-1} \cdot u_0$	$=$	$0.5000 \cdot 1.0000$	$=$	0.5000
u_2	$=$	$u_{-1} \cdot u_{-1}$	$=$	$0.5000 \cdot 0.5000$	$=$	0.2500
u_3	$=$	$\exp(u_0)$	$=$	$\exp(1)$	$=$	2.7183
u_4	$=$	$u_1 + u_2$	$=$	$0.5000 + 0.2500$	$=$	0.7500
u_5	$=$	$u_4 + u_3$	$=$	$0.7500 + 2.7183$	$=$	3.4683
y	$=$	u_5	$=$	3.4683		

Forward automatic differentiation calculate the derivatives of a temporary variable simultaneously, using the chain rule and the same sequence of elemental operations (see Table 4.2).

Table 4.2: Forward derivative calculations following the same computation sequence.

$\mathbf{J}u_{-1}$	$=$	$\begin{bmatrix}\frac{\partial u_{-1}}{\partial x_1} & \frac{\partial u_{-1}}{\partial x_2}\end{bmatrix}$	$=$	$[1 \quad 0]$				
$\mathbf{J}u_0$	$=$	$\begin{bmatrix}\frac{\partial u_0}{\partial x_1} & \frac{\partial u_0}{\partial x_2}\end{bmatrix}$	$=$	$[0 \quad 1]$				
$\mathbf{J}u_1$	$=$	$\mathbf{J}\left(u_{-1} \cdot u_0\right)$	$=$	$u_0 \cdot \mathbf{J}u_{-1} + u_{-1} \cdot \mathbf{J}u_0$	$=$	$1 \cdot [1 \quad 0] + 0.5 \cdot [0 \quad 1]$	$=$	$[1 \quad 0.5]$
$\mathbf{J}u_2$	$=$	$\mathbf{J}\left(u_{-1} \cdot u_{-1}\right)$	$=$	$2 \cdot u_{-1} \cdot \mathbf{J}u_{-1}$	$=$	$2 \cdot 0.5 \cdot [1 \quad 0]$	$=$	$[1 \quad 0]$
$\mathbf{J}u_3$	$=$	$\mathbf{J}\left(\exp\left(u_0\right)\right)$	$=$	$\exp\left(u_0\right) \cdot \mathbf{J}u_0$	$=$	$\exp\left(1\right)[0 \quad 1]$	$=$	$[0 \quad 2.72]$
$\mathbf{J}u_4$	$=$	$\mathbf{J}\left(u_1 + u_2\right)$	$=$	$\mathbf{J}u_1 + \mathbf{J}u_2$	$=$	$[1 \quad 0.5] + [1 \quad 0]$	$=$	$[2 \quad 0.5]$
$\mathbf{J}u_5$	$=$	$\mathbf{J}\left(u_4 + u_3\right)$	$=$	$\mathbf{J}u_4 + \mathbf{J}u_3$	$=$	$[2 \quad 0.5] + [0 \quad 2.72]$	$=$	$[2 \quad 3.22]$
$\mathbf{J}f$	$=$	$\mathbf{J}u_5$	$=$	$[2 \quad 3.22]$				

Although Example 4.8 was done for a $\mathcal{C}^1$ function, the same principles apply to nonsmooth functions, provided they are L-factorable [23].

Definition 4.11. *(From [23]). A function* $\mathbf{f}$ *is L-factorable if all the functions in its elemental library are L-smooth and their LD-derivatives are known or computable.*

Among L-factorable functions are the $\mathcal{PC}^1$ functions from the MHEX model [23]. LD-derivatives replace conventional derivatives for the nonsmooth case, and are included in an automatic differentiation framework through object-oriented programming. Specifically, by defining a new data type that also stores the variable's derivatives. In addition, operators and simple functions are overloaded, i.e. customized for a particular data type, to calculate LD-derivatives simultaneously using calculus rules. The procedures in Tables 4.1 and 4.2 thus do not occur separately but are intertwined in the model.

Algorithm 2 presents a general layout of the forward mode of automatic differentiation for an L-factorable function $\mathbf{f}$ [23].

Algorithm 2 Forward mode of automatic differentiation for an L-factorable function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$

```
1: Assign $u_{k-n} \leftarrow x_k$ for $k = 1, \dots, n$
2: Assign $\mathbf{U}_{k-n} \leftarrow \left[\mathbf{M}^{\mathrm{T}}\right]_k$ for $k = 1, \dots, n$
3: for $i = 1, \dots, l$ do
4:     Set $\mathbf{v}_i \leftarrow [u_j]_{j \prec i}$
5:     Calculate $u_i = \gamma_i(\mathbf{v}_i)$
6:     Set $\mathbf{V}_i \leftarrow [\mathbf{U}_j]_{j \prec i}$
7:     Calculate $\mathbf{U}_i = [\gamma_i]'(\mathbf{v}_i; \mathbf{V}_i)$
8: end for
9: return $\mathbf{f}(\mathbf{x}) = \mathbf{u}_l$ and $\mathbf{f}'(\mathbf{x}; \mathbf{M}) = \mathbf{U}_l$
```

Here $\mathbf{M} \in \mathbb{R}^{n \times p}$ is the directions matrix, while $\gamma_i, \forall i = 1, \dots, l$ are the l elemental operations that occur in the function. Furthermore, $\left[\mathbf{M}^{\mathrm{T}}\right]_k$ is the k^{th} row of $\mathbf{M}$. The notation $j \prec i$ is used to indicate direct dependency (i.e. $j \prec i$ is true if u_i is defined by a function $\mathbf{f}_{u_i}(\dots, u_j)$), whereas $[u_j]_{j \prec i}$ and $[\mathbf{U}_j]_{j \prec i}$ are stacked scalars and vectors, respectively, of variables (u_j) and derivatives $(\mathbf{U}_j)$ that occur in the i^{th} elemental operation. Example 4.9 shows the general outline of Algorithm 2.

Example 4.9. *Consider the L-factorable function*

$$f : \mathbb{R}^2 \to \mathbb{R} : (x_1, x_2) \to max\left\{x_1^2, x_2\right\}$$

at a point of nondifferentiability $(x_1, x_2) = (2, 4)$*, and a directions matrix*

$$\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Then f consists of $l = 2$ elemental operations, which are multiplication ($x_1 \cdot x_1$) and a max-function. The steps taken by Algorithm 2 are summarized in Table 4.3. The LD-derivative $\mathbf{U}_2$ for the max-function was found using the results from Example 4.5 and 4.6 for a directions matrix

$$\mathbf{V}_2 = \begin{bmatrix} 0 & 1 \\ 4 & 0 \end{bmatrix}.$$

$$\begin{aligned} \mathbf{U}_2 &= max' \left(\mathbf{v}_2; \mathbf{V}_2\right), \\ &= max' \left(\begin{bmatrix} 4 \\ 4 \end{bmatrix}; \begin{bmatrix} 0 & 1 \\ 4 & 0 \end{bmatrix} \right), \\ &= \begin{bmatrix} 4 & 0 \end{bmatrix}, \\ &= \mathbf{U}_1. \end{aligned}$$

Table 4.3: Numerical calculation of the LD-derivative of max $\left\{x_1^2, x_2\right\}$ at $(x_1, x_2) = (2, 4)$, using the forward mode of automatic differentiation for L-factorable functions.

u_{-1}	$=$	x_1	$=$	2	$\mathbf{U}_{-1}$	$=$	$\left[\mathbf{M}^T\right]_1$	$=$	$[1 \quad 0]$
u_0	$=$	x_2	$=$	4	$\mathbf{U}_0$	$=$	$\left[\mathbf{M}^T\right]_2$	$=$	$[0 \quad 1]$
$i = 1$:									
$\mathbf{v}_1$	$=$	$[u_{-1}]$			$\mathbf{V}_1$	$=$	$[\mathbf{U}_{-1}]$		
u_1	$=$	$u_{-1} \cdot u_{-1}$	$=$	4	$\mathbf{U}_1$	$=$	$2u_{-1} \cdot \mathbf{U}_{-1}$	$=$	$[4 \quad 0]$
$i = 2$:									
$\mathbf{v}_2$	$=$	$\begin{bmatrix} u_0 \\ u_1 \end{bmatrix}$			$\mathbf{V}_2$	$=$	$\begin{bmatrix} \mathbf{U}_0 \\ \mathbf{U}_1 \end{bmatrix}$		
u_2	$=$	$\max\{u_1, u_0\}$	$=$	4	$\mathbf{U}_2$	$=$	$\mathbf{U}_1$	$=$	$[4 \quad 0]$

Algorithm 4.9 therefore returns:

$$f(2, 4) = 4,$$

and

$$f'\left((2, 4); \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = \begin{bmatrix} 4 & 0 \end{bmatrix}.$$

Chapter 5

Implementation of a nonsmooth implicit function theorem

Section 3.3 presented an approach for evaluating analytical sensitivities of modular subroutines by invoking the classical implicit function theorem. For Theorem 3.1 to be satisfied, however, the functions need to be continuously differentiable, which is not the case for the MHEX model. Instead, the concept of LD-derivatives was studied in Chapter 4, along with an implicit function theorem for lexicographically smooth functions. The theorem states that there exists a unique solution $\mathbf{g}'(\hat{\mathbf{x}};\mathbf{M})$ to the equation system

$$\mathbf{f}'(\hat{\mathbf{x}},\hat{\mathbf{y}};\ (\mathbf{M},\mathbf{g}'(\hat{\mathbf{x}};\mathbf{M}))) = \mathbf{0}, \tag{5.1}$$

where $\mathbf{g}'(\hat{\mathbf{x}};\mathbf{M})$ is the LD-derivative of the implicit function. Like the result for $\mathcal{C}^1$ functions, it acts as an input-output sensitivity for $\mathcal{PC}^1$ subroutines. This chapter begins with presenting a lemma for solving (5.1), before implementing the procedure in C++ in Section 5.2. Finally, the algorithm is tested for the $\mathcal{PC}^1$ pQ-flash formulation in Equation (2.16).

5.1 Solving for the input-output sensitivities of $\mathcal{PC}^1$ subroutines

Equation (5.1) is solved by applying a lemma produced by Khan (Khan, Argonne National Laboratory, personal communication, 2015):

Lemma 5.1. *Let conditions of Theorem 4.2 hold. Then the k^{th} column $\mathbf{n}_k$ of $\mathbf{N} \equiv \mathbf{g}'(\mathbf{x}; \mathbf{M})$ is the unique solution to the set of equations:*

$$\mathbf{0} = \mathbf{f}^{(k-1)}_{\begin{bmatrix}\hat{\mathbf{x}}\\ \hat{\mathbf{y}}\end{bmatrix}, \begin{bmatrix}\mathbf{M}_{(k-1)}\\ \mathbf{N}_{(k-1)}\end{bmatrix}} \left(\begin{bmatrix}\mathbf{m}_k\\ \mathbf{n}\end{bmatrix}\right).$$

The residual function for this equation system, defined as

$$\mathbf{h} : \mathbf{v} \to \mathbf{f}^{(k-1)}_{\begin{bmatrix}\hat{\mathbf{x}}\\ \hat{\mathbf{y}}\end{bmatrix}, \begin{bmatrix}\mathbf{M}_{(k-1)}\\ \mathbf{N}_{(k-1)}\end{bmatrix}} \left(\begin{bmatrix}\mathbf{m}_k\\ \mathbf{v}\end{bmatrix}\right),$$

is L-smooth on $\mathbb{R}^n$; for any $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{n \times q}$, $q \in \mathbb{N}$

$$\mathbf{h}'(\mathbf{v}; \mathbf{A}) = \mathbf{f}'\left(\begin{bmatrix}\hat{\mathbf{x}}\\ \hat{\mathbf{y}}\end{bmatrix}; \begin{bmatrix}\mathbf{M}_{(k-1)} & \mathbf{m}_k & \mathbf{0}_{m\times q}\\ \mathbf{N}_{(k-1)} & \mathbf{v} & \mathbf{A}\end{bmatrix}\right) \begin{bmatrix}\mathbf{0}_{k\times q}\\ \mathbf{I}_{q\times q}\end{bmatrix}.$$

Lemma 5.1 allows solving for each column of $\mathbf{g}'(\mathbf{x}; \mathbf{M})$ sequentially using a nonsmooth Newton-type solver. At each step, $\mathbf{n}_k$ is found as the k^{th} order directional derivative of $\mathbf{f}$ in direction $[\mathbf{m}_k \quad \mathbf{n}_k]^{\mathrm{T}}$, approaches zero. Previously evaluated LD-derivative elements $\mathbf{N}_{(k-1)}$ of the implicit function, as well as the directions matrix $\mathbf{M}_k$ up to column k are used in the calculations. The search direction for the Newton-type solver is provided by the LD-derivative $\mathbf{h}'(\mathbf{v}; \mathbf{A})$ of the residual function $\mathbf{h}$, where $\mathbf{A}$ is a directions matrix. Algorithm 3 presents an outline for solving (5.1). The variable $N_{\text{var, o}}$ is here used to denote the total number of variables in the "outer problem", i.e. excluding the variables incorporated into subroutines.

Algorithm 3 Computing sensitivities for lexicographically smooth functions.

```
    Given input variables $\hat{\mathbf{x}}$, the solution of the output variables $\hat{\mathbf{y}}$ and directions
    matrices $\mathbf{A}$ and $\mathbf{M}$.
 1: Set $k = 1$.
 2: while $k \leq N_{\text{var, o}}$ do
 3:     if $k = 1$ then
 4:         Set $\mathbf{M}_1 \leftarrow \mathbf{m}_1$.
 5:     else
 6:         Set $\mathbf{M}_k \leftarrow \left[\mathbf{M}_{(k-1)} \quad \mathbf{m}_k\right]$.
 7:     end if
 8:     Set $\mathbf{v} \leftarrow \mathbf{0}$ as initial guess
 9:     Do a Newton solve with $\mathbf{H} \leftarrow \mathbf{h}'\left(\mathbf{v}^j; \mathbf{A}\right)$ as a generalized derivative element
    to iterate $\mathbf{v}^{(j+1)} = \mathbf{v}^j - \mathbf{H}^{-1}\mathbf{h}\left(\mathbf{v}^j\right)$, $j = 1, \ldots$, where $\mathbf{v}^0 = 0$.
10:     Set $\mathbf{N} \leftarrow [\mathbf{N} \quad \mathbf{v}^*]$.
11:     $k = k + 1$.
12: end while
13: return $\mathbf{N}$
```

5.2 Implementation in C++

Algorithm 3 was implemented in C++ using the existing automatic differentiation framework developed by Khan and Barton [23] for calculating LD-derivatives, included in the multistream heat-exchanger model. The residual functions are solved with the nonsmooth Newton-type method in (2.18) by Qi and Sun [32] using a generalized derivative element $\mathbf{G}\left(\mathbf{x}^k\right) \in \text{plen}\left(\partial\mathbf{f}\left(\mathbf{x}^k\right)\right)$ at each iteration.

```

%Sensitivity function. Calculates sensitivities for any function.

%[x] are the output variables from the subroutine.
%[nVar] is the number of variables in the "outer" problem
%[vari] are all the variables in the outer problem.
%[p] are all the scalar parameters needed in the subroutine.
%[p_vect] are all the vector parameters needed in the subroutine.
%[PQflash_sens] is a pointer to the residual functions.

void sensitivity(vector<ldouble> &x, vector<ldouble> vari, vector<ldouble>
     p, vector< vector<ldouble> > p_vect, LDfunc Resid){

    % Find the dimensions of the matrix A:
    int colA=x.size();% Number of inner problem variables determines the
        dimensions of A.
    int nVar=vari[0].getDepth(); % Number of variables in the outer
        problem.

    for(int i=0;i<colA;i++){
        x[i].setDepth(nVar);% Setting the depth of the variables in the inner problem such that the sensitivities can be stored.
    }

    % Setting up the vector for the parameters in the inner sensitivity function.
    vector<ldouble> var(x.size()+vari.size());% [var] contains all the variables in the outer problem + all the variables in the inner problem

    for(int i=0;i<var.size();i++){
        if(i<vari.size()){
            var[i]=vari[i].getValue();
        }
        else{
            var[i]=x[i-vari.size()].getValue();% adds all the inner problem variables
        }
    }


    % Calculates each of the unknown higher order directional derivatives [v] at each iteration.
    for(int i=0;i<nVar;i++){
        %setting the [v]-vector
        dVector v(colA);%[colA] represent the number of variables in the inner problem!
        for(int j=0;j<colA;j++){
            v[j]=0.0;%provides initial guesses for all the [v] elements
        }



        %Setting depths for all parameters:
        for(int j=0;j<var.size();j++){
            var[j].setDepth(i+colA+1);%[colA] here accounts for the zero matrix above A in Lemma 3.1.
        }

        %Copying the i first columns of the M matrix from the input variables:
        for(int j=0; j<vari.size();j++){
                for(int k=0;k<i+1;k++){
                    var[j].setDotValue(k,vari[j].getDotValue(k));
                }
        }

        %adding the A matrix to the remaining directional derivative columns of the output variables:
        for(int j=0;j<colA;j++){
                var[j+vari.size()].setDotValue(i+1+j,1.0);
        }


        %Adding the [i−1] columns of N calculated in previous iterations.
            This will only happen after the first iteration.
        if(i>0){
            for(int j=0;j<colA;j++){
                var[j+vari.size()].setDotValue(i−1,x[j].getDotValue(i−1));
            }
        }


        vector<vector<ldouble>> vect_params(2+p_vect.size());
        %This vector is initialized in order to collectively organise all
            the parameters to the module as one argument. % The first
            column in [vect_params] consists of the outer problem
            variables [vari].
        % The scalar parameters [p] to the module will make up the second
            column in the [vect_params] argument.
        % The nonscalar function parameters [p_vect] are placed in column
            three and onwards.


        vector<ldouble> inVar(colA);% All the output variables ("inner
            problem variables") of the module for which sensitivities will
             be found for.
        % All the inner problem variables are provided as a separate
            argument.

        vector<ldouble> OuterVar(vari.size());%All the outer problem
            variables plus LD−derivative information up until column i.

        for(int j=vari.size();j<vari.size()+colA;j++){
            inVar[j−vari.size()]=var[j];
        }
        for(int j=0;j<vari.size();j++){
            OuterVar[j]=var[j];
        }

        vect_params[0]=OuterVar;% Adding all the "outer model variables"
            to the first row of [vect_params].
        vect_params[1]=p;% Adding all the scalar module parameters to the
            second row of [vect_params].
        for(int j=0;j<p_vect.size();j++){
            vect_params[2+j]=p_vect[j];% Adding the remaining vector
                parameters.
        }



        bool flag;
        double S;% Used to temporary store the sensitivities
        flag=ssNewton_solve(colA, 1.0, v, inVar, vect_params, Resid);
        if(flag){
            for(int j=0;j<colA;j++){% Cycles through all the variables and
```

```
                     add the sensitivities corresponding to a given variable
                    i.
101                 S=v[j]; % Sensitivities
102                 x[j].setDotValue(i,S);
103             }
104         }
105     }
106 }
```

Input variables to the function are of a predefined data type ldouble that consists of a value of type double, which is a data type that is used for storing floating point numbers, as well as a vector containing the LD-derivative. Each element of this vector corresponds to an element in Equation (4.11), and is throughout the code referred to as a dot-value. The function setDepth at line 46 manages the dimension of the dot-values vector, which will increase by one for every iteration i. This additional space is needed to contain the vectors $\mathbf{v}$ and $\mathbf{m}_i$ from Lemma 5.1. The function getDepth returns the current dimension of the LD-derivative vector. Also an argument in the sensitivity function, is a pointer Resid to the residuals required by the nonsmooth Newton-type solver at line 98. Pointers to functions are used in order to make the code more versatile. The residual function used for testing the above algorithm is found in Appendix A.

The directions matrix $\mathbf{M}_{(i-1)}$ is added in the for-loop between lines 50 and 54. It consists of the LD-derivatives for the input variables, and will have a row dimension equal to the total number of variables in the outer problem. As a consequence of the chain rule, this derivative information is then transferred to the output variables.

The matrix $\mathbf{A}$ must be chosen to be square and nonsingular so that (4.12) holds. In this program, the identity matrix $\mathbf{I}$ was used. An initial guess for $\mathbf{v}$ is given at line 39 by setting all its elements to zero. Sensitivities are stored in the output variables $\mathbf{x}$ at line 98, after solving the residual functions. The $\mathbf{x}$ are passed to the function as address arguments, and are also used for updating the $\mathbf{N}_{(i-1)}$ columns at each iteration (lines 63-67).

5.3 Testing the sensitivity function for the nonsmooth VLE equation

A pQ-flash test problem with ideal properties was used for checking the implementation of the sensitivity function. Watson and Barton developed a nonsmooth model for flash calculations that considers the disappearance of phases at the boundary of the two-phase region [41]. The model is formulated using a mid function with the three possible outlet conditions: all liquid outlet, all vapour outlet and two-phase outlet as arguments.

$$\operatorname{mid}\left\{\alpha, \alpha-1, -\sum_{i=1}^{n_{\mathrm{C}}} \frac{z_i\left(K_i-1\right)}{1+\alpha\left(K_i-1\right)}\right\}=0. \tag{5.2}$$

The mid function is $\mathcal{PC}^1$ everywhere on its domain, meaning it is lexicographically smooth. Consequently, its LD-derivative is well-defined and the implicit function theorem (5.1) is applicable. Accompanying the mid function for a pQ-flash is the overall energy balance

$$h_{\mathrm{l}} L + h_{\mathrm{v}} V = h_{\mathrm{f}} F + Q, \tag{5.3}$$

which is also L-smooth.

Example 5.1. *Consider a pQ-flash at $p = 3.695 MPa$ with a feed composition as given in Table 5.1. The residual function used for this test problem can be found in Appendix A.*

Table 5.1: Feed compositions used for the pQ-flash test problem.

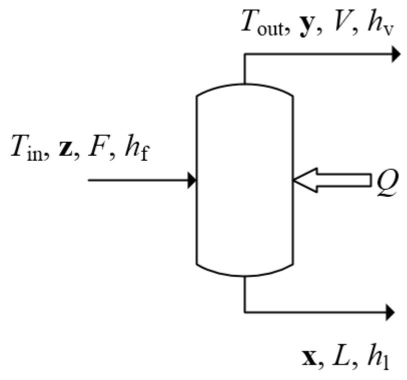
Component	**Mole fraction**
N_2	0.1532
CH_4	0.1779
C_2H_6	0.4086
C_3H_8	0.0041
C_4H_{10}	0.2562
Sum	1.0000

Inlet temperature and feed flowrate was set to $T_{in} = 240K$ and $F = 1.0 kmol/s$, respectively. The corresponding inlet enthalpy was found in Aspen Plus® to be $h_f = -93978.6 J/mol$ relative to the elements' standard state at 298.15K [3, p.16-17]. Heat influx to the flash vessel was specified to $1000 kW$.

pQ-flash calculations solve Equations (2.9)-(2.13) for the liquid- and vapour outlet compositions ($\mathbf{x}, \mathbf{y}$), the flash temperature (T_{out}), and the the vapour fraction α. The pQ-flash problem was simulated using both Aspen Plus® and Equations (5.2)-(5.3) to verify results before computing the sensitivities.

Table 5.2 shows some deviations between the nonsmooth model and Aspen Plus®. It is likely that these departures can be accredited to different methods for evaluating specific heat capacities at low temperatures. Aspen Plus® extrapolates many of its temperature dependent property models linearly for temperatures beyond the limits [4, p. 9]. The ideal thermodynamics implemented for the multistream heat exchanger model, on the other hand, simply extend these models at their endpoints.

Using the program outlined in Section 5.2, sensitivities for T_{out} and α with respect to the three pQ-flash parameters h_f, Q and p were found (see Table 5.3).



T_{out}, $\mathbf{y}$, V, h_{v}
T_{in}, $\mathbf{z}$, F, h_{f}
Q
$\mathbf{x}$, L, h_{l}

Figure 5.1: The single stage pQ-flash test problem.

Table 5.2: pQ-flash simulation results.

Flash model by Watson and Barton	
T_{out} [K]	α
248.03	0.34
Solution in Aspen Plus®	
T_{out} [K]	α
248.69	0.31

Both T_{out} and α show a low dependence on the heat influx Q and molar feed enthalpy h_f. Moreover, with a feed flowrate of $1.0 kmol/s$ they will be equally sensitive to these parameters. At this flowrate, a unit perturbation in h_f is equal to one kW and therefore of the same magnitude as a change in Q. The temperature increased with $8.7K$ in the vessel with $Q = 1000kW$, which constitutes $8.7 \cdot 10^{-3}$ increase. Consequently, a temperature dependence of $8.31 \cdot 10^{-3}$ on Q and h_f is reasonable.

Table 5.3 predicts a negative correlation between the vapour fraction and pressure. Higher pressures shift the boiling point temperatures upwards, condensation occurs and the vapour fraction decreases. The temperature–pressure relationship is more easily studied for pure compounds.

Table 5.3: Sensitivities for the output variables T_{out} and α.

	Q[kW]	h_{f}[kJ/kmol]	p[MPa]
T_{out}[K]	$8.31 \cdot 10^{-3}$	$8.31 \cdot 10^{-3}$	2.33
α	$3.52 \cdot 10^{-5}$	$3.52 \cdot 10^{-5}$	-0.03

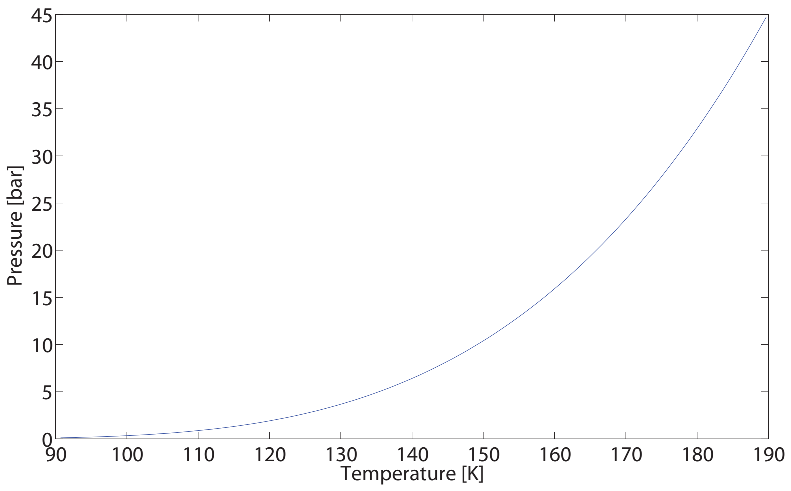


Figure 5.2: Boiling point curve of methane plotted from the extended Antoine equation [4].

Figure 5.2 reveals a gradient

$$\frac{dp}{dT} > 0, \; \forall T \in \left[T_{TP}, T_c\right], \tag{5.4}$$

where TP and c are used to denote the triple point and the critical point, respectively. A positive correlation between pressure and outlet temperature is therefore expected.

Chapter 6

Modularizing the vapour-liquid equilibrium calculations

This chapter presents a hybrid framework for the multistream heat exchanger model where the vapour-liquid equilibrium equations are modularized and solved sequentially. The goal is to find out whether the alternative formulation leads to a more robust model, capable of simulating complex liquefaction processes. First, Section 6.1 covers the general structure of the VLE subroutine, as well as presenting a modified initialization procedure. In Section 6.2, the VLE module is tested and verified using the equation-oriented PRICO model as reference. Section 6.3 then tests the robustness of the hybrid framework for the single mixed refrigerant processes studied in the specialization project [40]. The final section looks at the increase in simulation times for the alternative model. Ideal properties were used in all the simulations.

6.1 The vapour-liquid equilibrium module

With ideal properties, the pQ-flash problem is a two variables problem in T_{out} and V, hence the energy balance

$$h_{\text{l}}L + h_{\text{v}}V = h_{\text{f}}F + Q \tag{6.1}$$

is needed in addition to Equation (2.16). Arguments to the module are the pressure p, the feed enthalpy h_{f}, the feed composition $\mathbf{z}$, the feed flowrate F and the number of flash stages $n_{2\mathrm{p}}$. The flash stages are solved sequentially using a nonsmooth Newton-type solver, and correct LD-derivatives for the output variables are evaluated using the sensitivity function from Chapter 5. Before the next stage, the feed enthalpy is either reduced or increased by Q, depending on whether the stream is hot or cold. As mentioned in Section 2.3, Q is chosen such that heating/cooling in the pQ-flashes sum up to the total enthalpy difference in the two-phase region. The VLE-module uses an inbuilt function FlashStage for solving the flash equations, such that it becomes compatible with different thermodynamic models. Algorithm 4 presents the general outline of the module in pseudo-code.

Algorithm 4 The vapour-liquid equilibrium module

```
1: function VLE-MODULE($p$, $h_\mathrm{f}$, $\mathbf{z}$, $F$, $n_{2\mathrm{p}}$, $Q$)
2:     Sort the function arguments into input variables $\mathbf{v}_0$ and input parameters $\mathbf{p}$.
3:     Initialize the flash variables $T_{\mathrm{out},i}$ and $\alpha_i$ for each stage $i$.
4:     Set the iteration counter: $i \leftarrow 0$.
5:     while $i < n_{2\mathrm{p}}$ do
6:         Evaluate $(T_{\mathrm{out},i}, \alpha_i) \leftarrow$FLASHSTAGE($\mathbf{v}_i$, $\mathbf{p}$)
7:         Evaluate $\mathbf{g}_i'(T_{\mathrm{out},i}, \alpha_i; \mathbf{M}) \leftarrow$SENSITIVITY($T_{\mathrm{out},i}$, $\alpha_i$, $\mathbf{v}_i$, $\mathbf{p}$, FLASHSTAGE($\mathbf{v}_i$, $\mathbf{p}$))
8:         Set $H_\mathrm{f} \leftarrow H_\mathrm{f} + Q$ and $i \leftarrow i + 1$.
9:     end while
10: end function
```

A nonsmooth Newton-type solver is used in FlashStage for solving Equations (2.16) and (6.1), which requires good initial guesses for $T_{\mathrm{out},i}$ and α_i in order to converge. The VLE-module applies the same initialization procedure used in the original MHEX model, where temperatures and vapour fractions are modelled by linear interpolation:

$$T_{\mathrm{out},i} = T_{2\mathrm{p,\,in}} + \frac{i}{n_{2\mathrm{p}}}\left(T_{2\mathrm{p,\,out}} - T_{2\mathrm{p,\,in}}\right), \tag{6.2}$$

$$\alpha_i = \alpha_{2\mathrm{p,\,in}} + \frac{i}{n_{2\mathrm{p}}}\left(\alpha_{2\mathrm{p,\,out}} - \alpha_{2\mathrm{p,\,in}}\right). \tag{6.3}$$

The flash equations are then solved for the assumed values of $T_{\mathrm{out},i}$ and α improving the estimates.

6.2 Verifying the vapour-liquid equilibrium module

The VLE module needs to be tested and verified before comparing the two frameworks. This was done by simulating the PRICO process, which consists of a single MHEX and is thus regarded as the simplest of the single mixed refrigerant processes. With $n_{2\text{p}} = 5$, the equation-oriented model is comprised of 52 variables,

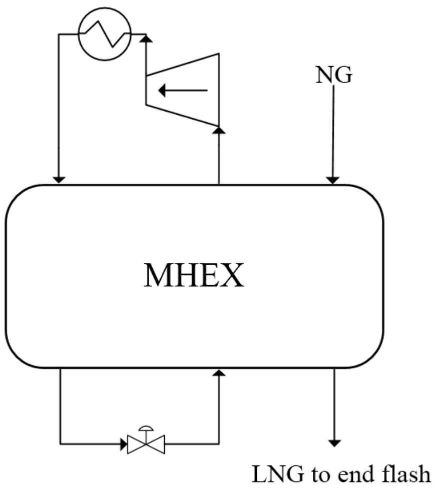


Figure 6.1: The PRICO process.

whereas with the hybrid framework the number is reduced to 28. Equations (2.4), (2.5), and (2.6) means the MHEX model can solve for three unknown quantities (v_1, v_2 and v_3). The remaining variables account for the dew-point and bubble-point temperatures, the temperatures in the superheated and subcooled regions, and in the case of no VLE subroutines, the temperatures and vapour fractions in the two-phase region. Temperatures in the single phase regions are found by energy balances over the affine segments i used to approximate the composite curves (see Section 2.2) [41]:

$$F\left(h^{\text{in}}_{\text{sup/sub},i} - h^{\text{out}}_{\text{sup/sub},i}\right) = \frac{Q_{\text{sup/sub}}}{n_{\text{sup/sub}}}, \tag{6.4}$$

where F is the molar flowrate of the stream, $h^{\text{in/out}}_{\text{sup/sub},i}$ are the molar enthalpies in/out of segment i, $n_{\text{sup/sub}}$ is the number of affine segments used to approximate the composite curves, and $Q_{\text{sup/sub}}$ is the total heat transferred in the phase region.

To test the VLE modules, two simulation cases were developed with different properties selected as unknown variables. Case I solves for the high pressure (HP) and the low pressure (LP) refrigerant temperature out of the multistream heat exchanger, as well as for the heat transfer area UA. Case II, on the other hand, solves for the low pressure level, the low pressure refrigerant temperature out, and ΔT_{min}. Both simulation cases are detailed in Table 6.1. The LP inlet temperature v_4 is also an unknown quantity in both cases, but is instead determined by the throttling valve (see Figure 6.1).

Table 6.1: Refrigerant stream and MHEX data used in simulations of the PRICO process.

Property	**Case I**	**Case II**
High pressure level [MPa]	3.695	3.695
Low pressure level [MPa]	0.170	v_1
Flowrate [kmol/s]	3.47	3.47
HP inlet temperature [K]	303.15	298.15
HP outlet temperature [K]	v_1	118.5
LP inlet temperature [K]	v_4	v_4
LP outlet temperature [K]	v_2	v_2
Composition [mol %]		
N_2	15.32	15.32
CH_4	17.79	17.79
C_2H_6	40.85	40.85
C_3H_8	0.41	0.41
n-C_4H_{10}	25.62	25.62
MHEX data		
UA [MW/K]	v_3	10
ΔT_{min} [K]	1.2	v_3

Case I was simulated in the two frameworks with the initial conditions given in Table 6.2. The table shows both models converged to the same solution, corresponding to an overall heat duty of 86 MW. Furthermore, according to Figure 6.2, the two models have identical driving force distributions in the MHEX. The ΔT–H plot has the characteristic shape of cryogenic processes, with smaller temperature differences at lower temperatures where thermodynamic losses become more significant [40]. Therefore, it is reasonable to conclude the VLE subroutines are functioning properly, and that the two frameworks are equivalent. However, before asserting the validity of the hybrid model, a similar test was also conducted for Case II.

In the second case, the initial condition in Table 6.3 was used, which again resulted in both models converging to the same solution. With both simulations returning identical results, the VLE-modules are seemingly working properly, such that the hybrid MHEX model can be used for simulating the more complex processes.

Table 6.2: Numerical results of Case I for the PRICO process.

Variable	**Initial values**	**Results**	
		EO model	**Hybrid model**
v_1 [K]	130.00	121.16	121.16
v_2 [K]	250.00	286.55	286.55
v_3 [MW/K]	10.00	18.70	18.70
v_4 [K]	116.95	116.95	116.95
Process:			
Total duty [MW]		85.97	85.97

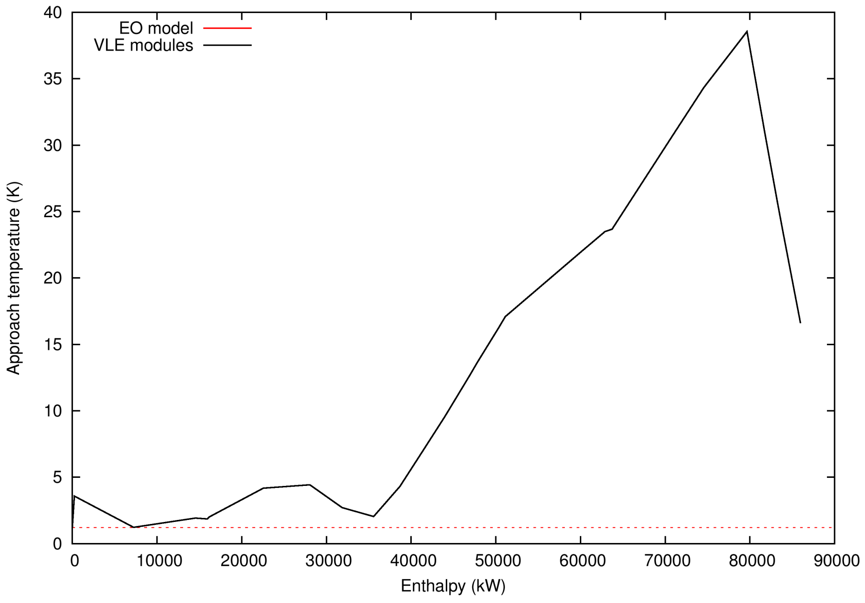


Figure 6.2: Case I driving force plots for the equation-oriented and the hybrid PRICO model.

Table 6.3: Numerical results of Case II for the PRICO process.

Variable	**Initial values**	**Results**	
		EO model	**Hybrid model**
v_1 [MPa]	0.140	0.120	0.120
v_2 [K]	220.00	271.38	271.38
v_3 [K]	1.50	2.80	2.80
v_4 [K]	116.65	114.39	114.39
Process:			
Total duty [MW]		83.46	83.46

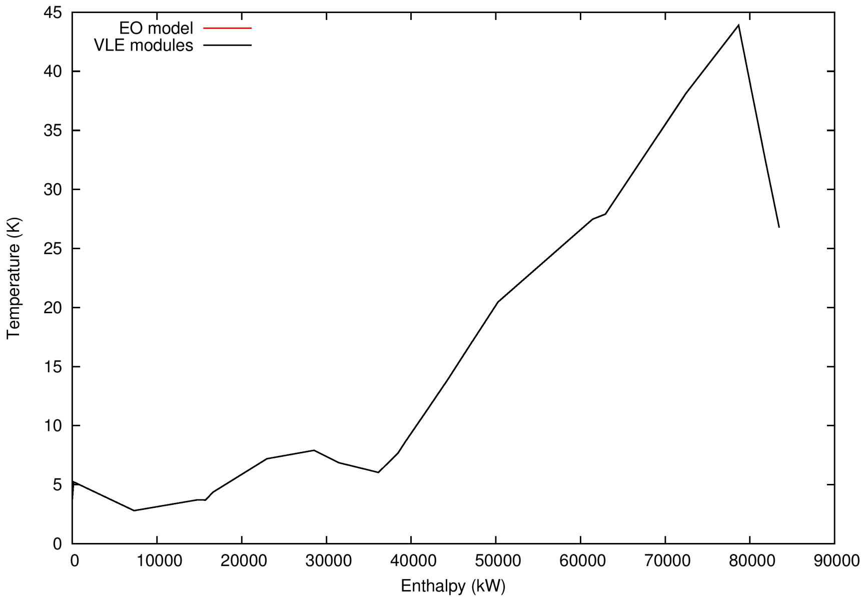


Figure 6.3: Case II driving force plots for the equation-oriented and the hybrid PRICO model.

6.3 Performance testing of the hybrid framework

The MHEX model with modularized flash calculations is tested for three different processes taken from the specialization project [40]: the PRICO process, the "extended" PRICO process, and the single mixed refrigerant process with integrated NGL extraction. Performance tests are done with respect to efficiency and robustness. Ideal properties were used for all the simulations, with the number of discrete flash stages in the two phase region of each MHEX set to 5. Simulations of the two PRICO models were done using the natural gas composition from Table 6.4. A "richer" composition was used for the single mixed refrigerant process with NGL extraction, in order to have separation at 240K (see Table 6.11). The focus of the simulations were on the cooling and heating curves in the MHEXs. No attention was therefore put on the compressor and aftercooler part of the models.

Table 6.4: Natural gas stream data used in the simulations of the PRICO and the extended PRICO process.

Property	**Value**
Pressure [MPa]	5.50
Flowrate [kmol/s]	1.00
Inlet temperature [K]	298.15
Outlet temperature [K]	118.15
Composition [mol %]	
N_2	1.00
CH_4	95.60
C_2H_6	3.10
C_3H_8	0.20
n-C_4H_{10}	0.10

In order to test robustness, the models were run over a range of inputs, registering when they converged. Properties like number of iterations and time per iteration were noted, such that the efficiency of the two frameworks could be analysed. Rather than entering the initial guesses manually for each simulation, a bash-script was constructed that calls the model N times using for-loops.

6.3.1 The PRICO model

The first model to be analysed is the PRICO process, which was also used for verifying the VLE modules in Section 6.2. The same cases are considered here, with intervals for the variables presented in Table 6.5. For case I, a grid of inputs $\Delta v_1 = 10$, $\Delta v_{2/3} = 5$ was used, i.e. 175 nodes. Case II, on the other hand, was simulated on a grid $\Delta v_1 = 0.02$, $\Delta v_2 = 10$ and $\Delta v_3 = 0.5$, a total of 224 nodes. In order to compare the spread in the results, relative standard deviation was used, defined as [1]:

$$\mathrm{RSD} \equiv \frac{\sigma}{\mu}, \tag{6.5}$$

where σ is the standard deviation of the data set, and μ is the mean.

Table 6.5: Results of the robustness and efficiency tests for the PRICO process.

	Case I		**Case II**	
v_1	[110, 150]		[0.120, 0.180]	
v_2	[250, 280]		[200, 330]	
v_3	[10, 30]		[1.00, 2.50]	
N	175		224	
	EO model	**Hybrid**	**EO model**	**Hybrid**
Solve %[1]	100.00	92.57	69.64	98.21
Iterations:				
Mean	159.99	100.10	154.63	118.10
Standard dev.	62.19	37.95	80.83	42.86
RSD %	38.87	37.91	52.27	36.29
Time per iteration:				
Mean [s/iteration]	0.034	0.143	0.036	0.144
Standard dev.	0.001	0.006	0.001	0.011
RSD %	2.94	4.20	2.78	7.64
Simulation time [s]:	5.44	14.31	5.57	17.01

[1] Percentage of simulations that converged within 500 iterations.

Counter-intuitively, results show that the alternative model actually performed worse in the first test. It failed to converge for 13 nodes where the original model succeeded, while being about 2.6 times slower on average. Investigating some of these nodes further, it was discovered that the model hits cycles in the nonsmooth Newton method for certain outlet temperatures of the HP refrigerant. How cycles occur in Newton-type methods can easily be illustrated for the smooth case:

Example 6.1. *Let* $X = [-3,\ 3]$,

$$f : X \subset \mathbb{R} \to \mathbb{R} : x \to x^3 - 2x + 2.$$

Then for any $x \in X$,

$$f'(x) = 3x^2 - 2.$$

Solving the equation $f(x) = 0$ *using Newton's method yields a cycle whenever* x^k *is either* 0 *or* 1.

At $x = 0$:

$$\begin{aligned} x^{k+1} &= x^k - \frac{f\left(x^k\right)}{f'\left(x^k\right)}, \\ &= 0 - \frac{0 - 2\cdot 0 + 2}{3\cdot 0 - 2}, \\ &= 0 - \frac{2}{-2} = 1. \end{aligned}$$

At $x = 1$:

$$\begin{aligned} x^{k+1} &= 1 - \frac{1^3 - 2\cdot 1 + 2}{3\cdot 1^2 - 2}, \\ &= 1 - \frac{1}{1} = 0. \end{aligned}$$

Therefore, upon hitting either of these points, Newton's method will oscillate and fails to converge.

For Case II, on the other hand, the hybrid framework is significantly more robust than its EO counterpart, converging 98% of the time. Furthermore, the average number of iterations required to solve the model is about 3/4 of the original. However, each iteration takes 4 times longer, making the method 3 times slower on average. The cause of this increase in simulation time is discussed in Section 6.4.

6.3.2 The extended PRICO process

Apart from the extra MHEX, the "extended" PRICO is identical to the original PRICO process. However, adding the second heat exchanger increases the number of variables from 52 to 108 for the EO model and from 28 to 56 for the hybrid solution. Also, with three supplementary MHEX equations, the model solves for six variables rather than three. Case I evaluates the HP and LP refrigerant temperatures out of the heat exchangers, as well as both UA values. The second case, however, keeps both HP refrigerant outlet temperatures fixed, instead solving for the refrigerant's flowrate and low pressure level.

Before conducting the performance analysis, Case I was tested with the initial guesses from Table 6.7. Both models converged to the same solution for the unknown variables, with overlapping driving force profiles in the MHEXs (see Figure 6.5). The HP and LP refrigerant outlet temperatures, the LP refrigerant inlet temperature, and the total duty in the process are the same as for the normal PRICO (see Table 6.2). In addition, Figure 6.5 resembles the ΔT plot for Case I of the PRICO process. There are some deviations between the curves in Figures

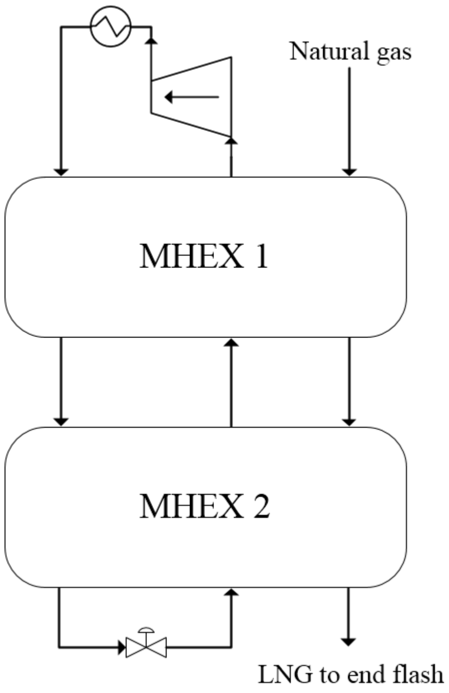


Figure 6.4: The extended PRICO process.

6.2 and 6.5, however, especially for the low temperature portion of the process. In the extended model, twice the number of segments are used to approximate phase-regions that occur in both heat exchangers, resulting in a better approximation of the composite curves and a tighter design. This becomes evident looking at the combined UA value, which is slightly higher for the extended PRICO (19.03 MW/K versus 18.70 MW/K).

Performance testing of the extended PRICO model for Case I was done in two parts to limit the number of nodes in the grid. The first simulations vary the LP refrigerant outlet temperatures $v_{2/3}$, and the HP refrigerant temperature out of MHEX 2 (v_4). Test 2 varies the UA values instead of variables v_3 and v_4. By comparing Tables 6.5 and 6.8, the equation-oriented approach is observed to perform notably worse for the extended model. Additional equations and variables in the model makes it difficult to find suitable inputs for the Newton-type method. The convergence ratio dropped from 100% to 30% for Test 1, and to 43.5% for Test 2, as well as requiring approximately 2.5 times the number of iterations to converge. With a tripling of the time per iteration compared to the original PRICO, these extra iterations greatly affect the efficiency of the equation-oriented model.

Table 6.6: Refrigerant stream and MHEX data used in simulations of the extended PRICO process and the single mixed refrigerant process with integrated NGL extraction (Case I only).

Property	**Case I**	**Case II**
High pressure level [MPa]	3.695	3.695
Low pressure level [MPa]	0.170	v_1
Flowrate [kmol/s]	3.47	v_2
MHEX 1:		
HP inlet temperature [K]	303.15	303.15
HP outlet temperature [K]	v_1	240
LP inlet temperature [K]	v_2	v_3
LP outlet temperature [K]	v_3	v_4
MHEX 2:		
HP inlet temperature [K]	v_1	240
HP outlet temperature [K]	v_4	122
LP inlet temperature [K]	v_5	v_5
LP outlet temperature [K]	v_2	v_3
Composition [mol %]		
N_2	15.32	15.32
CH_4	17.79	17.79
C_2H_6	40.85	40.85
C_3H_8	0.41	0.41
n-C_4H_{10}	25.62	25.62
MHEX data		
UA MHEX 1 [MW/K]	v_6	v_6
UA MHEX 2 [MW/K]	v_7	v_7
$\Delta T_{\min}$ [K]	1.2	1.2

Nevertheless, since the average number of iterations required by the EO model is relatively close to the self-imposed iteration limit, it is reasonable to assume some of the simulations may have converged had it been raised.

In contrast, the hybrid model achieved a much higher convergence ratio of 87.5% and 100% for the two tests. Moreover, it required about the same number of iterations to converge as for the original PRICO process. Compared to the EO model, however, test results show a bigger relative deviation in the number of iterations, indicating that there exist areas for which the alternative method struggles. Also, the hybrid model, like the EO alternative, sees a tripling in time per iteration. Fortunately, fewer required iterations make the effect far less pronounced, and the hybrid framework is actually faster than the EO model for Test 2.

Table 6.7: Numerical results of Case I for the extended PRICO process.

Variable	**Initial values**	**Results**	
		EO model	**Hybrid model**
v_1 [K]	240.00	240	240.00
v_2 [K]	220.00	222.89	222.89
v_3 [K]	280.00	286.54	286.54
v_4 [K]	120.00	121.16	121.16
v_5 [K]	116.95	116.95	116.95
v_6 [MW/K]	10.00	1.31	1.31
v_7 [MW/K]	10.00	17.72	17.72
Process:			
Total duty [MW]		85.97	85.97

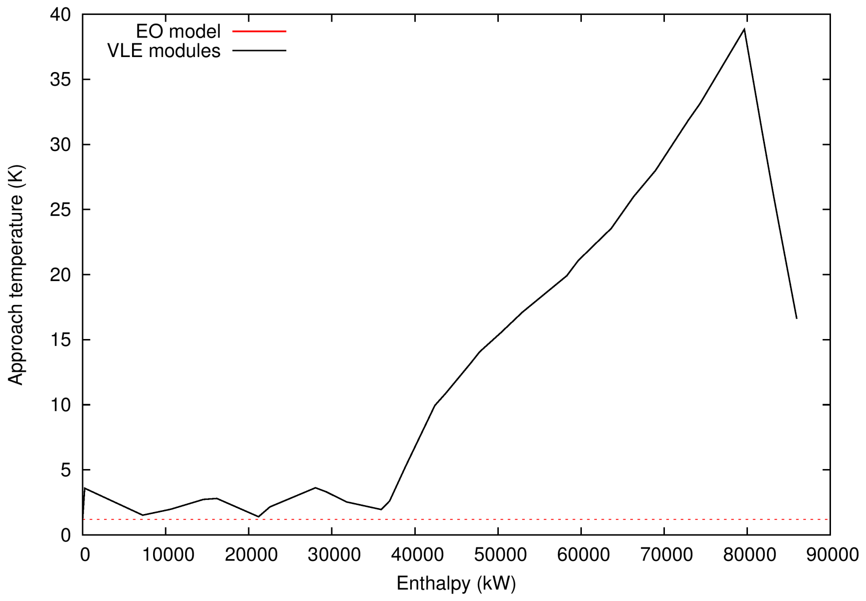


Figure 6.5: Case I driving force plots for the equation-oriented and hybrid extended PRICO models.

Table 6.8: Results of the robustness and efficiency tests for Case I of the extended PRICO model.

	Test 1		**Test 2**	
v_1 [K]	240		240	
v_2 [K]	[180, 230]		[180, 230]	
v_3 [K]	[250, 300]		280	
v_4 [K]	[110, 160]		120	
v_5 [K]	116.95		116.95	
v_6 [MW/K]	10		[5, 30]	
v_7 [MW/K]	10		[5, 30]	
N	216		216	
	EO model	**Hybrid**	**EO model**	**Hybrid**
Solve %[1]	30.09	87.50	43.52	100.00
Iterations:				
Mean	403.62	134.51	399.00	78.54
Standard dev.	51.28	47.49	60.04	18.37
RSD %	12.71	35.31	15.05	23.39
Time per iteration:				
Mean [s/iteration]	0.101	0.424	0.101	0.422
Standard dev.	0.001	0.005	0.003	0.006
RSD %	0.99	1.18	2.97	1.42
Simulation time [s]:	40.77	57.03	40.30	33.14

[1] Percentage of simulations that converged within 500 iterations.

In Case II, the HP refrigerant outlet temperatures are replaced with the refrigerant flowrate, and low pressure level as variables. Table 6.9 shows the two frameworks converged to slightly different solutions; yet with a termination tolerance of 10^{-9} it is unlikely the result of numerical errors. Instead, different ways of handling the VLE calculations may lead to the nonsmooth Newton method computing alternative step directions, ultimately terminating at separate solutions. Figure 6.6 indicates a similar distribution of driving forces for the two frameworks, though the hybrid model is somewhat shifted towards higher duties in the warm end. Small deviations in total duty are expected due to different refrigerant flowrates. The effect is bigger in MHEX 1, where the high pressure refrigerant is in the two-phase region, and therefore dominates the mC_p for the hot composite curve.

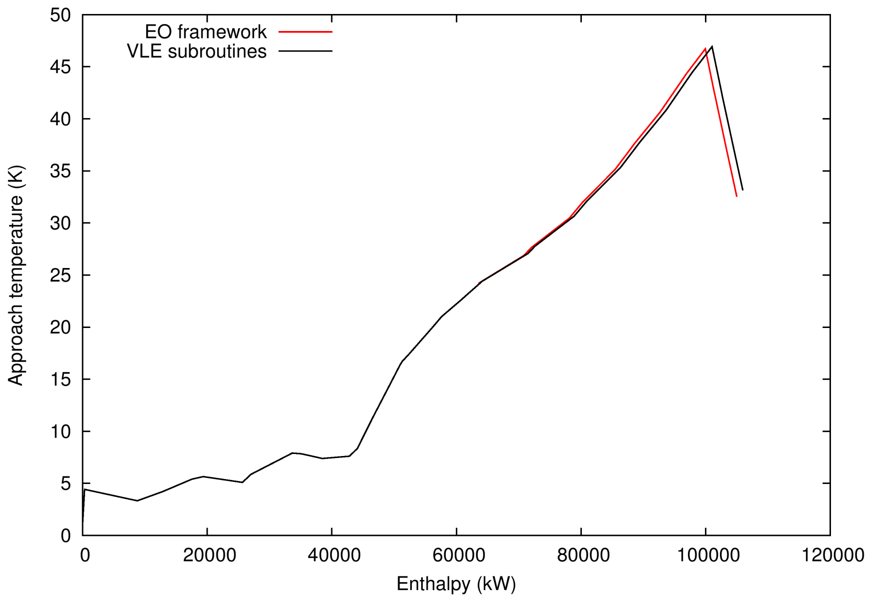


Figure 6.6: Case II driving force plots for the equation-oriented and hybrid extended PRICO models.

The hybrid framework is also considerably more robust than the fully equation-oriented model for Case II (see Table 6.10). In addition, it requires less than half the number of iterations to find the solution. However, the additional time required per iteration, makes it between 1.5-1.8 times slower on average.

Table 6.9: Numerical results of Case II for the extended PRICO model.

Variable	**Initial values**	**Results**	
		EO model	**Hybrid model**
v_1 [MPa]	0.15	0.14	0.14
v_2 [kmol/s]	3.00	4.41	4.46
v_3 [K]	220.00	215.79	215.59
v_4 [K]	270.00	270.60	270.00
v_5 [K]	116.95	116.95	116.95
v_6 [MW/K]	10.00	1.25	1.25
v_7 [MW/K]	10.00	9.71	9.68
Process:			
Total duty [MW]		105.03	105.97

Table 6.10: Results of the robustness and efficiency tests for Case II of the extended PRICO model.

	Test 1		**Test 2**	
v_1 [MPa]	[0.15, 0.20]		[0.15, 0.20]	
v_2 [kmol/s]	[2, 5]		4.00	
v_3 [K]	220		[190, 240]	
v_4 [K]	[250, 270]		280	
v_5 [K]	116.95		116.95	
v_6 [MW/K]	10		10	
v_7 [MW/K]	10		[5, 30]	
N	126		216	
	EO model	**Hybrid**	**EO model**	**Hybrid**
Solve %[1]	71.43	94.44	81.94	99.07
Iterations:				
Mean	247.29	109.66	313.58	110.30
Standard dev.	43.29	43.81	83.44	31.18
RSD %	17.51	39.95	26.61	28.27
Time per iteration:				
Mean [s/iteration]	0.105	0.432	0.10	0.432
Standard dev.	0.002	0.005	0.001	0.006
RSD %	1.90	1.16	1.00	1.39
Simulation time [s]:	25.97	47.37	31.36	47.65

[1] Percentage of simulations that converged within 500 iterations.

6.3.3 The SMR process with NGL extraction

The third process studied in this report is the single mixed refrigerant process from Figure 6.7. Apart from the vessel for natural gas liquids (NGL) extraction,

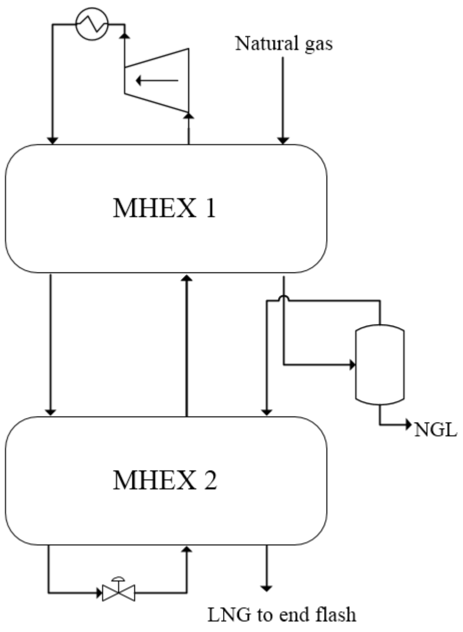


Figure 6.7: The single mixed refrigerant process with NGL extraction (from US patent no. 4,033,735 [39]).

the process is identical to the extended PRICO process. Additional variables were needed, however, to account for the varying natural gas flowrate and composition. As a result, the equation-oriented and hybrid frameworks are expanded to 120 and 68 variables, respectively. A scrubber temperature of 240K was used in the simulations. The temperature was set close to the normal boiling point of propane at −35°C to have sufficient recovery of NGLs. Also, a richer natural gas composition was used, along with a lower natural gas pressure (see Table 6.11). The SMR process was simulated for Case I in Table 6.6.

Numerical results are presented in Table 6.12. Despite significant changes in natural gas composition and the additional scrubber, the solution is close to the results for the extended PRICO process. Nevertheless, Figure 6.8 indicates that the SMR process is less tight in the low temperature region, hence the lower UA value. A higher total duty for the SMR process is reasonable given the rich natural gas composition. Heavier hydrocarbons have a higher latent heat of vaporization and

Table 6.11: Natural gas stream data used in the simulations of the SMR process with integrated NGL extraction.

Property	**Value**
Pressure [MPa]	4.00
Feed flowrate [kmol/s]	1.00
Inlet temperature [K]	298.15
Outlet temperature [K]	118.15
Composition [mol %]	
N_2	3.00
CH_4	83.00
C_2H_6	7.00
C_3H_8	5.00
n-C_4H_{10}	2.00

thus require more cooling to condense.

Table 6.12: Numerical results for the single mixed refrigerant process with integrated NGL extraction.

Variable	**Initial values**	**Results**	
		EO model	**Hybrid model**
v_1 [K]	240.00	240	240.00
v_2 [K]	220.00	222.15	222.15
v_3 [K]	280.00	289.85	289.85
v_4 [K]	120.00	120.76	120.76
v_5 [K]	116.95	116.95	116.95
v_6 [MW/K]	10.00	1.45	1.45
v_7 [MW/K]	15.00	15.76	15.76
Process:			
Total duty [MW]		86.77	86.77
Flowrate MHEX 2 [kmol/s]		0.90	0.90

Performance testing of the SMR process with NGL extraction was done for a different range of input values than the extended PRICO. In Test 1, the v_4 interval is shrunk to half the length, instead using a step-size $\Delta v_4 = 5$. In addition, the v_3 interval was shifted from $[250,\ 300]$ to $[270,\ 320]$. In Test 2, the range of v_2 was adjusted from $[180,\ 230]$ to $[190,\ 240]$.

Like for the extended PRICO, the convergence ratio is significantly higher for the hybrid model. However, it struggles more in Test 2, converging only 76% of the time. For the same test, extra iterations were required in order to solve the model. The average time per iteration is approximately 5 times that of the equation-

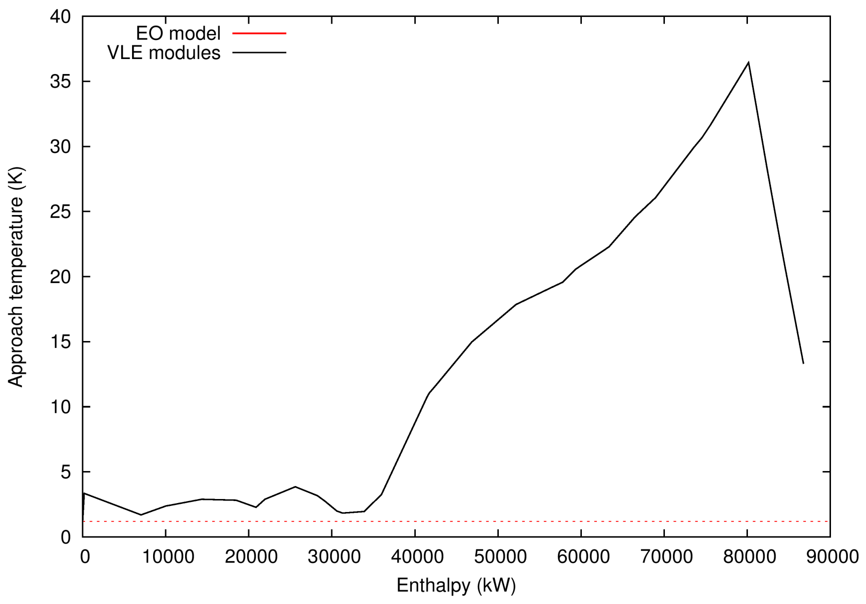


Figure 6.8: Driving force plot in the MHEXs for the simulation of the SMR process with NGL extraction

oriented model, which is higher than what was observed for the other models. On the other hand, the total simulation time is 2.6-3 times longer, due to fewer iterations required by the hybrid models. Longer iteration times provide evidence of more difficult VLE calculations, i.e. additional iterations are required for the subroutines to converge. Also, the iteration times are affected by extra variables in the model (68 compared to 56 for the extended PRICO). The equation-oriented model did not experience such an increase in time per iteration. The EO model is only slightly bigger than for the extended PRICO, plus no additional time is lost to the VLE calculations.

Table 6.13: Results of the robustness and efficiency tests for simulations of the single mixed refrigerant process with NGL extraction.

	Test 1		**Test 2**	
v_1 [K]	240		240	
v_2 [K]	[180, 230]		[190, 240]	
v_3 [K]	[270, 320]		280	
v_4 [K]	[120, 145]		160	
v_5 [K]	116.95		116.95	
v_6 [MW/K]	10		[5, 30]	
v_7 [MW/K]	15		[5, 30]	
N	216		216	
	EO model	**Hybrid**	**EO model**	**Hybrid**
Solve %[1]	48.15	86.11	50.46	75.93
Iterations:				
Mean	197.35	117.98	344.07	188.76
Standard dev.	66.49	42.74	26.27	20.32
RSD %	33.69	36.23	7.64	10.76
Time per iteration:				
Mean [s/iteration]	0.112	0.567	0.116	0.567
Standard dev.	0.002	0.007	0.009	0.012
RSD %	1.79	1.23	7.76	2.12
Simulation time [s]:	22.10	66.89	39.91	107.03

[1] Percentage of simulations that converged within 500 iterations.

6.4 VLE modules and simulation times

As observed in Section 6.3, nesting the VLE calculations makes the model significantly slower than solving everything simultaneously. For the test cases, the time per iteration was between 4 and 5 times longer for the hybrid framework. What causes this drop in efficiency can be found from the call graph of the hybrid model.

Figure 6.9 shows that the vapour-liquid equilibrium calculations make up approximately 50% of the required iteration time. Of this,

$$\frac{47.63}{48.42} = 98.37\%,$$

is spent on evaluating the derivatives of the modules, meaning that less than 2% is used for the "actual" VLE calculations. The algorithm presented in Chapter 5 solves for each column of the LD-derivative sequentially using an iterative approach. With a directions matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, therefore, n equations systems are needed to

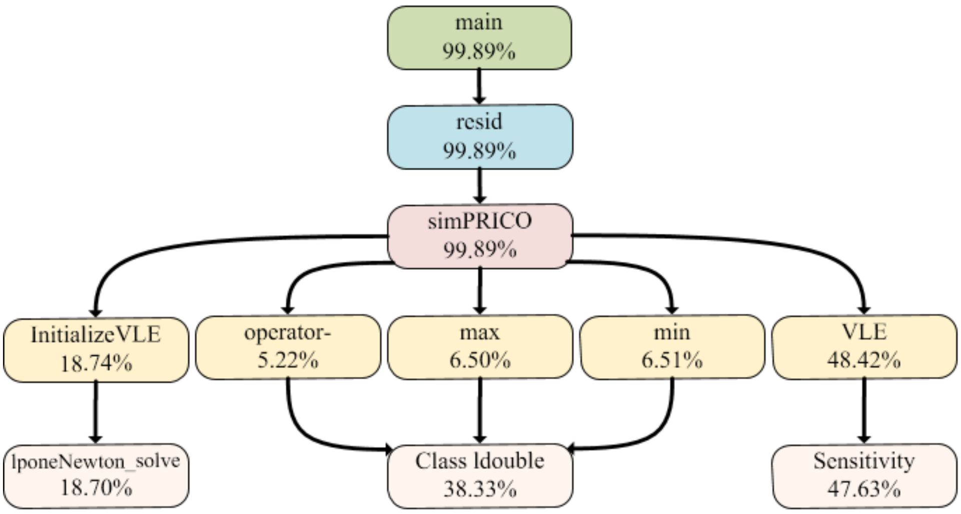


Figure 6.9: The function call distribution for the hybrid PRICO model.

find the appropriate sensitivities for an equation. For the models in this report, the dimensions of $\mathbf{M}$ corresponds to the total number of variables; e.g. which equals 28 for the hybrid PRICO. Since each vapour-liquid equilibrium module consists of two equations, $2n$ sensitivity calculations are required. All the simulations in this chapter were conducted for $n_{2\text{p}} = 5$, such that each three-stream MHEX consists of 3×5 flash calculations excluding the throttling valve. Adding an extra heat exchanger, therefore increases the number of sensitivity calculations by

$$30 \cdot n_{\text{MHEX}}. \tag{6.6}$$

Besides from the VLE and accompanying derivative calculations, initializing the modules accommodates for the biggest portion of CPU time. Currently, the VLE modules are being initialized on every iteration of the MHEX model, which is computationally inefficient. The primary reason for this is that the VLE variables are not variables in the MHEX model, i.e. they are not saved for subsequent iterations. Storing these variables will therefore require modifications to the model. Also, proper initialization of the two-phase variables becomes more important for complex LNG models. The test cases which did not converge in Section 6.3, frequently failed in the VLE calculations, and without an initialization procedure it is likely that robustness will suffer. Furthermore, poor inputs to the LNG model cause large variations in the stream variables, making the initialization procedure quite useful.

Chapter 7

Sequential-modular liquefaction models

This chapter studies the multistream heat exchanger model in a sequential-modular framework. First, a general outline of the approach is presented, plus a short description of the MHEX module. Next, in Section 7.2, the PRICO model is used for testing and validating the MHEX module. Finally, performance testing is conducted for the PRICO and extended PRICO processes.

7.1 Sequential-modular framework

Natural gas liquefaction processes are complex systems that consist of various sub-processes, e.g. heat exchange, expansion, separation, and compression. The sequential-modular approach handles each process separately using standardized sub-routines, in a calculation sequence similar to the information flow in the model (see Section 3.1). Process recycles are managed by tearing streams and instead simulating an acyclic flowsheet. The tear variables are expressed by a set of tear equations (3.1), implicitly defined from the modules. Algorithm 5, presents the general framework of the sequential-modular approach.

Algorithm 5 Sequential-modular approach for flowsheet modeling

```
1: while $\left\|\mathbf{w}^{k+1} - \mathbf{w}^{k}\right\|_\infty \geq \epsilon$ do
2:     Set $\mathbf{w}^{k+1} \leftarrow \mathbf{t}\left(\mathbf{x}, \mathbf{w}^{k}\right)$
3: end while
```

The tear equations $\mathbf{t}\left(\mathbf{x}, \mathbf{w}^k\right)$ can be solved iteratively using either direct substitution, or a nonsmooth Newton solver, e.g. Equation (2.18), which is the approach here.

7.1.1 The MHEX module

Constructing the MHEX module is analogous to the VLE subroutines in Chapter 6. It consists of the same equations as the original model and can thus solve for the same number of unknowns. In order to successfully incorporate it into a sequential-modular framework, however, the MHEX model had to be converted to an input-output format. Inputs to the module are the inlet stream variables and model parameters. Since a nonsmooth Newton solver is going to be used for solving the tear equations, derivative information is needed. A pseudo-code of the MHEX module is presented in Algorithm 6.

Algorithm 6 The multistream heat exchanger module

```
   Let $\boldsymbol{\alpha}_{\mathrm{in/out}}$ and $\mathbf{T}_{\mathrm{in/out}}$ be the collection of inlet and outlet vapour fractions
   and temperatures, respectively.
1: function MHEX-MODULE($\boldsymbol{\alpha}_{\mathrm{in}}$, $\mathbf{T}_{\mathrm{in}}$, $\mathbf{p}$)
2:     Solve ($\boldsymbol{\alpha}_{\mathrm{out}}$, $\mathbf{T}_{\mathrm{out}}$) ←MHEX-MODEL($\boldsymbol{\alpha}_{\mathrm{in}}$, $\mathbf{T}_{\mathrm{in}}$, $\mathbf{p}$)
3:     Evaluate the sensitivities: $\mathbf{g}'(\boldsymbol{\alpha}_{\mathrm{out}}, \mathbf{T}_{\mathrm{out}}; \mathbf{M})$ ←SENSITIVITY($\boldsymbol{\alpha}_{\mathrm{out}}$, $\mathbf{T}_{\mathrm{out}}$,
   MHEX-MODEL($\boldsymbol{\alpha}_{\mathrm{in}}$, $\mathbf{T}_{\mathrm{in}}$, $\mathbf{p}$))
4: end function
```

As it turns out, the MHEX module is more constrained compared to the equation-oriented model. The reason for this is the input-output structure, which treats all input streams as constant. A more general discussion about this issue is given in Section 7.3.3.

7.2 Sequential-modular PRICO simulation model

Before studying more advanced liquefaction processes, the sequential-modular approach was first tested for the basic PRICO process. Figure 7.1 reveals two process recycles, each requiring a tear stream. However, the simulation models in this report only consider the cooling and heating curves in the MHEX, with no attention put on the compressor and aftercooler section. Instead, inlet conditions for the high pressure refrigerant are assumed fixed, thus reducing the number of tear streams to one.

The liquefaction section of the PRICO process consists of two sub-processes: isenthalpic throttling and heat exchange. A tear stream with $C + 2$ corresponding

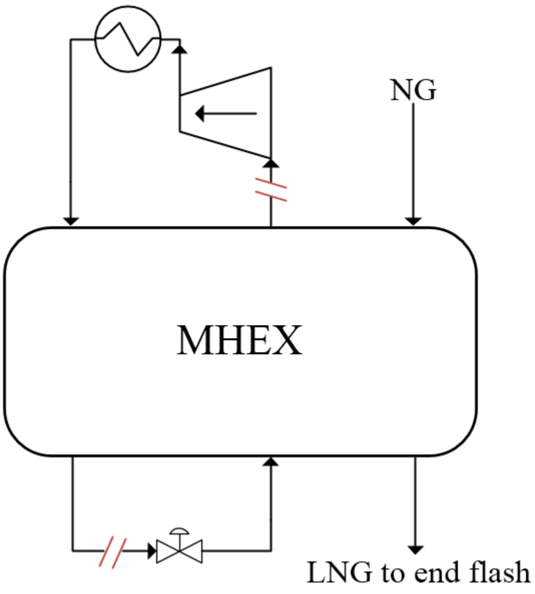


Figure 7.1: The PRICO process with tear streams indicated.

variables is needed at the cold end of the MHEX. In this particular case, however, the composition will remain fixed throughout the refrigerant cycle, therefore limiting the number of tear variables to two. The vapour fraction and stream temperature were picked as tear variables in the problem. Applying Algorithm 5 directly, the method in Algorithm 7 can be derived.

The sequential-modular framework was verified using the test cases studied for the PRICO process in Chapter 6 (see Table 6.1). However, different input values were needed to achieve convergence (the changed input value is emboldened).

Table 7.1: Numerical results of Case I for the sequential-modular PRICO process.

Variable	**Initial values**	**Results**	
		EO model	**SM model**
v_1 [K]	**120.00**	120.77	120.77
v_2 [K]	250.00	286.55	286.55
v_3 [MW/K]	10.00	18.82	18.82
v_4 [K]	116.95	116.60	116.60
Process:			
Total duty [MW]		86.11	86.11

Algorithm 7 The sequential-modular PRICO simulation model

```
Let $\mathbf{f}_t : S_1 \subset \mathbb{R}^{n_w+n_y} \to \mathbb{R}^{n_y}$ denote the module equations for the throttling valve, and let $\mathbf{f}_{\text{MHEX}} : S_2 \subset \mathbb{R}^{n_p+n_y+n_w} \to \mathbb{R}^{n_w}$ be the MHEX model. Here, $n_y = n_w = 2$ as there are two tear variables in the problem, and $n_p$ are the total number of parameters in the MHEX model.
1: Choose any nonsingular directions matrix $\mathbf{M} \in \mathbb{R}^{n_w \times n_w}$.
2: Provide an initial guess for $\mathbf{w}$.
3: **while** $\|\mathbf{w}^* - \mathbf{w}\|_\infty \geq \epsilon$ **do**
4:     Solve the throttling valve module $\mathbf{f}_t(\mathbf{w}, \mathbf{y}) = \mathbf{0}_{n_y}$ for $\mathbf{y}$, and compute the LD derivatives $\mathbf{g}_1'(\mathbf{w}; \mathbf{M}) = \mathbf{0}_{n_w}$ for the implicit function $\mathbf{y} = \mathbf{g}_1(\mathbf{w})$.
5:     Solve the MHEX module $\mathbf{f}_{\text{MHEX}}(\mathbf{p}, \mathbf{y}, \mathbf{w})$ for the tear variables $\mathbf{w}^*$, and evaluate $\mathbf{g}_2'(\mathbf{p}, \mathbf{y}; \mathbf{g}_1'(\mathbf{w}; \mathbf{M}))$.
6:     Evaluate the residual $\mathbf{r} = \mathbf{w}^* - \mathbf{w}^k$, and use a nonsmooth Newton-type method to compute the next iterate $\mathbf{w}^{k+1}$.
7: **end while**
```

As seen from Table 7.1 and Figure 7.2, both frameworks converged to the same solution. Results deviate slightly from Table 6.2, yet the driving force profiles remain strikingly similar. Different solutions are expected given the new initial guess. Nevertheless, their close proximity further strengthen the validity of the sequential-modular framework.

Case II experienced convergence issues in a sequential-modular framework. No solution was found when the HP refrigerant outlet temperature remained fixed to 118.5K (see Table 7.3). Instead it had to be raised to 120K with accompanying adjustments in inputs. Again, both models converged to the same solution. Therefore, by the argument made for the VLE modules, the sequential-modular framework is verified.

Table 7.2: Numerical results of Case II for the PRICO process.

Variable	**Initial values**	**Results**	
		EO model	**SM model**
v_1 [MPa]	0.140	0.120	0.120
v_2 [K]	**260.00**	271.38	271.38
v_3 [K]	**2.50**	2.80	2.80
v_4 [K]	**115.65**	114.39	114.39
Process:			
Total duty [MW]		83.46	83.46

Since the HP refrigerant outlet temperature is held constant here, the tear stream needs to be changed. If not, the solver will break out of the loop after the first

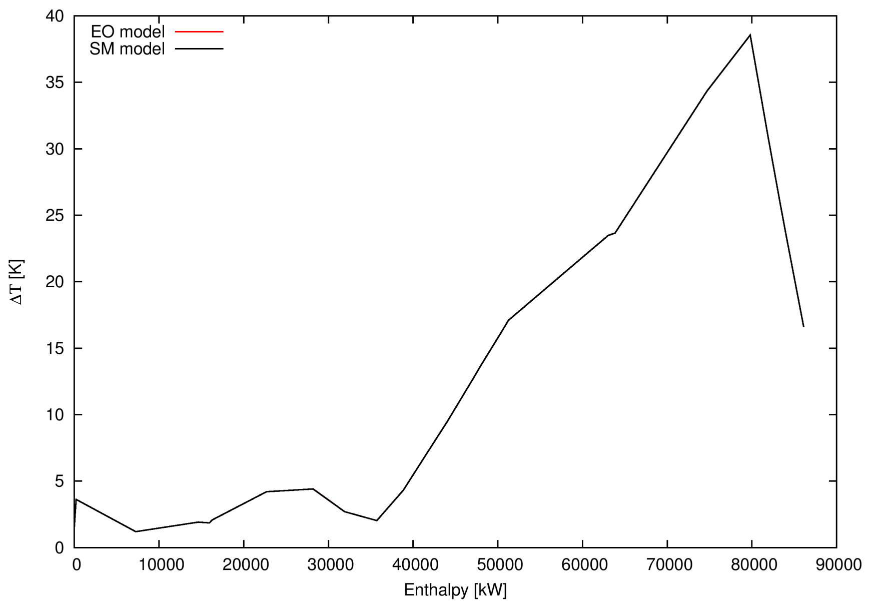


Figure 7.2: Case I driving force plots for the sequential-modular and equation-oriented PRICO models.

iteration. The only alternative tear location is on the low pressure side of the throttling valve. It is important to note that changing the tear stream also alters the calculation sequence of the open flowsheet in Algorithm 7.

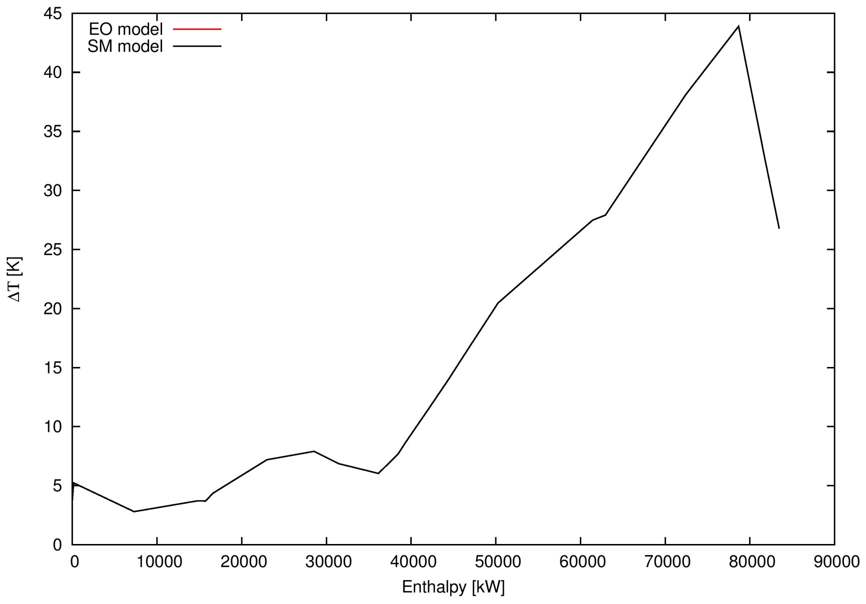


Figure 7.3: Case II driving force plot for the sequential-modular and equation-oriented PRICO models.

7.3 Performance testing of the sequential-modular framework

Robustness and efficiency testing of the sequential-modular framework was carried out the same way as in Section 6.3. Ideal properties were used throughout, with $n_{2\mathrm{p}}$ set to 5. The maximum number of iterations was limited to 500 for the individual modules, and to 50 for the overall model. Natural gas data from Table 6.4 was used.

7.3.1 The PRICO process

The sequential-modular model of the PRICO flowsheet was tested using the grid of inputs from Section 6.3.1. By comparing Tables 6.5 and 7.3, the sequential-modular model clearly under-performs for both cases. Only 30% of the simulations converged for Case I, which is about a third of the other frameworks. It was especially sensitive to the initial HP refrigerant outlet temperatures (v_1), solving only for $v_1 = \{110,\ 120\}$.

Table 7.3: Results of the robustness and efficiency tests for the PRICO process.

	Case I	**Case II**
v_1	[110, 150]	[0.120, 0.180]
v_2	[250, 280]	[200, 330]
v_3	[10, 30]	[1.00, 2.50]
N	175	224
Solve %[1]	31.43	0.00
Total simulation time:		
Mean [s]	38.78	–
Standard dev.	27.14	–
RSD %	69.98	–

[1] With the maximum number of iterations set to 500 for the individual modules and 50 for the overall flowsheet.

Even when the model converged, however, it was still–in accordance with the theory presented in Chapter 3–slower on average. Table 7.4 show the simulation times for the three frameworks. The sequential-modular model was 2.7 and 7.1 times slower than the hybrid and equation-oriented methods, respectively. In addition, the simulation times fluctuated more violently because of the computing time associated with additional flowsheet passes.

Table 7.4: Total simulation times for the three frameworks.

	SM model	**Hybrid model**	**EO model**
Average simulation time [s]	38.78	14.26	5.43
Standard deviation	27.14	5.40	2.10
RSD %	69.98	37.87	38.67

None of the simulations converged for Case II with the refrigerant stream data in Table 6.1. Instead, the HP refrigerant outlet temperature had to be raised to 120K before a solution was found. Nevertheless, the sequential-modular model still performed significantly worse than the hybrid alternative. Not only was it less robust, but it was also approximately 6 times slower on average (see Table 7.5). Similarily to Case I, the SM model struggled for a particular range of inputs, though here it was the $\Delta T_{\min}$ variable (v_3). Specifically, the model converged only for $v_3 = \{2,\ 2.5\}$.

Table 7.5: Results of the robustness and efficiency tests for the PRICO process.

	Case II at 120K	
	SM model	**Hybrid model**
Solve %	48.21	98.21
Total simulation time:		
Mean [s]	94.77	16.35
Standard dev.	25.16	6.14
RSD %	26.55	37.57

7.3.2 The extended PRICO process

The extended PRICO process is presented in Figure 7.4 with a possible tear set. An additional tear stream is needed for the midsection between the two MHEXs, adding two extra equations to the outer problem. Apart from this, the program structure for the extended PRICO remains similar to Algorithm 7.

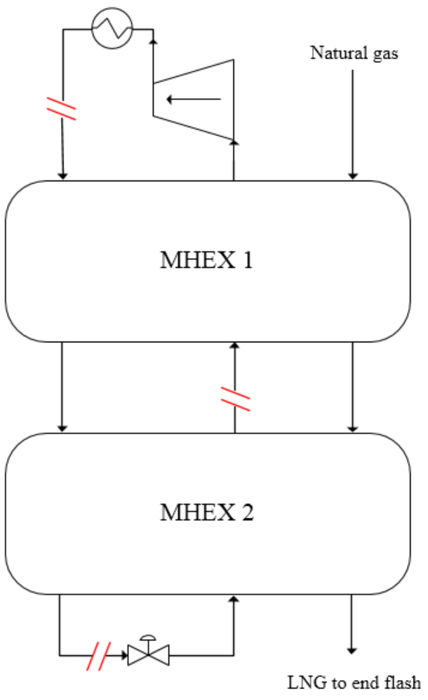


Figure 7.4: The extended PRICO process with tear streams indicated.

Like for the sequential-modular PRICO, the model suffered from convergence difficulties. Therefore, rather than conducting the performance tests using the grid of inputs from Section 6.3.2, it was tightened around the known solution. Node spacings were also made smaller with $\Delta v_2, \Delta v_3$ and Δv_4 equal to 5 for Test 1. For Test 2, the UA intervals were left unchanged while the v_2 range was reduced.

Table 7.6: Results of the robustness and efficiency tests for Case I of the sequential-modular extended PRICO model.

	Test 1	**Test 2**
v_1 [K]	240	240
v_2 [K]	[215, 230]	[210, 235]
v_3 [K]	[270, 290]	280
v_4 [K]	[115, 130]	120
v_5 [K]	116.95	116.95
v_6 [MW/K]	10	[5, 30]
v_7 [MW/K]	10	[5, 30]
N	80	216
Solve %[1]	0.00	0.00
Total simulation time:		
Mean [s]	–	–
Standard dev.	–	–
RSD %	–	–

[1] With the maximum number of iterations set to 500 for the individual modules and 50 for the overall flowsheet.

Despite efforts trying to get the SM model to converge, no solution was found for the extended PRICO. Frequent convergence errors were caused by poor HP refrigerant inlet temperatures into MHEX 2, a direct consequence of the calculation sequence created by the acyclic network. Upstream modules do not take into account the subsequent subroutines, and can thus produce outputs for which there is no solution. In the extended PRICO case, MHEX 1 converged by lowering the HP outlet temperature while keeping the LP outlet temperature constant. As a result, no solution could be found for the second MHEX, eventually causing the model to fail. The sequential-modular framework exhibited such poor convergence for the SMR processes studied that it will be unpractical for simulating more complex LNG liquefaction processers.

7.3.3 Convergence issues for the sequential-modular framework

The performance tests demonstrate a drop in robustness for the sequential-modular framework, despite having to solve less equations simultaneously. The MHEX module, for instance, consists of just 24 variables, i.e. 4 less than the hybrid PRICO model. However, due to the input-output format of the MHEX module, it is more constrained than for the models in Chapter 6.

The equation-oriented and hybrid frameworks calculate the whole network simultaneously, and as a consequence, inlet variables to the MHEXs need only be feasible at the solution. For sequential-modular models, on the other hand, the inlet temperatures are held fixed for every iteration of the acyclic flowsheet, and are instead treated by the tear equations in an outer loop. Each solution of the MHEX module, therefore, corresponds to a set of inlet temperatures. However, for certain sets of inputs there exist no solution at all, which will prompt the sequential-modular method to fail.

Example 7.1. *A simulation of the sequential-modular PRICO model that did not converge, is shown in Figures 7.5 and 7.6. The simulation was for Case II with initial values* $(v_1, v_2, v_3, v_4) = (0.14,\ 270,\ 1.2,\ 116.95)$.

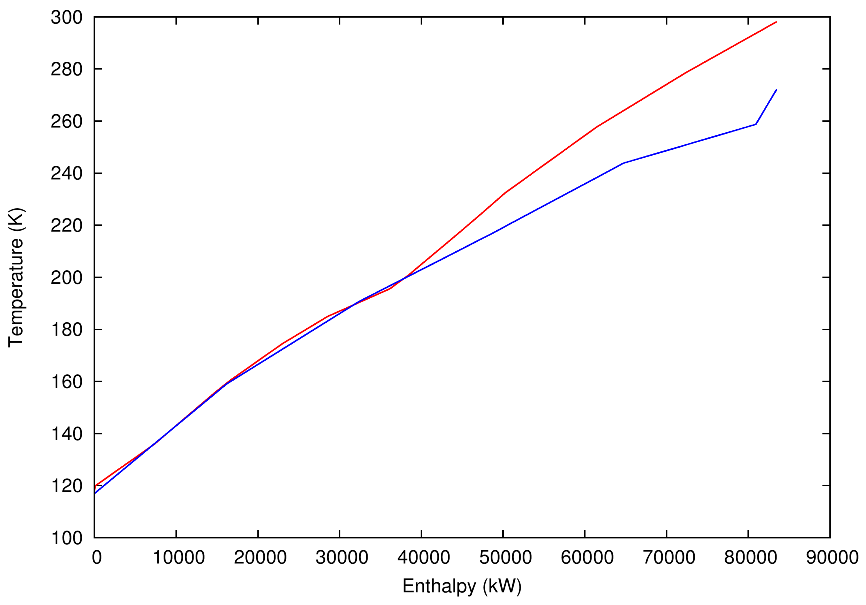


Figure 7.5: Composite curves for a simulation of the sequential-modular PRICO model that did not converge.

The high LP inlet temperature ($116.95K$) *pushes the cold composite curve towards the hot composite curve, and no feasible solution can be found for the MHEX module due to temperature crossovers. Moreover, since the LP inlet temperature is fixed for each solution of the MHEX, the solver cannot adjust it to find a feasible*

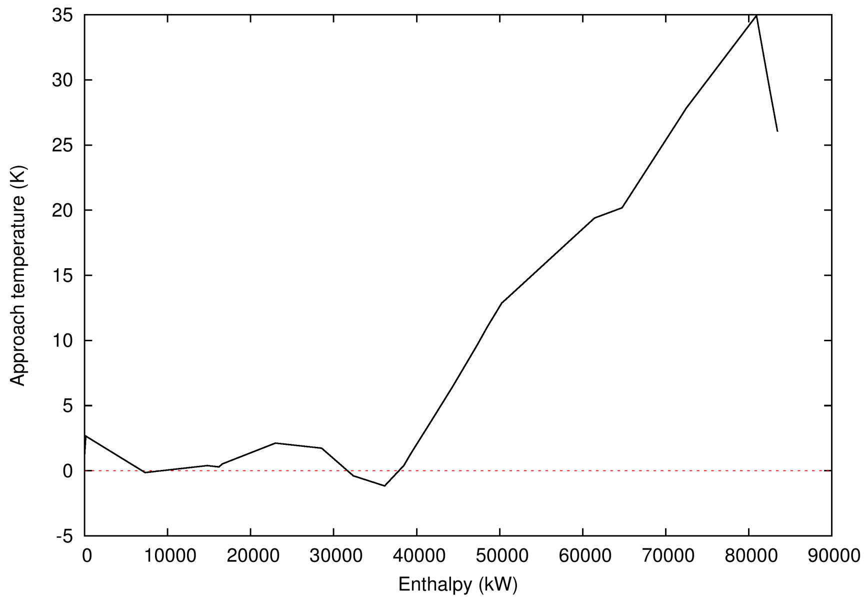


Figure 7.6: Driving force plot for a simulation of the sequential-modular PRICO model that did not converge.

solution, and thus fails to converge. On the other hand, equation-oriented and hybrid frameworks solve everything simultaneously, including the inlet variables. Consequently, if no solution exists for the given inlet variables, they are changed by the Newton-solver until a feasible solution can be found.

Another example of this issue is the strong dependency on HP refrigerant outlet temperature observed for Case I of the PRICO model. The HP refrigerant outlet temperature determines the LP inlet temperature implicitly through the throttling valve. As a consequence, temperatures greater than 120K will result in inlet temperatures for which there are no feasible solutions for the MHEX module.

Chapter 8

Simulating a more advanced LNG process

This chapter studies the more advanced single mixed refrigerant process depicted in Figure 8.1. The process was simulated in the specialization project using the equation-oriented framework, though no solution was obtained [40]. Here, the hybrid framework is employed to investigate whether it is capable of simulating more complex models. Like in Chapters 6 and 7, ideal properties were used.

8.1 The single mixed refrigerant process

The single mixed refrigerant process closely resembles the extended PRICO. However, a scrubber is placed downstream of the aftercooler to separate the high pressure refrigerant into a liquid mixture and a vapour stream. The liquid mixture is subcooled in MHEX 1 before being throttled to the low pressure level. Here it mixes with the LP refrigerant stream from MHEX 2, effectively lowering the temperature. The vapour stream, on the other hand, will provide cooling in MHEX 2. The advantage with such a setup is that heavier hydrocarbons are not cooled down to cryogenic temperatures, but are instead used for precooling. Heavy hydrocarbons in the refrigerant mixture account for a significant portion of the total energy consumption due to their high mC_{p}s. Avoiding excessive cooling, therefore, yields a large economic benefit.

MHEX 1 consists of four streams, which drastically increases the number of variables from 28 to 50. Additional variables were needed for describing the refrigerant compositions, flowrates, dew- and bubble-point temperatures, as well as stream

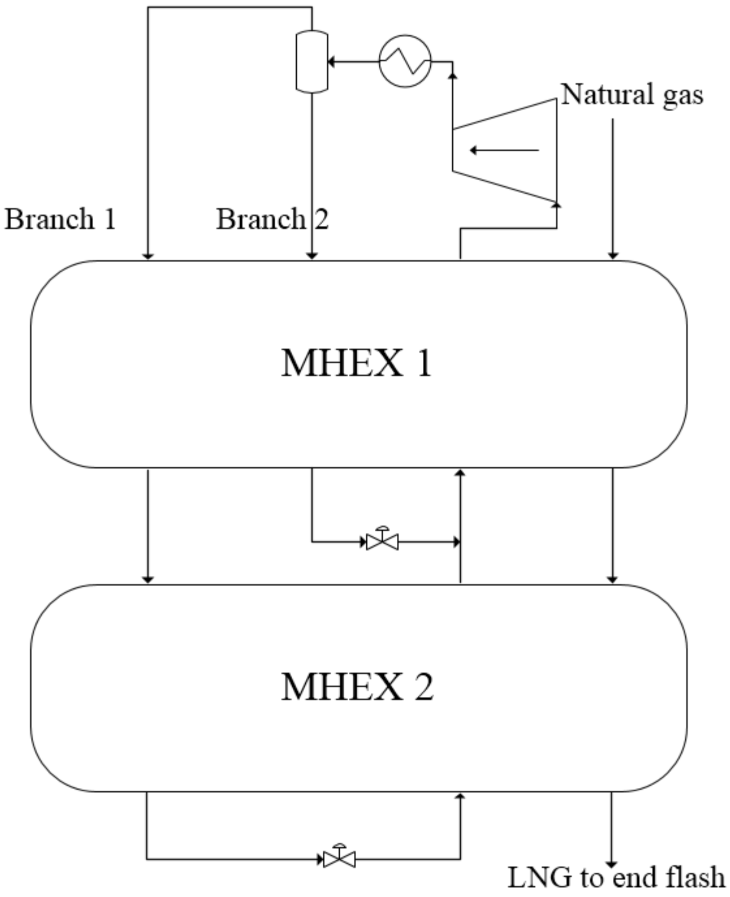


Figure 8.1: The SMR process studied is adapted from US patent no. 3,932,154 [39].

temperatures. In total, the model consists of 84 variables, which is 28 and 16 more than the extended PRICO and the SMR process with integrated NGL extraction.

With two MHEXs, the model can solve for six unknown variables, which are presented in Tables 8.1 and 8.2 along with the refrigerant and MHEX data. The simulation was performed using the natural gas composition from Table 6.4 and with $n_{2\mathrm{p}} = 4$. Tables 8.1 and 8.2 sort the variables into three types. The v_n variables are solved by the MHEX model and the throttling valves, whereas the branch compositions c_i and flowrates f_k are calculated from the inlet flash problem.

Table 8.1: Refrigerant stream and MHEX data used in MHEX 1 for simulations of the SMR process.

Property	**Value**
High pressure level [MPa]	3.695
Low pressure level [MPa]	0.150
HP inlet temperature [K]	303.15
LP refrigerant:	
Flowrate [kmol/s]	3.47
LP inlet temperature [K]	v_1
LP outlet temperature [K]	v_2
Feed composition [mol %]	
N_2	15.32
CH_4	17.79
C_2H_6	40.85
C_3H_8	0.41
n-C_4H_{10}	25.62
HP refrigerant branch 1:	
Flowrate [kmol/s]	f_1
HP outlet temperature [K]	v_3
Composition [mol %]	
N_2	c_1
CH_4	c_2
C_2H_6	c_3
C_3H_8	c_4
n-C_4H_{10}	c_5
HP refrigerant branch 2:	
Flowrate [kmol/s]	f_2
HP outlet temperature [K]	240
Composition [mol %]	
N_2	c_6
CH_4	c_7
C_2H_6	c_8
C_3H_8	c_9
n-C_4H_{10}	c_{10}
MHEX data	
UA [MW/K]	v_4

Table 8.2: Refrigerant stream and MHEX data used in MHEX 2 for simulations of the SMR process.

Property	**Case I**
High pressure level [MPa]	3.695
Low pressure level [MPa]	0.150
Flowrate [kmol/s]	f_1
HP inlet temperature [K]	v_3
HP outlet temperature [K]	v_5
LP inlet temperature [K]	v_6
LP outlet temperature [K]	v_7
Composition [mol %]	
N_2	c_1
CH_4	c_2
C_2H_6	c_3
C_3H_8	c_4
n-C_4H_{10}	c_5
MHEX data	
UA [MW/K]	v_8
ΔT_{min} [K]	1.2

8.2 SMR process simulation

Tables 8.3 and 8.4 show the converged solution for the SMR process. Unsurprisingly, there is a significant energy saving associated with splitting refrigerant streams in the process. Approximately 2/3 of the refrigerant circulates through the cold MHEX, resulting in a total duty of 69.74 MW, 16.23 MW less than Case I for the extended PRICO. On the other hand, the UA value in MHEX 2 is only about 25% of the extended PRICO model because of larger temperature differences in the exchanger.

The composite curves and driving force plot for the warm MHEX are presented in Figures 8.2 and 8.3. Figure 8.3 resembles the warm side of Figure 6.5, although more pinched at the outlet. The kink is caused by fixing the outlet temperatures for the natural gas and the second refrigerant branch to 240K, while letting the third temperature vary.

Figures 8.4 and 8.5 show the composite curves and driving force plot for the cold MHEX. A kink also exists here, due to cooling down the natural gas stream from an elevated temperature of 240K. At this temperature, natural gas is still in the superheated region, such that the mC_{p} is low. The process pinch occurs at $(T_{\text{H}}/T_{\text{C}}) = 233.42/232.22\,\text{K}$, which is the inlet of the HP refrigerant. The relatively

Table 8.3: Branch compositions and flowrates for the SMR process.

Variable	**Initial values**	**Results**
Branch 1:		
Flowrate f_1 [kmol/s]	1.86	2.22
Composition [mol %]:		
c_1	28.46	23.88
c_2	31.08	26.99
c_3	37.62	44.09
c_4	0.14	0.22
c_5	2.69	4.81
Branch 2:		
Flowrate f_2 [kmol/s]	1.61	1.25
Composition [mol %]:		
c_6	0.18	0.1
c_7	2.48	1.45
c_8	44.57	35.09
c_9	0.72	0.75
c_{10}	52.04	62.60

Table 8.4: Simulation results for the SMR process.

Variable	**Initial values**	**Results**
v_1 [K]	220	220.65
v_2 [K]	285	286.54
v_3 [K]	240	233.42
v_4 [MW/K]	10	1.47
v_5 [K]	122	119.10
v_6 [K]	116.95	109.09
v_7 [K]	220	234.71
v_8 [MW/K]	10	4.40
Total duty 1 [MW]		33.42
Total duty 2 [MW]		36.32

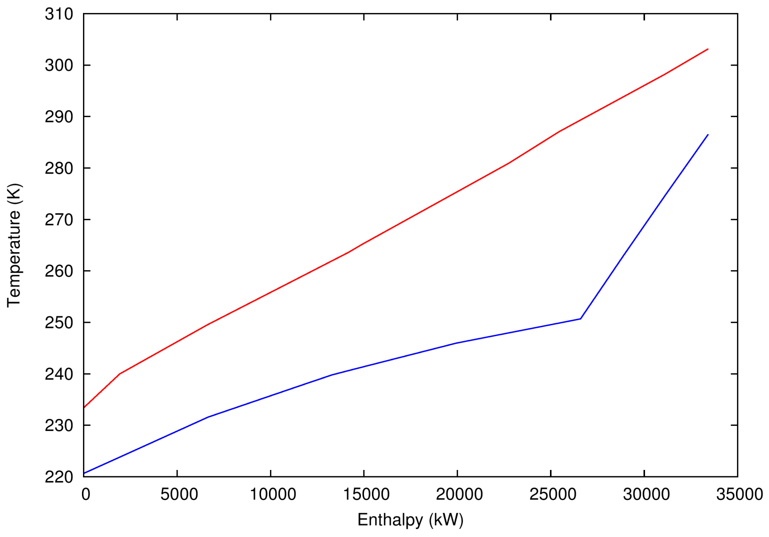


Figure 8.2: Plot showing the composite curves for MHEX 1.

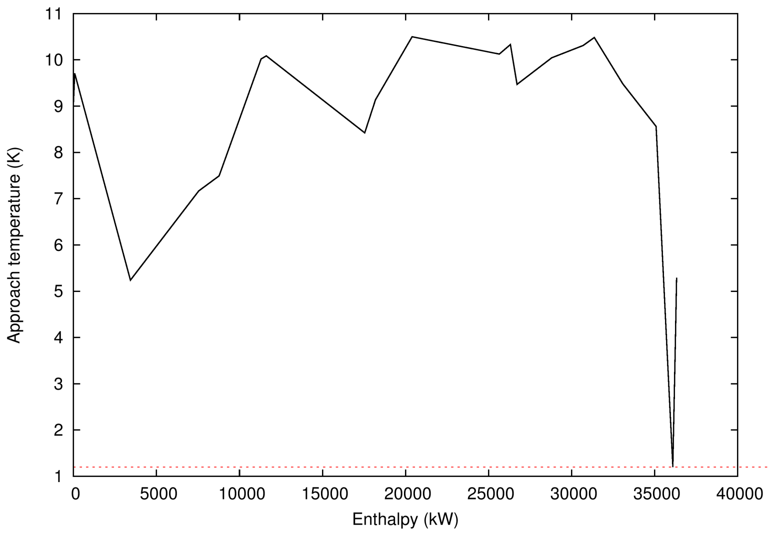


Figure 8.3: Driving force plot for MHEX 1.

high process pinch location makes the design less tight for colder temperatures, explaining the low UA value observed for MHEX 2.

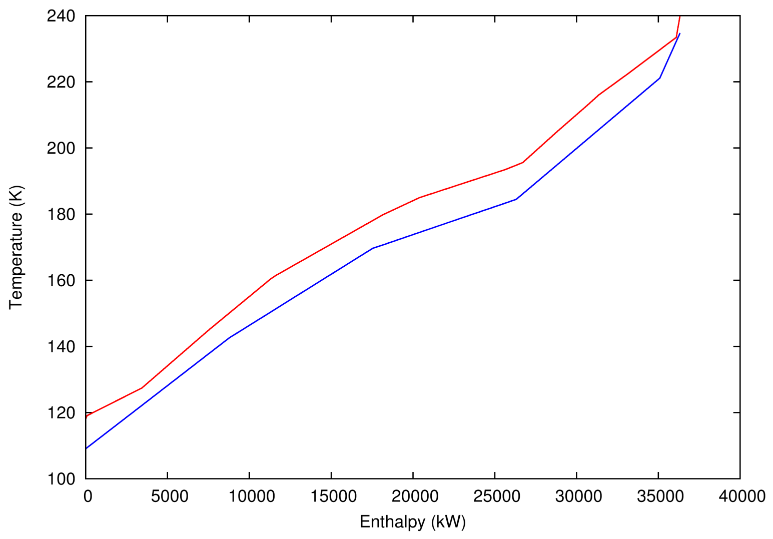


Figure 8.4: Plot showing the composite curves for MHEX 2.

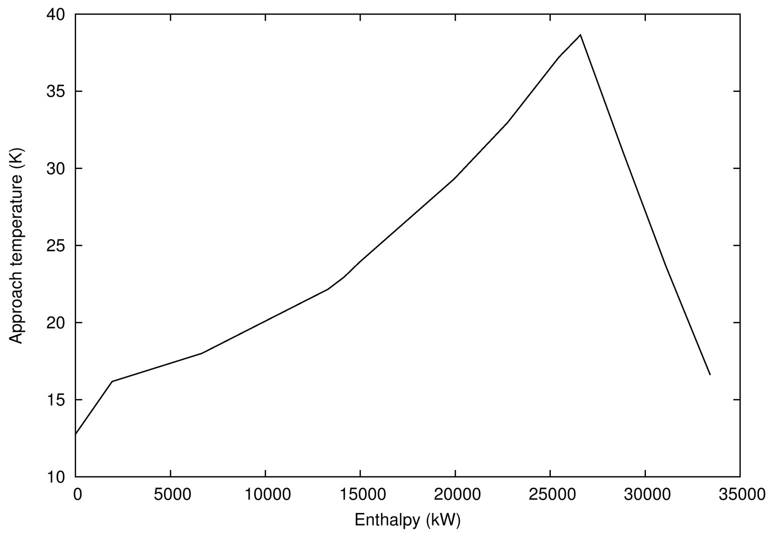


Figure 8.5: Driving force plot for MHEX 2.

The kinks in the composite curves and the pinch point location are byproducts of temperature shifts in the process. Normally, the SMR process can be represented with a single spiral wound heat exchanger (SWHE) [30], where cooling of single, low heat capacity flowrate, streams does not occur. Alternatively, the model can be solved for a different set of variables, while keeping the HP refrigerant outlet temperatures fixed. Given the current setup, however, results are reasonable, providing evidence that the framework is indeed suitable for solving more complex thermodynamic models.

The SMR model is an essential building block for larger liquefaction processes, e.g. the C_3MR and DMR designs [30]. Figure 8.6 highlights the SMR part of the DMR process (mixed refrigerant cycle 2). In addition, other mixed refrigerant cycles that employ a SWHE exhibit similar setups. The framework's ability to solve this particular model is thus an important step towards developing rigorous simulation models for complex liquefaction systems.

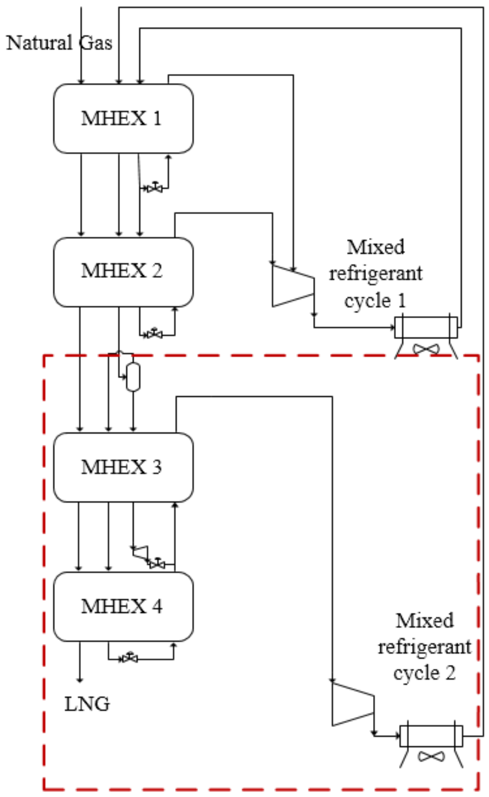


Figure 8.6: The DMR process with the SMR cycle highlighted.

Chapter 9

Conclusions and recommendations for future work

9.1 Conclusions

In this master thesis, hybrid and sequential-modular frameworks were developed and tested for different LNG processes. To successfully integrate the modular subroutines in the overall model, input-output sensitivities had to be computed using results from the implicit function theorem for lexicographically smooth functions. Apart from Case I of the PRICO process, the partially modular framework was markedly more robust than the equation-oriented alternative. For both the PRICO and the extended PRICO process the convergence ratio remained close to 100%, whereas it dropped to between 75 and 86% for the single mixed refrigerant process with integrated NGL extraction. In comparison, the convergence ratios for the EO model dropped to between 30 and 50% both in the first case of the extended PRICO and for the SMR model. In accordance with the theory, however, the gain in robustness came at the expense of reduced efficency. The time per iteration quadrupled for the PRICO and the extended PRICO process, and even quintupled for the SMR process.

According to the call-graph in Figure 6.9, approximately half the computing time in the hybrid PRICO model is spent on the vapour-liquid equilibrium modules. The sensitivity calculations make up around 98% of the unit's call-time, due to solving n_{MHEX} equation systems per flash. In addition, around 20% of the total running time is spent in an initialization procedure for the flash variables, which is called upon every iteration of the MHEX model. The procedure is needed for more complex processes and for initial guesses far away from the solution.

The hybrid framework was used to simulate the more advanced LNG process in Figure 8.1 originally studied in the specialization project, though it did not converge with the equation-oriented method. The process occurs as an important building block in other more complex and commercially interesting designs. For example, as the cold end of the DMR and C3MR processes.

The fully sequential-modular framework performed notably worse than the hybrid and equation-oriented models. It converged only a third of the time for Case I of the PRICO process, for instance, and parameters had to be adjusted for the model to solve for Case II at all. Moreover, no solution was found for the extended PRICO process. Before including the MHEX model in the sequential-modular flowsheet, it had to be redesigned in an input-output format where each solution corresponds to a given set of inlet stream variables. This made it more constrained, however, as inputs are determined either explicitly or implicitly by the tear variables and thus remain constant for each flowsheet pass. Also, with several stream outlets, the model has the freedom to vary only a few of the outputs, which might cause difficulties downstream. It is a combination of these effects that are the cause of the convergence problems experienced with the extended PRICO model. Apart from the low convergence ratio, the sequential-modular models suffered from long simulation times. For example, the sequential-modular PRICO was 2.7 and 6 times slower on average for the repective simulation cases than the hybrid framework. Poor convergence properties and long simulation times make the sequential-modular framework unsuitable for LNG process simulations.

9.2 Recommendations for future work

The simulations in this thesis were done with ideal properties, even though the software is also intended to be used with a cubic equation of state. More rigorous VLE calculations are needed before simulating LNG processes with complex thermodynamic models, and the inside-out algorithm by Boston and Britt has become a workhorse for state-of-the-art simulation tools like Aspen Plus®. To implement it in the MHEX model, however, the algorithm must include functionality for phase detection, which in the ideal case was done through the piecewise continuously differentiable mid-function (Equation (2.16)). A nonsmooth inside-out algorithm is currently under development by Watson et al. [43] that adds a modified version of (2.16) to the inner loop. The algorithm has been tested for different flash problems and has so far shown promising results.

Like the VLE module in Chapter 6, derivatives must be calculated for the output variables in order to include the nonsmooth inside-out algorithm in the hybrid framework. Applying the sensitivity code directly, however, is quite challenging as the output variables are functions of the external and internal loop variables. These are again dependent on flash inputs and external loop variables (the internal loop variables), such that nested sensitivity calculations are necessary. Furthermore, due to modifications made to the outer loop (see [43]), the algorithm currently fails to capture the output variables' dependence on the flash inputs whenever in the single-phase regions. As a result, applying the sensitivity code directly will not work unless modifications are made to the implementation.

Another possibility is to calculate derivatives from the VLE equations. Unfortunately, this turns out to be quite difficult in the current implementation of the algorithm, primarily because the inner loop approximates equilibrium constants K_i with $K_r e^{\phi_i}$ (see Section 2.4) before solving the flash problem. These approximations are only valid in the two-phase region, however, and depart from the property models in single-phase. This does not affect the solution of the flash problem, as the K-values have no real interpretation in single-phase and only affect the *shadow-composition*, i.e. an imaginary composition of the non-existent phase. However, the deviations result in the solution $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ no longer satisfying the VLE equations when using a cubic property model, such that sensitivities cannot be calculated. Therefore, a priority should be to modify the algorithm and come up with approximations for the K-values that are valid everywhere. In addition, once more rigorous VLE modules have been developed for the MHEX model, the LNG processes studied in this thesis should be simulated with a cubic equation of state.

This master thesis focused solely on flowsheet simulation, and it would therefore be interesting to try to optimize the various LNG processes. Furthermore, the models should be expanded further to check whether the hybrid framework is capable of both simulating and optimizing more commercially interesting designs like the DMR or C3MR.

Another interesting topic for further studies would be to try making the VLE subroutines more effective. As mentioned in Section 6.4, the modules plus initialization take up almost 70% of the total computing time for the PRICO model, and improving the code could therefore lead to great time savings, especially for larger models. One idea is to try making a smarter initialization procedure, for instance by re-initializing the flash variables only after a certain number of iterations or whenever the inputs have changed significantly. That way, the computing time spent on initialization may be reduced without a loss in robustness.

Bibliography

[1] Coefficient of Variation. Wikipedia. `https://en.wikipedia.org/wiki/Coefficient_of_variation`. [Online; accessed 11-April-2016].

[2] *Aspen Plus User Guide - Version 10.2*. Aspen Technology, Inc., 2000.

[3] *Aspen Properties: Toolkit Manual*. Aspen Technology, Inc., 2006.

[4] *Aspen Physical Property System: Physical Property Models*. Aspen Technology, Inc., 2012.

[5] S. Balakrishna and L. T. Biegler. Targeting strategies for the synthesis and energy integration of nonisothermal reactor networks. *Industrial & Engineering Chemistry Research*, 31(9):2152–2164, 1992.

[6] P. I. Barton. Foundations of process systems engineering. Massachusetts Institute of Technology: unpublished notes, 2003.

[7] P. I. Barton. Mixed-integer and nonconvex optimization. Massachusetts Institute of Technology: unpublished notes, 2011.

[8] L. T. Biegler. *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*, volume 10. SIAM, 2010.

[9] L. T. Biegler and J. E. Cuthrell. Improved infeasible path optimization for sequential modular simulators–II: the optimization algorithm. *Computers & Chemical Engineering*, 9(3):257–267, 1985.

[10] L. T. Biegler and R. R. Hughes. Infeasible path optimization with sequential modular simulators. *AIChE Journal*, 28(6):994–1002, 1982.

[11] J. F. Boston and H. I. Britt. A radically different formulation and solution of the single-stage flash problem. *Computers & Chemical Engineering*, 2(2):109–122, 1978.

[12] H-S. Chen and M. A. Stadtherr. A simultaneous-modular approach to process flowsheeting and optimization. part I: Theory and implementation. *AICHE Journal*, 31(11):1843–1856, 1985.

[13] F. H. Clarke. *Optimization and nonsmooth analysis*, volume 5. SIAM, 1990.

[14] C. A. Dorao. Fundamentals of natural gas processing. Lecture slides in TEP08 – Gas processing and LNG, Department of Energy and Process Engineering, Norwegian University of Science and Technology, 2015.

[15] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[16] I. E. Grossmann, H. Yeomans, and Z. Kravanja. A rigorous disjunctive optimization model for simultaneous flowsheet optimization and heat integration. *Computers & Chemical Engineering*, 22:S157–S164, 1998.

[17] M. M. F. Hasan, I. A. Karimi, H. Alfadala, and H. Grootjans. Modeling and simulation of main cryogenic heat exchanger in a base-load liquefied natural gas plant. *Computer Aided Chemical Engineering*, 24:219–224, 2007.

[18] M. M. F. Hasan, I. A. Karimi, H. Alfadala, and H. Grootjans. Operational modeling of multistream heat exchangers with phase changes. *AIChE Journal*, 55 (1):150–171, 2009.

[19] J-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms II – Advanced Theory and Bundle Methods*, volume 306 of *Grundlehren der mathematischen Wissenschaften*. Springer, 1993.

[20] R. S. Kamath, L. T. Biegler, and I. E. Grossmann. Modeling multistream heat exchangers with and without phase changes for simultaneous optimization and heat integration. *AIChE Journal*, 58(1):190–204, 2012.

[21] K. A. Khan and P. I. Barton. Generalized derivatives for solutions of parametric ordinary differential equations with non-differentiable right-hand sides. *Journal of Optimization Theory and Applications*, 163(2):355–386, 2014.

[22] K. A. Khan and P. I. Barton. Generalized derivatives for hybrid systems. *Submitted*, 2015.

[23] K. A. Khan and P. I. Barton. A vector forward mode of automatic differentiation for generalized derivative evaluation. *Optimization Methods and Software*, 30(6):1185–1212, 2015.

[24] B. Koehret. *Conception d'un simulateur de procédés*. PhD thesis, Toulouse, INPT, 1987.

[25] Y. Nesterov. Lexicographic differentiation of nonsmooth functions. *Mathematical Programming*, 104(2-3):669–700, 2005.

[26] V. S. Parekh and P. M. Mathias. Efficient flash calculations for chemical process design–extension of the boston-britt "inside-out" flash algorithm to extreme conditions and new flash types. *Computers & Chemical engineering*, 22(10):1371–1380, 1998.

[27] J. Pettersen. LNG characteristics. Lecture slides in TEP4185 – Natural gas technology, Department of Energy and Process Engineering, Norwegian University of Science and Technology, 2014.

[28] J. Pettersen. Natural gas liquefaction process fundamentals. Lecture slides in TEP4185 – Natural gas technology, Department of Energy and Process Engineering, Norwegian University of Science and Technology, 2014.

[29] J. Pettersen. Natural Gas Liquefaction. Lecture slides in TEP08 – Gas processing and LNG, Department of Energy and Process Engineering, Norwegian University of Science and Technology, 2015.

[30] J. Pettersen. Recap from TEP4185 – LNG part. Lecture slides in TEP08 – Gas processing and LNG, Department of Energy and Process Engineering, Norwegian University of Science and Technology, 2015.

[31] L. Qi. Convergence analysis of some algorithms for solving nonsmooth equations. *Mathematics of Operations Research*, 18(1):227–244, 1993.

[32] L. Qi and J. Sun. A nonsmooth version of newton's method. *Mathematical Programming*, 58(1-3):353–367, 1993.

[33] H. H. Rachford Jr and J. D. Rice. Procedure for use of electronic digital computers in calculating flash vaporization hydrocarbon equilibrium. *Journal of Petroleum Technology*, 4(10):19–3, 1952.

[34] W. Rudin. *Principles of mathematical analysis*, volume 3. McGraw-Hill New York, 1964.

[35] A. M. Sahlodin, H. A. J. Watson, and P. I. Barton. Nonsmooth model for dynamic simulation of phase changes. *Submitted*, 2016.

[36] S. Scholtes. *Introduction to piecewise differentiable equations*. Springer Science & Business Media, 2012.

[37] J. M. Smith, H. Van Ness, and M. M. Abbott. *Introduction to Chemical Engineering Thermodynamics*. McGraw-Hill Education, New York, NY, 7th edition, 2005.

[38] T. H. Sweetser III. A minimal set-valued strong derivative for vector-valued lipschitz functions. *Journal of Optimization Theory and Applications*, 23(4):549–562, 1977.

[39] G. Venkatarathnam. *Cryogenic mixed refrigerant processes*. Springer, 2008.

[40] M. Vikse. Mixed-integer and nonconvex optimization. Project Thesis, Norwegian University of Science and Technology, Department of Energy and Process Engineering, December 2015.

[41] H. A. J. Watson and P. I. Barton. Modeling phase changes in multistream heat exchangers. *Submitted*, 2015.

[42] H. A. J. Watson, K. A. Khan, and P. I. Barton. Multistream heat exchanger modeling and design. *AIChE Journal*, 61(10):3390–3403, 2015.

[43] H. A. J. Watson, M. Vikse, D. Kim, T. Gundersen, and P. I. Barton. A nonsmooth inside-out algorithm for robust flash calculations. *In preparation*, 2016.

[44] D. Wolbert, X. Joulia, B. Koehret, and L. T. Biegler. Flowsheet optimization and optimal sensitivity analysis using analytical derivatives. *Computers & Chemical Engineering*, 18(11):1083–1095, 1994.

[45] D. Wolbert, X. Joulia, B. Koehret, and M. Pons. Analyse de sensibilité pour l'optimisation des procédés chimiques. *Récents Progrès en Génie des Procédés*, 5(3ième Congrès de Génie des Procédés, Compiègne, France):415–420, 1991.

[46] M. Yunt. *Nonsmooth dynamic optimization of systems with varying structure.* PhD thesis, Massachusetts Institute of Technology, 2011.

Appendix A

Residual function for the pQ-flash test problem in Chapter 5 This code evaluates the residuals in Lemma 5.1, and sets the step direction for the nonsmooth Newton solver.

```

void PQflash_sens(vector<ldouble> v, vector<ldouble> outputVar, vector<
    vector<ldouble>>& vect, vector<ldouble>& y)
{


    % hf, Q and p are input variables to the problem

    % composition z and the molar flowrate F are parameters in the problem

    % v: guess for column n, which is what is solved for in the
        sensitivity code.
    % outputVar: the flash outputs solved for in the VLE problem
    % vect: inputs

    % vect[0]: input variables
    % vect[1]: scalar parameters
    % vect[2]: composition z

    % outputVar[0]: temperature out
    % outputVar[1]: vapour fraction out

    % outputVar[2]: Hf
    % outputVar[3] = P


    int nc=5; % number of components


    int colA=outputVar.size();% columns of directions matrix A set to the
        same as the number of output variables solved for in the VLE
        problem.


    int depth=outputVar[0].getDepth();% Retrieves the length of the dot-
        value to the input/output variables. The length is set in the
        sensitivity code and contains the (k-1)th columns of M and N, the
        directions matrix A, as well as the column n solved for in the
        problem.


    %===========setting the dot values for the output variables==========%
    for(int i=0;i<colA;i++){
        outputVar[i].setDotValue(depth-colA-1, v[i].getValue()); % Sets
            the dot-value to the current guess v[i].
    }

    ldouble Hf, P, F, Q, Temp, alpha;

    % Input variables.
    Hf = vect[0][0];
    Q = vect[0][1];
    P = vect[0][2];

    % The parameters in the function make up the second column of vect.
    P=vect[1][0];
    F=vect[1][1];


    % The feed composition z makes up the third column:
    vector<ldouble> z(nc);
    for(int i=0;i<nc;i++){
        z[i]=vect[2][i];
    }


    %The output variables
    Temp=outputVar[0];
    alpha=outputVar[1];



    % Solve the flash equations:
    ldouble f1, f2; % These are the function values of the VLE equaitons

    idealFlash(nc, Temp, P, z, Hf, alpha, F, Q, f1, f2);

    % Since Temp and alpha are the solutions to temperature and vapour
        fractions for the flash problem, f1 and f2 will be approximately 0
        .

    %However, for v to be a solution, the nth column dot-values for f1 and
         f2 also need to be zero (see Chapter 5). I.e. they provide the
        residuals for the nonsmooth Newton solver in the sensitivity code:
    y[0].setValue(f1.getDotValue(depth-colA-1));
    y[1].setValue(f2.getDotValue(depth−colA−1));

    % The Newton solver also needs LD−derivatives of the dot−values to
        provide the step direction. This is done through the inner
        directions matrix A:
    y[0].setDepth(colA);
    y[1].setDepth(colA);

    % setting the LD−derivative of the computed dot−values v:
    for(int j=0;j<colA;j++){
        y[0].setDotValue(j,f1.getDotValue(depth−colA+j));% The LD−
            derivatives for the column v are the dot−values for f1 and f2
            provided by directions matrix A.
        y[1].setDotValue(j,f2.getDotValue(depth−colA+j));
    }

 return;
}
```

Appendix B

Derivations of Equation (2.26)

First, an expression for x_i is found from the component molar balance (Equation (2.10)):

$$x_i L + y_i V = z_i F, \tag{B.1}$$

which implies

$$x_i L + y_i (F - L) = z_i F, \tag{B.2}$$

so that

$$x_i L + K_r e^{\phi_i} x_i (F - L) = z_i F, \tag{B.3}$$

and hence

$$x_i = \frac{z_i F}{L + K_r e^{\phi_i} x_i (F - L)}. \tag{B.4}$$

Then, the definition of R is used for expressing the unknown liquid flowrate as a function of R, K_r and F:

$$R = \frac{K_r V}{K_r V + K_r^0 (F - V)}, \tag{B.5}$$

which gives

$$R K_r^0 L = K_r V (1 - R), \tag{B.6}$$

and

$$R K_r^0 L = K_r (F - L)(1 - R), \tag{B.7}$$

implying that

$$L = \frac{K_r F (1 - R)}{R K_r^0 + K_r (1 - R)}. \tag{B.8}$$

Substituting the expressions for x_i and L into (2.23) and rearranging yields Equation (2.26):

$$\begin{aligned}
r_i &= \frac{x_i L}{1-R}, && \text{(B.9)}\\
&= \frac{z_i F L \left(1-R\right)^{-1}}{L + K_r e^{\phi_i}\left(F-L\right)}, && \text{(B.10)}\\
&= \frac{z_i F \left(1-R\right)^{-1}}{1 + K_r e^{\phi_i}\left(\frac{F}{L}-1\right)}, && \text{(B.11)}\\
&= \frac{z_i F \left(1-R\right)^{-1}}{1 + K_r e^{\phi_i}\left(\frac{F(RK_r^0 + K_r(1-R))}{K_r F(1-R)}-1\right)}, && \text{(B.12)}\\
&= \frac{z_i F \left(1-R\right)^{-1}}{1 + K_r e^{\phi_i}\left(\frac{RK_r^0}{K_r(1-R)}\right)}, && \text{(B.13)}\\
&= \frac{z_i F}{1-R+K_r^0 R\mathrm{e}^{\phi_i}}. && \text{(B.14)}
\end{aligned}$$

Deriving Equations (2.27)-(2.30)

Before deriving the expressions for the liquid flowrate and molar liquid/vapour compositions, an alternate expression for K_r is needed:

$$\begin{aligned}
\sum_i x_i &= \sum_i y_i, && \text{(B.15)}\\
&= \sum_i K_r e^{\phi_i} x_i, && \text{(B.16)}
\end{aligned}$$

so that

$$\begin{aligned}
K_r &= \frac{\sum_i x_i}{\sum_i e^{\phi_i} x_i}, && \text{(B.17)}\\
&= \frac{\sum_i \left(1-R\right) r_i}{L}\left(\frac{\sum_i e^{\phi_i}\left(1-R\right) r_i}{L}\right)^{-1}, && \text{(B.18)}\\
&= \frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i}. && \text{(B.19)}
\end{aligned}$$

where the step from (B.17) to (B.18) is done by replacing x_i with r_i (Equation

(2.23)). Combining (B.19), (B.8) and (2.23), Equations (2.28)-(2.30) are found:

$$\begin{aligned} x_i &= \frac{(1-R)\,r_i}{L}, && \text{(B.20)} \\ &= (1-R)\,r_i \left(\frac{K_r F\,(1-R)}{RK_r^0 + K_r\,(1-R)} \right)^{-1} && \text{(B.21)} \\ &= \frac{r_i \left(RK_r^0 + K_r\,(1-R) \right)}{K_r F}, && \text{(B.22)} \\ &= r_i \left(RK_r^0 + \frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i}\,(1-R) \right) \left(\frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i} F \right)^{-1}. && \text{(B.23)} \end{aligned}$$

Recognizing that

$$F = \sum_i z_i F, \tag{B.24}$$

and that the product $z_i F$ is found from (2.26)

$$z_i F = r_i \left(1 - R + K_r^0 R e^{\phi_i} \right), \tag{B.25}$$

another expression for F is given by

$$\begin{aligned} F &= \sum_i r_i \left(1 - R + K_r^0 R e^{\phi_i} \right), && \text{(B.26)} \\ &= \sum_i r_i\,(1-R) + K_r^0 R \sum_i e^{\phi_i} r_i. && \text{(B.27)} \end{aligned}$$

Inserting for F in (B.23):

$$\begin{aligned} x_i &= \frac{r_i \left(RK_r^0 + \frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i}\,(1-R) \right)}{\frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i} \left(\sum_i r_i\,(1-R) + K_r^0 R \sum_i e^{\phi_i} r_i \right)}, && \text{(B.28)} \\ &= \frac{r_i \left(RK_r^0 + \frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i}\,(1-R) \right)}{\sum_i r_i \left(\frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i}\,(1-R) + K_r^0 R \right)}, && \text{(B.29)} \\ &= \frac{r_i}{\sum_i r_i}. && \text{(B.30)} \end{aligned}$$

The vapour fractions y_i are then found from Equations (B.19) and (B.30):

$$y_i = K_r e^{\phi_i} x_i, \tag{B.31}$$

$$= K_r e^{\phi_i} \frac{r_i}{\sum_i r_i}, \tag{B.32}$$

$$= \frac{\sum_i r_i}{\sum_i e^{\phi_i} r_i} \cdot e^{\phi_i} \frac{r_i}{\sum_i r_i}, \tag{B.33}$$

$$= \frac{e^{\phi_i} r_i}{\sum_i e^{\phi_i} r_i}. \tag{B.34}$$

Finally, the expression for the liquid flowrate is derived directly from the definition of the r_i variables:

$$L = \frac{(1-R)\, r_i}{x_i}, \tag{B.35}$$

$$= \frac{(1-R)\, r_i}{r_i / \sum_i r_i}, \tag{B.36}$$

$$= (1-R) \sum_i r_i. \tag{B.37}$$

Appendix C

Appendix C contains the code for the sequential-modular models used in Chapter 7. The MHEX model is not presented here as it was borrowed from Watson and Barton [41, 42] and not developed in this work.

The sequential-modular PRICO model:

```
% The PRICO function:

void PRICO(vector<ldouble> w, vector<ldouble> y, vector<vector<ldouble>>&
    vect_p, vector<ldouble>& resid){
    % w: tear variables, y: output variables from the MHEX, resid: tear
        variable residuals.
    %vect_p[0]: input variables to the MHEX, vect_p[1] model variables in
        the MHEX, vect_p[2]: parameters in the MHEX


    %w[0] is the tear variable for the stream temperature.
    %w[1] is the tear variable for the vapour fraction.


%=====================Feed enthalpy calculations=====================%
    % Ideal Compressibility factors
    ldouble Zl, Zv;
    Zl = 1.0;
    Zv = 1.0;


    int nc=5; %The number of components in the refrigerant mixture


    vector<ldouble> xliq(nc), yvap(nc), Psat(nc), K(nc), z(nc);
    %xliq: the liquid composition vector.
    %yvap: the vapour composition vector.
    %Psat: the vapour pressures for the different components.
    %K: the equilibrium constants for the components.
    %z: the feed composition.

    ldouble Hf; % the feed enthalpy to the end—flash module.


    vaporPressure(nc, w[0], Psat);%Function that evaluates the vapour
        pressures at the tear stream temperature.


    for(int i=0;i<nc;i++){
        z[i]=vect_p[2][6+i];%The feed composition is provided as
            parameters in the model.
    }

    %Raoult's Law
    for(int i=0; i<nc; i++) {
        K[i] = Psat[i]/vect_p[2][2]; %vect_p[2][2] is the pressure of the
            high pressure refrigerant in the model, and is providded as a
            parameter.
    }

    % Compositions and Rachford—Rice expression
    for(int i=0; i<nc; i++) {
        xliq[i] = z[i]/(1+x[1]*(K[i]—1));
        yvap[i] = K[i]*xliq[i];
    }

    Hf=w[1]*vaporEnthalpy(nc,yvap, w[0], vect_p[2][2], Zv) + (1—w[1])*
        liquidEnthalpy(nc,xliq,w[0],vect_p[2][2], Zl); %calculates the
        feed enthalpy to the end—flash module.


%========================End—flash module==========================%

    vector<ldouble> Output(2);% Outputs from the end—flash module.

    Output[0]=vect_p[0][2];% Initial guess for the LP refrigerant inlet
        temperature. Set equal to the temperature from the previous
        iteration.
    Output[1]=vect_p[0][3];% Initial guess for the LP refrigerant inlet
        vapour fraction. Set equal to the vapour fraction from the
        previous iteration.

    //vect_p[2][1]=F_R, vect_p[2][3]=P_low

    EndFlash(Hf, vect_p[2][1], vect_p[2][3], z, inVar);%The module
        function for the throttling valve, which calculates new values for
         the temperature and vapour fraction at the low pressure
        refrigerant inlet.

    %vect_p[2][1] is the refrigerant flowrate.
    %vect_p[2][3] is the pressure of the low pressure refrigerant.

    %Update the input variables to the MHEX module.
    vect_p[0][2]=Output[0];
    vect_p[0][3]=Output[1];

%==========================MHEX module=============================%

    MHEXmodule(y, vect_p[0], vect_p[1], vect_p[2], Hf);

%========================Tear equations============================%

    resid[0]=w[0]−y[0];% Tear equation for the stream temperature.
    resid[1]=w[1]−y[1];% Tear equation for the vapour fraction.
    return;
}
```

The sequential-modular extended PRICO model:

```


void PRICOext(vector<ldouble> w, vector<vector<ldouble>>& vect_p, vector<
    ldouble>& resid){
    %w: tear variables
    %vect_p[0]: input variables to MHEX 1.
    %vect_p[1]: input variables to MHEX 2.
    %vect_p[2]: inner variables in MHEX 1 module.
    %vect_p[3]: inner variables in MHEX 2 module.
    %vect_p[4]: parameters in MHEX 1 module
    %vect_p[5]: parameters in MHEX 2 module.
    %resid: residuals


    % Ideal Compressibility factors
    ldouble Zl, Zv;
    Zl = 1.0;
    Zv = 1.0;


    int nc=5;
    vector<ldouble> xliq(nc), yvap(nc), Psat(nc), K(nc), z(nc),
        Outputs_MHEX1(5), Outputs_MHEX2(5);
    ldouble Hf, adiabatic;

    % xliq: liquid composition in the flash.
    % yvap: vapour composition in the flash
    % Psat: the vapour pressures for the different components.
    %K: the equilibrium constants for the components.
    %z: the feed composition.
    %Outputs_MHEX1/MHEX2: the output variables from the MHEX.


    % the refrigerant composition is given as a parameter, and is the same
        in both MHEX:
    for(int i=0;i<nc;i++){
        z[i]=vect_p[4][6+i];
    }


%============================MHEX 1============================%
    % This part solves the first MHEX module.

    %Low pressure input variables are represented by the tear variables:
    % Temperature in:
    vect_p[0][2] = w[0];

    %Vapour fraction in:
    vect_p[0][3] = w[1];

    % The MHEX module needs the enthalpy at the LP refrigerant inlet to be specified.
    % The enthalpy is found by doing a pQ flash at the inlet with Q=0 and at constant pressure.

    % First the component vapour pressures are calculated for the given temperature w[0]:
    vaporPressure(nc, w[0], Psat);

    % Equilibrium constants and liquid/vapour compositions are then calculated from Raoult's law and the Rachford–Rice formulation:
    % vect_p[4][3]==P_low
    for(int i=0; i<nc; i++) {
        K[i] = Psat[i]/vect_p[4][3];
    }


    % Compositions and Rachford–Rice expression
    for(int i=0; i<nc; i++) {
        xliq[i] = z[i]/(1+w[1]*(K[i]−1));//w[1]==alpha_low_in
        yvap[i] = K[i]*xliq[i];
    }

    % With calculated compositions we can find the enthalpy at the LP inlet.
    Hf=w[1]*vaporEnthalpy(nc,yvap, w[0], vect_p[4][3], Zv) + (1−w[1])*liquidEnthalpy(nc,xliq,w[0],vect_p[4][3], Zl);

    % After the LP feed enthalpy has been found, the MHEX 1 module can now be solved.
    MHEXmodule(Outputs_MHEX1, vect_p[0], vect_p[2], vect_p[4], Hf);


    % The HP refrigerant temperature and vapour fraction out of MHEX 1 are input variables for MHEX 2 as we solve them sequentially:
    %HP refrigerant temperature in to MHEX 2:
    vect_p[1][0] = Outputs_MHEX1[0];
    %HP refrigerant vapour fraction in to MHEX 2:
    vect_p[1][1] = Outputs_MHEX1[1];

```

```
%=======================End flash module=======================%
    % This module solves the throttling valve as a pQ-flash problem:
    vector<ldouble> inVar(2);
    % inVar: the pQ-flash inner problem variables which are the LP
        refrigerant temperature
    % and vapour fraction into MHEX 2.

    %initialize the variables using calculated values from the previous
        iteration:
    % Initial guess for LP refrigerant temperature into MHEX 2:
    inVar[0]=vect_p[1][2];

    % Initial guess for LP refrigerant vapour fraction into MHEX 2:
    inVar[1]=vect_p[1][3];



    %Calculate the LP refrigerant inlet temperature for MHEX 2, using the
        same procedure as above:

    % For the second MHEX, there is isenthalpic throttling, which means
        that the LP refrigerant inlet enthalpy is the same as the HP
        refrigerant enthalpy out of MHEX 2.

    % W[2] is the HP temperature out of MHEX 2, which is a tear variable
        in the problem.
    vaporPressure(nc, w[2], Psat);

    % Raoult's Law
    for(int i=0; i<nc; i++) {
        K[i] = Psat[i]/vect_p[5][2];
    }
    % Here vect_p[5][2]==P_high, which is the pressure of the HP
        refrigerant


    % Compositions and Rachford-Rice expression:
    for(int i=0; i<nc; i++) {
        xliq[i] = z[i]/(1+w[3]*(K[i]-1));%w[3] is the vapour fraction of
            the HP refrigerant out of MHEX 2
        yvap[i] = K[i]*xliq[i];
    }

    % Calculate the feed enthalpy for the throttling valve/LP refrigerant:
    Hf=w[3]*vaporEnthalpy(nc,yvap, w[2], vect_p[5][2], Zv) + (1-w[3])*
        liquidEnthalpy(nc,xliq,w[2],vect_p[5][2], Zl);


    %Solve the throttling valve module:
    %vect_p[5][1]: Refrigerant flowrate,  vect_p[5][3]: the low pressure
    EndFlash(Hf, vect_p[5][1], vect_p[5][3], z, inVar);

    %The output variables for the throttling valve are then used as inputs
         to the second MHEX module:

    %LP refrigerant temperature in:
    vect_p[1][2]=inVar[0];

    %LP refrigerant vapour fraction in:
    vect_p[1][3]=inVar[1];



%============================MHEX 2============================%
    % solves the second MHEX module:

    MHEXmodule(Outputs_MHEX2, vect_p[1], vect_p[3], vect_p[5], Hf);


%========================Tear equations========================%

    % The residuals are now evaluated using the tear equations:

    %Tear stream 1: LP refrigerant stream between MHEX 1 and MHEX 2.
    %The LP refrigerant temperature tear variable:
    y[0]=w[0]-Outputs_MHEX2[2];%Tear equation Tin_low MHEX1 -- Tout_low
        MHEX2

    %The LP refrigerant vapour fraction tear variable:
    y[1]=w[1]-Outputs_MHEX2[3];%Tear equation alpha_low_in MHEX1 --
        alpha_low_out MHEX2


    %Tear stream 2: HP refrigerant stream out of MHEX 2.
    %The HP refrigerant temperature tear variable:
    y[2]=w[2]-Outputs_MHEX2[0];%Tear equation Tout_high into throttling
        valve -- Tout_high MHEX2

    %The HP refrigerant vapour fraction tear variable:
    y[3]=w[3]-Outputs_MHEX2[1];%Tear equation alpha_high_out into
        throttling valve -- alpha_high_out MHEX2

    return;
}
```

Appendix D

This chapter derives the Rachford-Rice reformulation used in the mid-function for flash calculations.

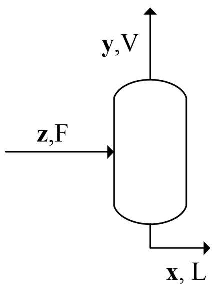


Figure D.1: A single stage equilibrium flash used for the VLE calculations in the MHEX model.

By combining the total material balance:

$$F = V + L, \tag{D.1}$$

with the component material balance, an expression for the vapour and liquid fractions can be found:

$$Fz_i = Vy_i + Lx_i, \tag{D.2}$$

which yields that

$$\begin{aligned} z_i &= \left(\frac{V}{F}\right) y_i + \left(\frac{L}{F}\right) x_i, && \text{(D.3)} \\ &= \left(\frac{V}{F}\right) y_i + \left(\frac{F-V}{F}\right) x_i. && \text{(D.4)} \end{aligned}$$

Furthermore, recognizing that the first fraction is equal to the vapour fraction α:

$$\alpha \equiv \frac{V}{F}, \tag{D.5}$$

(D.4) can be rewritten in terms of vapour fractions.

$$z_i = \alpha y_i + (1 - \alpha)\, x_i. \tag{D.6}$$

As equilibrium is assumed for each flash calculation, the following relationship between the molar fractions x_i and y_i exists:

$$y_i = K_i x_i, \tag{D.7}$$

where K_i is the equilibrium constant for component i. Combining Equations (D.6) and (D.7) will result in the following convenient expression for x_i:

$$z_i = \alpha K_i x_i + (1 - \alpha)\, x_i \tag{D.8}$$

hence,

$$\begin{aligned} x_i &= \frac{z_i}{\alpha K_i + 1 - \alpha}, && \text{(D.9)} \\ &= \frac{z_i}{1 + \alpha\,(K_i - \alpha)}. && \text{(D.10)} \end{aligned}$$

At each stage, equilibrium and the consitutive equation is assumed to hold:

$$\sum_i^n y_i - \sum_j^n x_j = 0, \tag{D.11}$$

i.e.,

$$\sum_i^n K_i x_i - \sum_j^n x_j = 0. \tag{D.12}$$

Substituting x_i in this last expression, will produce the formula used by the MHEX model for the flash calculations:

$$\sum_i^n \frac{K_i z_i}{1 + \alpha\,(K_i - 1)} - \sum_j^n \frac{z_j}{1 + \alpha\,(K_j - 1)} = 0, \tag{D.13}$$

which implies that

$$\sum_i \frac{z_i\,(K_i - 1)}{1 + \alpha\,(K_i - 1)} = 0. \tag{D.14}$$