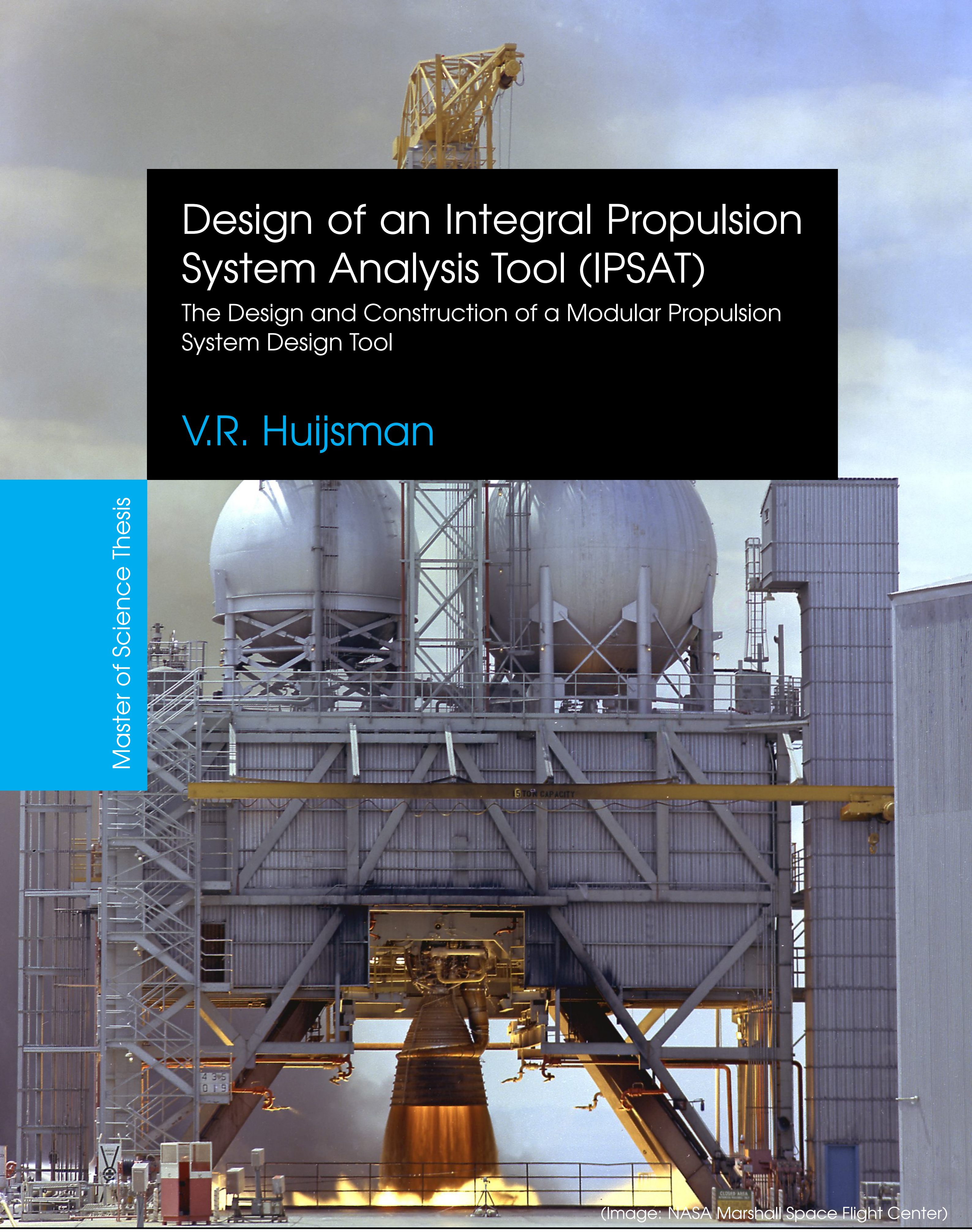


Design of an Integral Propulsion System Analysis Tool (IPSAT)

The Design and Construction of a Modular Propulsion System Design Tool

V.R. Huijsman

Master of Science Thesis

(Image: NASA Marshall Space Flight Center)

TU Delft Delft University of Technology

Space Systems Engineering

Design of an Integral Propulsion System Analysis Tool (IPSAT)

The Design and Construction of a Modular Propulsion System Design Tool

Master of Science Thesis

For the degree of Master of Science in Aerospace Engineering at Delft University of Technology

V.R. Huijsman

May 30, 2018

Faculty of Aerospace Engineering (AE) · Delft University of Technology

Table of Contents

List of Figures

List of Tables

Preface

At the start of the writing of this thesis, I have spend a total of seven years participating in the student rocket society called DARE. Here I have had many years of practical experience designing and building small rocket systems. I see my time at DARE as a very valuable practical addition to the theory focused study of Aerospace Engineering at the TU Delft. The work which I am going to present in this thesis would not have existed if I would not have been able to do my work at DARE. I would like dedicate this space to introduce my time at DARE and provide the context which forms the backdrop of the work presented in this thesis project.

My fascination and dedication towards developing rocket systems began when a group DARE members founded project DAWN in 2010. Project DAWN had the goal of developing the hybrid rocket propulsion system for the Stratos II rocket (a DARE sounding rocket planned to deliver a payload to 50 km). The project began with almost no pre-knowledge about hybrid propulsion systems and was executed by a group of undergraduate students. Despite this seemingly disadvantageous position, the team managed to build and fire the first hybrid rocket engine in the history of DARE within the first year of its founding.

In 2011 I participated in a minor organized by DARE and supervised by the TU Delft. Within this minor I initiated and oversaw the development of a basic hybrid rocket modeling tool. This tool was the first step in starting to understand the theory behind combustion dynamics and provided a crucial link between practice and theory. This project also sparked my interest in the development of modeling tools and has greatly contributed to work which is currently presented. The tool eventually would prove useful for predicting the performance of later hybrid motors.

Most of the people that participated in the minor continued their work in the development of the motor for Stratos II. This motor, named the DHX-200 Aurora, was a hybrid rocket motor running on liquid nitrous oxide as oxidizer and an exotic combination of sorbitol, paraffin and aluminium powder as fuel. This motor was, at that time, the largest and most powerful rocket motor ever developed by DARE.

Whilst the DHX-200 Aurora engine development was still ongoing I was also heavily involved with the technical design of the Stratos II rocket. Here I focused my efforts in solving the problems which plagued the Stratos II rocket in 2014. Finally, in October of 2015, the improved Stratos II rocket soared to a record altitude of 21.5 km in the south of spain. This launch was the accumulation of 5 years of dedication of many students. I am very grateful to all the other Stratos members to have been part of this inspiring project. With the results of this thesis project I would like to provide DARE with more powerful modeling abilities which will hopefully help the new generation of students in achieving their goals.

Acknowledgements

First and foremost I would like to take this opportunity to thank my parents and my sister for their unconditional support and love. This support was essential for the completion of this thesis project and it helped me greatly during the many difficult times which I faced during the completion of my study.

I would also like to thank my supervisor Angelo Cervone for giving me the opportunity and freedom to completely personalize the contents of this thesis project. I also appreciate his support in the guidance and evaluation of the work that is presented here.

My time at DARE has completely shaped my student life. I am very thankful for the many inspirational people which I have met during my time at DARE, especially the people with whom I have worked closely during the Stratos II project. Without those people this thesis would not have taken shape.

Special thanks goes to Johannes Ehlen, for his help in the error checking of this thesis report.

Delft, University of Technology V.R. Huijsman
May 30, 2018

Glossary

List of Acronyms

ASIC	Application Specific Integrated Circuit
AST	Angewandte System Technik Gruppe (Applied System Technology Group)
AUTOCOM	Automated Combustor Design Code
BWR	Benedict-Webb-Rubin equation of state
CEA	Chemical Equilibrium Analysis
CFD	Computational Fluid Dynamics
CICM	Coaxial Injection Combustion Model
DARE	Delft Aerospace Rocket Engineering
DAE	Differential-Algebraic Equations
DHX	Delft Hybrid Experimental
DOD	Department of Defense
EAI	Empresarios Agrupados Internacional (International Engineering Group)
EL	EcosimPro Language
EoS	Equation of State
ESA	European Space Agency
ESPSS	European Space Propulsion System Simulation
FEM	Finite Element Method
FVM	Finite Volume Method
GFSSP	Generalized Fluid System Simulation Program
GIM	Generalized Instability Model
GSTP	General Support Technology Programme

IPSAT	Integral Propulsion System Analysis Tool
LISP	Liquid Injector Spray Pattern
MATLAB	MATrix LABoratory
mBWR	modified Benedict-Webb-Rubin equation of state
MEOS	Multivariable Equation of State
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
ODE	Ordinary Differential Equations
PTFE	Polytetrafluorethylene (Teflon)
REFPROP	NIST Reference Fluid Thermodynamic and Transport Properties Database
ROCCID	Rocket Combustor Interactive Design
RPA	Rocket Propulsion Analysis
TNO	Toegepast Natuurwetenschappelijk Onderzoek (Applied Scientific Research)
TU Delft	Delft University of Technology
U.S.	United States
VTASC	Visual Thermofluid Dynamics Analyzer for Systems and Components

List of Symbols

Greek Symbols

α	$-$	Reduced Helmholtz energy
α^r	$-$	Residual component of the reduced Helmholtz energy
α^0	$-$	Ideal gas component of the reduced Helmholtz energy
Γ	m	Amplitude
δ	$-$	Reduced density (ρ/ρ_c)
δ°	$-$	Reduced density at reference conditions
ϵ	m	Surface roughness
ϵ/k	K	Lennard-Jones energy parameter
η	N·s·m^{-2}	Viscosity
η^0	N·s·m^{-2}	Dilute gas viscosity component
η^r	N·s·m^{-2}	Residual viscosity component
θ	rad	Angle
λ	W·m^{-1}·K^{-1}	Thermal conductivity
λ^0	W·m^{-1}·K^{-1}	Dilute gas thermal conductivity component
λ^c	W·m^{-1}·K^{-1}	Critical enhancement of the thermal conductivity component
λ^r	W·m^{-1}·K^{-1}	Residual thermal conductivity component
ξ	m	Correlation length
ξ_0	m	Amplitude

ρ	kg·m^{-3}	Density
ρ_c	kg·m^{-3}	Density at the critical point
$\rho_{l,sat}$	kg·m^{-3}	Density of the saturated liquid
$\rho_{v,sat}$	kg·m^{-3}	Density of the saturated vapour
σ	N·m^{-1}	Surface tension
σ_{LJ}	m	Lennard-Jones size parameter
τ	$-$	Reduced temperature (T_c/T)
τ°	$-$	Reduced temperature at reference conditions
Ω	$-$	Collision integral
ω	$-$	Acentric factor

Latin Symbols

A	m^2	Area
C	$-$	Constant, or fitting coefficient
c	m	Circumference
c_p	J·kg^{-1}·K^{-1}	Specific heat capacity at constant pressure
c_p°	J·kg^{-1}·K^{-1}	Specific heat capacity at constant pressure at reference conditions
c_v	J·kg^{-1}·K^{-1}	Specific heat capacity at constant volume
D	m	Diameter
D_{eff}	m	Effective diameter
D_h	m	Hydraulic diameter
E	J	Energy
$\dot{E}$	J·s^{-1}	Energy flow
F	N	Force
f	$-$	Darcy friction factor
H	J	Enthalpy
H°	J	Enthalpy at reference conditions
h	J·kg^{-1}	Specific enthalpy
K_f	kg^{-1}·m^{-1}	Fluid friction factor
k	$-$	Relaxation factor
L	m	Length
m	kg	Mass
$\dot{m}$	kg·s^{-1}	Massflow
M	kg·mol^{-1}	Molar mass
P	N·m^{-2}	Pressure
P_c	N·m^{-2}	Pressure at the critical point
P_m	N·m^{-2}	Pressure at the melting point
P_o	$-$	Poseuille number
P_{tp}	N·m^{-2}	Pressure at the triple point
P_v	N·m^{-2}	Vapour pressure
Q	J	Heat
$\dot{Q}$	J·s^{-1}	Heat flow
q_D	m	Maximum cutoff wave number
Re	$-$	Reynolds number
Re_{eff}	$-$	Effective Reynolds number
Re_h	$-$	Hydraulic equivalent Reynolds number
S	J·K^{-1}	Entropy
S°	J·K^{-1}	Entropy at reference conditions
s	J·K^{-1}·kg^{-1}	specific entropy
T	K	Temperature
T_c	K	Temperature at the critical point

T_m	K	Temperature at the melting line
T_{tp}	K	Temperature at the triple point
t	s	Time
U	J	Internal energy
u	J·kg^{-1}	Specific internal energy
v	m·s^{-1}	Velocity
V	m^3	Volume
W	J	Work
$\dot{W}$	J·s^{-1}	Work flow
w	m·s^{-1}	Speed of sound
z	–	Real gas compressibility factor

Physical Constants

ν	–	Critical exponent ≈ 5.07451·10^{-1}
g_0	m·s^{-2}	Gravitational acceleration at sea level ≈ 9.80665
k	J·K^{-1}	Boltzmann constant (R/N_A) ≈ 1.38065·10^{-23}
N_A	mol^{-1}	Avogadro constant ≈ 6.022140·10^{23}
R	J·K^{-1}mol^{-1}	Universal gas constant ≈ 8.31446
R_0	–	Amplitude ≈ 1.01

Mathematical Operators

$\overline{a}$	–	Mean value
$\|a\|$	–	Absolute value
$\|\|\mathbf{A}\|\|$	–	Matrix norm
$\mathbf{A^{-1}}$	–	Matrix inverse
$\mathbf{A}^\mathsf{T}$	–	Matrix transpose
∂	–	Partial derivative
Δ	–	Single value difference
$\boldsymbol{\Delta}$	–	Matrix difference
$\exp()$	–	exponent of e
f	–	Single function value
f'	–	Derivative of a function
$\mathbf{f}$	–	Function value vector
$\mathbf{J}$	–	Jacobian matrix
$\ln()$	–	logarithm with base e, or natural logarithm
$\log_x()$	–	logarithm with base x
$\sum$	–	Discrete summation
$\int$	–	Normal Integral
$\oint$	–	Volume integral
$\simeq$	–	Equivalence
x	–	Single variable value
$\dot{x}$	–	Derivative of a variable
$\mathbf{x}$	–	Variable value vector

Mathematical Constants

e	–	Exponential number ≈ 2.71828
π	–	Ratio of circumference over diameter of a circle ≈ 3.14159

Chapter 1

Introduction

In the early design phase of a propulsion system, the first step is to generate propulsion system designs which satisfy the top level system requirements. Most designs will then be eliminated by incorporating a trade-off scheme in order to select a best concept. In this phase of the design, the designer is faced with a difficult task. Analyzing the pros and cons of designs requires an in-depth analysis of the entire propulsion system. A good trade-off requires the designer to quantify how much benefit can be gained by changing the design in specific parts. Therefore, it is necessary to know how a design change in one part influences the overall performance of the propulsion system.

In order to address this problem, a number of design tools are available to the designer. There are two types of design tools. The first type is specifically aimed at a single aspect of rocket engine design. A few examples include RPA (1) and CEA (2) for equilibrium combustion performance, Coolprop (3) and REFPROP (4) for thermophysical properties of propellants, CICM (5) and LISP (6) for injector performance modeling, and ROCCID (7) and GIM (8) for combustion instability analysis.

The second type of tools are general propulsion system design tools. These tools allow the designer to investigate the propulsion system in its entirety. A few examples include AUTOCOM (9), GFSSP (10) and EcosimPro (11). These tools are a collection of specialized modules aimed at specific parts of the propulsion system design.

The aforementioned tools generally perform well, however they do have several downsides when it comes to the initial design phase of a propulsion system. A few of the main downsides include:

1. Many of the aforementioned tools are specialized to perform only one task. A design cycle often requires the use of a number of these tools, making quick design iterations difficult. Additionally, the input and output of most of the available tools is optimized for a specific type of analysis, intended by the designer of the tool. Different tools are optimized for different types of analyses, and therefore are often incompatible with each other. This makes it difficult to incorporate multiple design tools into a single environment without modification of the tools.
2. Most of the tools are proprietary, which restricts its use and/or restricts the modification of the tool. The modification of the tools is further complicated by the different programming languages and environments used to construct these tools,

which are often incompatible with each other. Additionally, these tools offer little to no explanation of their inner workings. This adds to the so called 'black box' user experience, where information is retrieved from the program but with little to no explanation about how it is produced.

3. many of the design tools do not mention the (complete) source of the data and/or the assumptions which are made. Therefore, a measure of the uncertainty of the outcome is difficult, if not impossible, to get. This lack of guidance not only adds to the 'black box' experience, in the worst case this creates a false sense of certainty and might lead to a misleading design.

The goal of this research project is to address these problems by creating an integral propulsion system analysis tool, which is to be named IPSAT in the remainder of this report. The aim of this tool is to create a platform which integrates different models in order to create a single design environment. The focus in this research project lies mainly in the construction of the platform. A minimal set of essential modules are constructed to demonstrate the capabilities. It is expected that future research projects will focus more on the addition of specific modules to increase the efficacy of the tool.

Delft Aerospace Rocket Engineering (DARE) is highly interested in such a design tool, due to its increased activity in propulsion system development. Most recently, DARE aims to develop a hybrid rocket which is capable of reaching 100 kilometers altitude. For this rocket, DARE is improving the existing DHX-200 Aurora hybrid rocket engine. The improvement process involves the analysis of many different design options. An analysis tool, which is able to analyze the impact of these options, is highly sought after. Additionally, such a tool will be useful for the early design phase of many other projects within DARE. An example of such a project is the design of a 10 kN LOx/CH_4 liquid rocket engine for use in future high altitude sounding rockets. DARE is seen as a major stakeholder in this research and has a big role in defining the requirements of the program.

The main goals and requirements of IPSAT are described in chapter 2. This chapter will also present the setup of this research project. The main concepts are introduced through a brief literature survey which is presented in chapter 3. The focus of the literature survey is mainly to investigate and map out different models and programs which are commonly used in industry. The IPSAT program and the computer program architecture is explained in chapter 4. The modules, which form the backbone of IPSAT, are explained and discussed in chapter 5. The verification procedures, which have been implemented for the different modules, are presented in chapter 6. The program is validated by using the DHX-200 Aurora test data gathered during the Stratos II development program, which is presented in chapter 7. The validation process is presented in chapter 8. Lastly, chapter 9 presents the main conclusions of this project and further recommendations for future projects.

Chapter 2

Project Setup

This chapter will introduce the thesis project and will explain the motivation behind the decisions made during the project. It is imperative that the project has a clearly defined structure and that its success can be measured against what is expected. The goal of this chapter is to provide a clear structure for the thesis and to derive a set of requirements which form the baseline of the project.

This chapter is divided into four sections. Section 2-1 will highlight the motivation behind this thesis project. Section 2-2 presents the research objective that defines the thesis project. A number of top level goals are derived from the problem statement and the research objective. These top level goals are presented in section 2-3. Finally, section 2-4 gives a full list of requirements for the program which are derived from the top level program goals.

2-1 Problem Statement

Delft Aerospace Rocket Engineering wants to partake in more ambitious projects. The main goal of this student society is to develop a rocket which is able to reach 100 km. In order to meet this ambitious goal, larger and more complex propulsion systems need to be developed. The design and development of a propulsion system currently makes up more than 40% of the total rocket development program budget (12). Most of this is spend on propulsion system testing needed for the characterization of the engine design. A tool that would help to reduce these engine characterization tests would be able to significantly reduce the cost for the development of the rocket. The development of computational tools are not a new phenomenon within DARE. Many tools have been developed in the past. However, none of these tools have been able to address all of the problems laid out below.

DARE currently works with many different specialized tools which deal with isolated parts of a propulsion system. These tools are well suited for addressing specific design questions (e.g. what is the combustion temperature of a oxygen - methane engine). However, the design of a complete propulsion system encompasses a multitude of these specific design problems, which are often inter related. The design needs can also change depending on the type of propulsion system (e.g. designing a liquid rocket requires different models and tools compared to designing a hybrid rocket). Furthermore, working with many different specialized tools can be very cumbersome when an existing design is iterated or, in the worst case, the design needs to be changed completely. Within DARE there is a need for a tool which is able to bring together many specialized tools into one environment where each part of the tool is able to perform a specific analysis.

The initial design of a propulsion system is generally not an issue. Ideal rocket theory is sufficient to correctly design for the main parameters. It is relatively straight forward to arrive at an initial design which meets the top level requirements. The issues arise after testing the propulsion system, when discrepancies are found between the predicted and the actual engine behavior. These problems are often contributed to complex feed system transients and rapidly changing thermo-physical properties of the propellant(s). It often takes many costly design iterations, which all need to be tested, to arrive at an acceptable engine design. There is a need for a tool that is able to encapsulate many of these complex phenomena for different types of engine configurations.

DARE is a student society where students join and leave in a period of only a few years. It is inevitable that much of the experience, gained during the timespan of a project, is lost when they leave after that project is finished. Every few years new students need to be able to design larger and more complex propulsion systems without the large expanse of knowledge that was obtained during the last project. This only worsens the aforementioned knowledge gap problem significantly. There is a need for tools which guide new designers when designing more complex propulsion systems. Although the knowledge gap problem is especially true for DARE, it also impacts many research institutes and companies.

It is customary within DARE to create custom computer models in order to analyze specific physical phenomena. These computer models often have a short lifetime, serve specific needs and are poorly documented. This leads to problems when new designers are tasked to use these computer programs or extend these programs to increase its functionality. In the worst case, the designer needs to spend time to create the exact same tool personalized to his/her preference. It is evident that time can be saved if a computer program is developed which is well documented and flexible by facilitating the designer to change and tweak the program in order to increase its functionality.

2-2 Research Objective

If DARE wants to develop rockets with ever increasing complexity and performance needs, it is inevitable that the problems mentioned in section 2-1 need to be addressed. It is the objective of this thesis project to address these problems by creating a general purpose fluid system design tool. This tool will combine a number of different modules to work together and provide the designer with an in-depth analysis of the propulsion system that is being designed. These objectives are captured best in the following sentence:

The aim of this thesis project is to develop a general purpose fluid system analysis tool for the design of a propulsion system by integrating various modules into one computer program.

The goal is to provide DARE with a fluid system design tool that has a large library of customizable modules. The inherent flexibility will enable the designer to design propulsion systems of varying types and in various stages of the design process. Additionally, the customizable modules will keep the program state-of-the art because designers can actively tweak the existing models and/or add new modules when required. Lastly, the standardized inputs and outputs of each module, and the structured breakup in modules, allows for a clear presentation of the capabilities of the program. This will increase the educational and academic potential of the design tool.

2-3 Project Goals

In order to meet the stated research objective and address the problems stated in section 2-1, a number of project goals have been formulated. The research objective is met if all the project goals are met. It is the aim of this thesis project to meet these goals and therefore complete the research objective. The project goals can therefore also be seen as a list of top level requirements for the computer program. The project goals are defined as follows:

1. A computer program shall be constructed which is modular.
2. A computer program shall be constructed which contains the basic modules for the analysis of a propulsion system.
3. A computer program shall be constructed which is transparent.
4. A computer program shall be constructed which provides flexibility to the designer.
5. The computer program shall be verified.
6. The computer program shall be validated.

The first project goal is derived from the need for a broader and more encompassing propulsion system design tool. Modularity plays an essential part in the relevance of the tool. Especially in an organization as DARE where the designer and developer base changes rapidly. As an academic tool, modularity should also provide customization to suit specific academic problems. A developer can create custom modules which suits his or her specific needs. The modularity of the computer program is specified as follows:

- Modularity on a system level. The goal is to create a program which is split in a number of modules where each module has its own defined function. These modules should be able to interact with each other through universal variables.
- Modularity on a module level. Each module can contain a number of different models. New models should be able to be integrated easily in the existing modules. This means that within each module the inputs and outputs need to be communicated in a structured way.

The second project goal is derived from the first goal and sets the minimum functionality of the program. It is important that this minimum functionality is sufficient to demonstrate the current and future capabilities of the program. Furthermore, the program should be able to demonstrate that the concept of linking specialized modules together can be used to analyze a larger system. The research goal is to develop a general purpose fluid system analysis tool, the following modules are selected to reflect that:

- An initialization module. This module should be able initialize the universal variables and prepare the inputs required for each module
- A solver module. This module should be able to solve a set of (non linear) equations.
- A fluid friction module. This module contains the methods that are used to evaluate feed system pressure losses due to fluid friction.
- A thermophysical module. This module should be able to accurately describe the thermophysical properties of different commonly used fluids in the relevant states (e.g. liquid, super critical).
- A conservation equation module. This module should be able to calculate the results of a number of fundamental conservation equations.

An in depth description of the exact functionality of these modules will be presented in the section 2-4. These modules define the first version of the program and the scope of this thesis project. It is expected that more modules shall be added in future projects.

The third project goal is derived from the need for a tool which is able to assist new designers in the design of complex fluid systems. This goal suggests the formatting of the communication with the designer. The computer program is intended to be used by students and academics. It is therefore deemed important to provide transparency to the designer. It is also an important factor in the legitimacy of the tool. If it is clear where the information comes from and how it is used in the system, it is much easier for a designer to correctly evaluate the results. The project goal is to implement transparency by the following means:

- The program shall provide insight into the models which are used. Models are an essential part of the program. It is important to inform the designer about the different models which are used to evaluate the problem and how they work.
- The program shall provide insight into the sources used in the evaluation of the problem. In many different parts of the program values are used which originate from a specific source. It is important to recognize this and to convey to the designer where this information can be found.
- Each part of the program code shall include documentation explaining the function it fulfills. In case the program is extended, it is important for the designer to understand the structure of the program code.

The fourth project goal is derived from the need for a tool which can cope with a large and varying user base. The goal will determine the way the designer will interact with the program. The project goal is to create a computer program which gives the designer a large amount of freedom of how to solve the problem. The flexibility of the computer program is defined as follows:

- Flexibility in operating the program. When interacting with the program interface, the designer shall be able to use and select all of the available functions of the program.
- Customization of the model inputs. When solving a problem, the designer should be able to customize the model inputs to suit the needs of the designer.

The fifth project goal is derived from the need of a functioning computer program. The verification process of a program guarantees that the program functions as designed. According to the U.S. Department of Defense (DOD) documentation of verification, validation and accreditation for models and simulations (13), the definition of the verification process for computer models is defined as:

> *"The process of determining that a model, simulation or federation of models and simulations implementations and their associated data accurately represents the developer's conceptual description and specifications."*

It is important for the designer to know that eventual discrepancies between the program results and the experiments are not caused by faults in the computer program implementation. The verification process is divided into the following two steps:

- Each module shall be verified individually. After the construction of each module, it is important that it is verified before implementing it into the computer program. It also guarantees that verification can be guaranteed even if new modules are added. The exact verifications steps that need to be taken depend on the module and the models used in each module.
- The program shall be verified as a whole. Once all the modules are implemented and verified, the last step is to verify the collaboration of the different modules.

The sixth project goal is derived from the need of a program that models reality. The validation process guarantees that the results of the program accurately predict what can be obtained from experiments. According to the U.S. DOD documentation of verification, validation and accreditation for models and simulations (13), the definition of the validation process for computer models is defined as:

> *"The process of determining the degree to which a model, simulation, or federation of models and simulations implementations and their associated data are accurate representations of the real world from the perspective of the intended use(s)."*

Even if the program does not fully predict reality, it is important to know what can and what cannot be modeled by the program and what kind of uncertainty can be expected from the program. The validation process requires relevant test data to be available. For this project the goal is to compare the results of the program with the following source of validation test data:

- The program shall be validated using the DHX-200 Aurora engine test data. The DHX-200 Aurora (14) is a hybrid engine developed by DARE between 2012 and 2015. Many different design configurations were tested during this time period and the available data can provide a valuable source of validation data.

2-4 Computer Program Requirements

The project goals, as presented in the previous section, denote the overall functionality of the program. These goals can be distilled further into more detailed requirements which will stipulate the exact steps required to arrive at the desired program. Tables 2-1 till 2-6 list the computer program requirements which are obtained from the project goals.

Table 2-1: Computer program requirements derived from the goal that the program shall be modular.

1		**The computer program shall be designed to be modular.**
1.1		The computer program shall be modular on a system level.
	1.1.1	The computer program shall be constructed out of distinct modules.
	1.1.2	The computer program modules shall be able to communicate to each other by universal variables.
	1.1.3	The computer program modules shall be able to be added to and removed from the computer program.
	1.1.4	The computer program modules shall be able to operate independently from each other.
1.2		The computer program shall be modular on a model level.
	1.2.1	Each module shall be able to contain different models.
	1.2.1	Each model in a specific module has the same in- and outputs.

Table 2-2: Computer program requirements derived from the goal that the program shall provide basic functionality.

2		**The computer program shall contain the basic modules for the analysis of a propulsion system.**
2.1		The computer program shall contain an initialization module.
	2.1.1	The initialization module shall be able to verify the designer inputs
	2.1.2	The initialization module shall be able to initialize the settings of other modules.
	2.1.3	The initialization module shall be able to provide the initial values required by the solver module.
2.1		The computer program shall contain a solver module.
	2.2.1	The solver module shall incorporate the fundamental fluid conservation equations.
	2.2.2	The solver module shall be able to successfully iterate towards a solution.
	2.2.3	The solver module shall be able to call other modules in order to update variables.
2.3		The computer program shall contain a thermophysics module.
	2.3.1	The thermophysics module shall be able to determine the thermophysical properties of a fluid in every fluid node.
	2.3.2	The thermophysics module shall be able to determine the following set of thermophysical variables: pressure, temperature, compressibility, enthalpy, internal energy, entropy, heat capacity at constant pressure, heat capacity at constant volume and speed of sound.
	2.3.3	The thermophysics module shall be able to determine the following set of transport properties: surface tension, viscosity and thermal conductivity.
	2.3.4	The thermophysics module shall be able to determine the properties listed in 2.3.2 and 2.3.3 of commonly used fluids in rocket engines.
	2.3.5	The thermophysics module shall be able to determine the properties listed in 2.3.2 and 2.3.3 of fluids in the following states: liquid, gaseous, vapour and super critical.
2.4		The computer program shall contain a fluid friction module.
	2.4.2	The fluid friction module shall be able to determine the fluid friction factor in every fluid branch.
	2.4.1	The fluid friction module shall be able to determine the friction factor for commonly used orifice shapes.

Table 2-3: Computer program requirements derived from the goal that the program shall be transparent.

3		**The computer program shall be designed to be transparent.**
3.1		The computer program shall provide insight into the models which are used.
	3.1.1	A list of available models for each module shall be available to the designer.
	3.1.2	The models which are used shall be presented in the output file.
3.2		The computer program shall provide insight into the sources used in the evaluation of the problem.
	3.2.1	The sources for the model constants shall be presented in the output file.
3.3		The computer program shall provide insight into the uncertainties figures where available.
	3.3.1	The uncertainty figures shall be incorporated in the model when available.
	3.3.2	The uncertainty figures for each model shall be presented in the output file when available.
3.4		The computer program code shall be able to be understandable to the designer.
	3.4.1	Each part of the program code shall include comments explaining the function it fulfills.

Table 2-4: Computer program requirements derived from the goal that the program shall be flexible.

4		**The computer program shall be designed to provide flexibility to the designer.**
4.1		The computer program shall provide flexibility when operating the program.
	4.1.1	The designer shall be able to select the modules of the program.
	4.1.2	The designer shall be able to select the models used in each module.
	4.1.3	The designer shall be able to construct a propulsion system in a systematic way.
4.2		The computer program shall provide customization of the model inputs.
	4.2.1	Model constants shall be able to be accessed and changed by the designer.

Table 2-5: Computer program requirements derived from the goal that the program shall be verified.

5		**The computer program shall be verified**
5.1		Each module shall be verified individually.
	5.1.1	Each individual module shall be verified for functionality.
5.2		The basic computer program shall be verified.
	5.2.1	The basic computer program shall be verified for functionality.

Table 2-6: Computer program requirements derived from the goal that the program shall be validated.

6		**The computer program shall be validated**
6.1		The computer program shall be validated using the DHX-200 Aurora test data.
	6.1.1	The computer program shall take a systematic diagram of the DHX-200 Aurora static test engine as input.
	6.1.2	The computer program shall take tank pressure data and combustion pressure data from the DHX-200 Aurora test as input.
	6.1.3	The computer program shall provide the fluid system response of the DHX-200 Aurora engine as output.
	6.1.4	The computer program output shall be compared to the original test data.

Chapter 3

Literature Survey

The first step in the development of a new design tool, is to investigate the different types of modeling techniques which are currently in use and/or have been used in the past. This investigation serves to give an overview of the state-of-the-art models and techniques.

The goal of this literature survey is to investigate a small number of design tools which are often used, or have been used in the past, in the development of propulsion systems. This chapter will map out what kind of models and techniques are convenient and relevant to use, while also trying to map out the most state-of-the art methods currently implemented. The survey will not provide an extensive overview of all methods and models. Instead, it will focus more on the currently popular and relevant models.

This chapter is divided into two main sections. The first section, 3-1, presents a number of general propulsion system design tools. These tools are similar to what IPSAT tries to achieve in that they contain different models for different design problems and they are intended to be used for the design of a complete propulsion system.

The second part of this chapter, which is covered by sections 3-2 to 3-3, presents a number of specialized tools intended to provide information regarding a specific part of a propulsion system design. These sections are ordered according to their field of application. Section 3-2 presents tools which are used to characterize chemical reactions. Section 3-3 presents tools which are used to characterize fluid properties.

3-1 General Propulsion System Design Tools

3-1-1 Automated Combustor Design Code

In the beginning of the 1970's the Aerophysics Research Corporation developed the Automated Combustor Design Code (AUTOCOM). The project was funded and supervised by NASA with the aim of developing a liquid rocket engine design code. Although the code mainly focuses on the design of the combustor, it encompasses many ideas of an overall engine design tool.

AUTOCOM optimizes a combustor design for a given set of characteristics. A design is given a rating based of the weighted sum of these characteristics. The code then tries to minimize this rating in order to find the design which best fulfills the given set of characteristics. The standard set of characteristics which are considered by AUTOCOM are:

1. Performance
2. Stability
3. Pressure Drop
4. Injector Complexity
5. Chamber Length
6. Chamber Diameter
7. Mixture Ratio

The values for each of these characteristics are calculated by making use of various models which are defined as different subroutines and can be customized by the designer. The program therefore successfully separates the optimization part from the detailed analysis part. This provides the designer with a general optimization platform which can be heavily customized.

An interesting feature provided by AUTOCOM is the option to analyze the combustion stability of a combustor design. There is no clear analytical method to analyze the combustion stability of a given combustor design. However, there are several empirical and semi-empirical methods, that have been developed in the past, which can provide information regarding the combustion stability of an engine for a given set of parameters. AUTOCOM provides the designer with the option to select a preferred method for optimization, as well as the ability to see the results from all the methods.

The program itself is written in Fortran and is not openly published. Detailed information about the program and the models used are well documented (9, 15) which makes it possible to find information about the different models and methods behind the AUTOCOM program. Figure 3-1 shows an example of a subroutine dedicated to the characterization of injector droplets.

Although the program is relatively old, it has some interesting concepts. The designer is able to select from a set of different models whilst maintaining an overarching solver. This feature increases the flexibility and applicability of the program. Additionally, the ability to compare the results from different models is a valuable mechanism to provide context and value to the results. The available documentation also provides a list of empirical methods to analyze the combustion instability phenomena. These methods can be used to provide the stability data of the combustor part of the program.

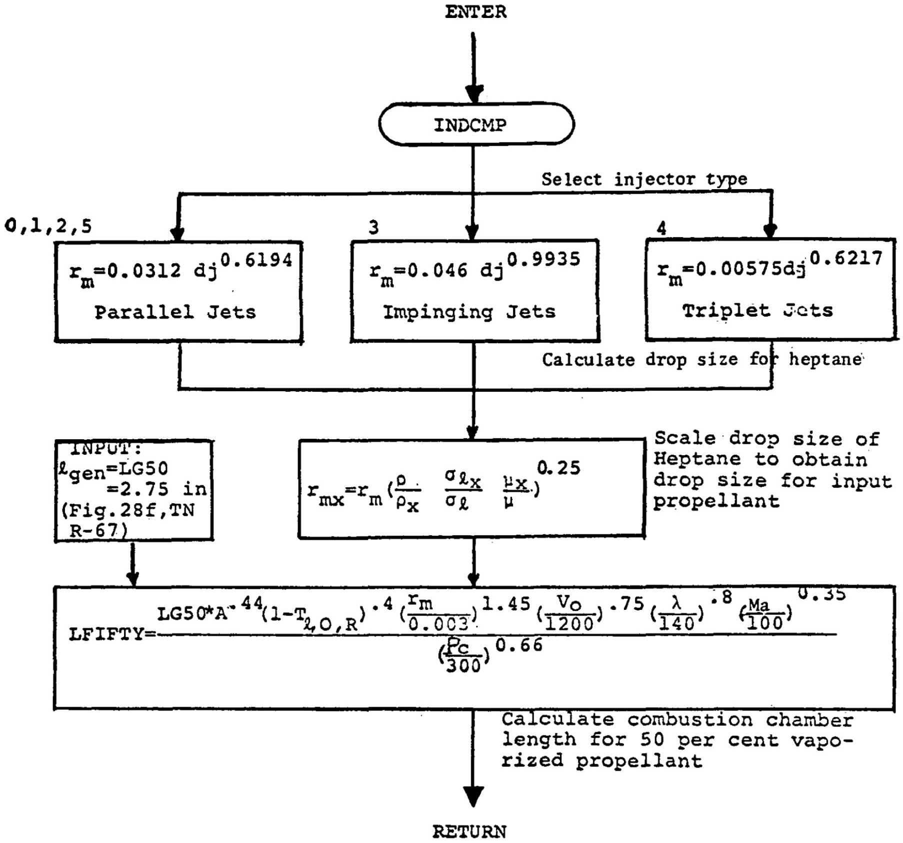


Figure 3-1: Example of the injector design optimization logic in AUTOCOM (15).

3-1-2 Generalized Fluid System Simulation Program

The Generalized Fluid System Simulation Program (GFSSP) is a program developed by Majumdar, van Hooser and colleagues at the Marshall Space Flight Center in the mid 1990's (16). The program was originally developed with the aim to analyze the complex fluid flow in rocket engine turbomachinery (16, 17, 18, 19). The program was later used in the analysis of general propulsion feed systems (10, 20, 21, 22). GFSSP was notably used in the development of the turbopump of the Fastrac engine (23), the Simplex turbopump (19) and the X-34 propulsion system (20, 24, 25). The program is also often used as a reference program for the verification of other, mostly in-house developed, programs (26). GFSSP still receives regular updates and as of 2016 the program has entered it 6th version (10).

GFSSP is a general purpose computer program which is able to analyze fluid systems in steady state and transient mode (10). The fluid system is discretized into nodes, branches, and conductors. The nodes act as the control volumes in a finite volume method solver, and the branches and conductors act as the interfaces between the volumes. The conservation equations for mass, fluid energy and species are solved in each node whilst the equation for conservation of momentum and conservation of energy in a solid is solved in each branch and conductor respectively.

To solve the set of equations GFSSP uses a combination of a Newton-Raphson method and a successive substitution method. The conservation of mass momentum and specie concentration are solved with a Newton-Raphson method whilst the conservation of fluid and solid energy are solved by with a successive substitution method. The idea behind

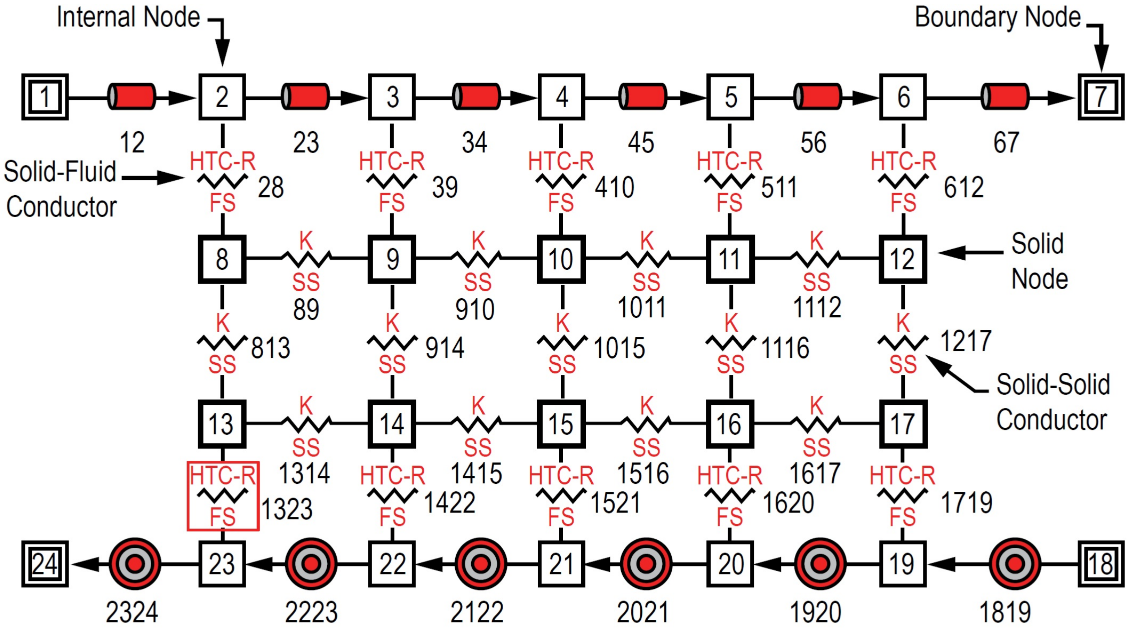


Figure 3-2: Example of a fluid system in GFSSP (10).

this split is that the strongly coupled equations are grouped together and solved by a high convergence solver (Newton-Raphson) whilst the weakly coupled equations are solved using a more slowly converging solver with increased stability (successive substitution) (10).

GFSSP is capable of modeling real fluids including phase changes, gas compressibility and mixture thermodynamics. The thermophysical properties of the fluids are retrieved by two thermodynamic property programs GASP/WASP and GASPAK (10). The program also allows the designer to provide tabulated fluid property data to add a custom fluid which does not exists in the provided library of either GASP/WASP or GASPAK.

In the more recent versions of GFSSP the visual thermofluid dynamics analyzer for systems and components, VTASC for short, has been introduced (10). VTASC adds a user interface on top of the existing program architecture similar to other general fluid system simulation programs. The user interface allows the designer to visually construct a fluid system by dragging and attaching different components to each other. This allows the designer to directly translate a visual feed system diagram into a GFSSP analysis case. VTASC will translate the visual construct into a GFSSP input file once an analysis is conducted (10). An example of a fluid system constructed in VTASC is shown in figure 3-2.

The technique used to evaluate the fluid system is intuitive and relatively user friendly. The technique also allows for modular additions of different analysis techniques. For example, a combustion chamber node, with its own analysis methods, can be added to the existing library of nodes without the need of changing the solver or solution methods. The implemented solving method is also relatively straightforward and can be modified for solving a large range of problems.

3-1-3 European Space Propulsion System Simulation

The European Space Propulsion System Simulation (ESPSS) is a module in the analysis software EcosimPro. ESPSS has been developed in 2008 by a joint European team in the frame of a General Support Technology Programme (GSTP), and is coordinated and funded by ESA. EcosimPro is a general purpose solver developed by Empresarios Agrupados Internacional (International Engineering Group) (EAI). EcosimPro is optimized to model dynamic systems which are represented by Differential-Algebraic Equations (DAE)'s, Ordinary Differential Equations (ODE)'s or discrete events. EAI has developed its own object oriented programming language called EcosimPro Language (EL). This language is custom tailored to be used for modeling discrete and continuous processes. The language can be used to simulate both steady state and transient problems (11).

ESPSS uses a list of libraries to model a complete propulsion system. The main libraries which are implemented by ESPSS are: a fluid properties library, a one dimensional fluid flow library, a combustion chamber library, a fluid tank library and a turbo machinery library. Each library contains models and functions which define the properties of the object.

The fluid properties library contains different types of options to calculate fluid properties. ESPSS can calculate fluid properties of perfect gases, where the state variables are temperature dependent and the transport and heat capacity properties are obtained from property tables (27). In the case of properties of gases, resulting from a combustion process, the transport and heat capacity properties are obtained through temperature dependent polynomials (11). Fluids can be modeled as either simplified fluids or real fluids. The properties of simplified fluids only depend on temperature and are obtained by interpolating real fluid property tables as function of temperature (11, 27). Real fluids are obtained by a search algorithm, which tries to find the fluid properties by searching for the right combination of two thermodynamic properties (11, 27).

The one dimensional fluid flow library in ESPSS works similarly to the fluid flow system of GFSSP. Feed system components are connected together to assemble a bigger fluid system. Figure 3-3 shows an example of such a feed system constructed in ESPSS. In ESPSS a feed system element is either capacitive or resistive (27). A capacitive element receives the flow variables as input and returns the state variables. A resistive element receives the state variables as input and returns the flow variables. Capacitive elements can only connect to resistive elements and vice versa (27). This method is comparable to the nodes and branches method of GFSSP. Every capacitive element solves the conservation of mass, energy and specie in differentiable form. Every resistive element solves the conservation on momentum in differentiable form (11).

The combustion chambers library contains a list of methods to simulate combustion chambers. ESPSS can calculate combustion at equilibrium conditions using the same method as is used in CEA, see section 3-2-1. The program is also able to simulate non-equilibrium combustion transients, using a time delay method. The combustion chambers library also contains a large number of combustion chamber periphery systems. These include the modeling of cooling jackets, pre-burners, nozzles and igniters. The combustion chambers library can be combined with components from the one dimensional fluid flow library and works similarly with respect to connecting different components to each other.

The fluid tanks library contains models for different types of fluid tanks, each with its own tank emptying mechanic for an in-depth tank transient analysis (27). The tanks can be linked to the one dimensional fluid flow library and are filled with fluids which are defined by the fluid properties library. ESPSS accommodates the modeling of heat transfer through the tank wall for different commonly used tank shapes. Tanks can also be customized by creating a custom number of nodes which increases the fidelity of the analysis. This also allows the designer to define a liquid and a gas region in the tank (27).

The turbo machinery library contains a list of turbo machinery components which can be linked to existing feed system components available in the one dimensional fluid flow library. The three main components in this library are pumps, compressors and turbines (11, 27). For all three components, performance is calculated by implementing

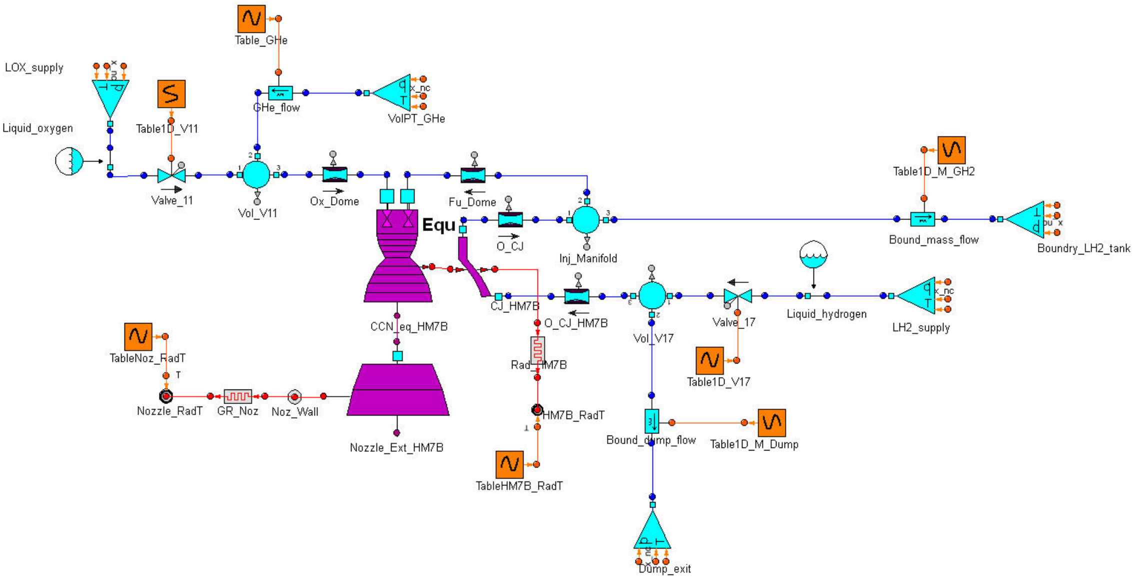


Figure 3-3: Example of a fluid system in ESPSS (28).

performance curves defined by the designer (11). These dimensionless performance curves allow the components to be used with all types of fluids. If these performance curves are not available, ESPSS also offers the ability to use generic performance curves which covers the entire operating range (11).

ESPSS provides a vast library of different components and simulation methods. Especially, the components and analysis methods of both the one dimensional fluid flow library and the combustion chambers library are state-of-the-art. ESPSS also comes with an extensive user interface and a simple drag and drop method to build a complete rocket propulsion system. A major downside is that the program is proprietary due to its integration in the general EcosimPro software. Furthermore, information regarding the inner workings of the program is scarce, and most information seems to come from external studies.

3-2 Chemical Reaction Analysis Tools

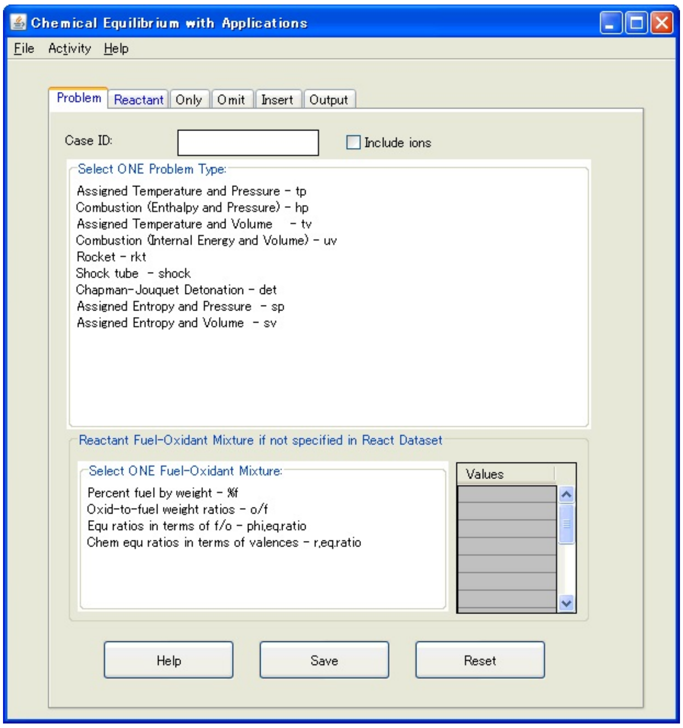
3-2-1 Chemical Equilibrium and Applications

Starting from the early 1950's, NASA's Lewis Research center has been developing computer programs and methods to calculate the chemical equilibrium and thermodynamic properties of equilibrium mixtures (2). Early research focused mainly on the development of a generalized procedure of calculating the equilibrium conditions of complex chemical reactions (29, 30, 31). In general, a chemical equilibrium for a given set of reactants and products can be calculated by solving for the conservation of mass, minimization of free energy, Dalton's law of partial pressure and either the conservation of enthalpy in case of adiabatic combustion or the conservation of entropy in case of isentropic expansion (29, 30, 31). From 1962 onwards, these methods were implemented in one chemical equilibrium calculation program which received regular updates (32, 33). In 1994 the latest version of the chemical equilibrium calculation program was created which was named the Chemical Equilibrium and Applications (CEA) (2, 34).

CEA can calculate the chemical equilibrium compositions by assigning two thermodynamic states. These states can be two of the following list of thermophysical properties:

temperature, pressure, density, internal energy, enthalpy and entropy. The program is also able to calculate Chapman-Jouget detonations and shock tube parameters for both incident and reflected shocks. Figure 3-4 shows the CEA interface where multiple problem types can be selected. The chemical equilibrium compositions are calculated by the minimization of Gibbs or Helmholtz energy depending on the selection of the two thermodynamic states (2). Next to calculating the composition of the products, CEA also calculates the transport properties of the products.

In order to obtain the equilibrium parameters, CEA consults a database of thermodynamic properties. These properties include the variation of heat capacity, enthalpy, entropy and Gibbs energy over temperature for a large number of species. CEA uses an interpolation function (2) which fits the tabulated data coming from a number of thermodynamic reference documents, such as the works of Mc Bride (35) and Chase (36).



Case ID:

□ Include ions

Select ONE Problem Type:

Assigned Temperature and Pressure - tp
Combustion (Enthalpy and Pressure) - hp
Assigned Temperature and Volume - tv
Combustion (Internal Energy and Volume) - uv
Rocket - rkt
Shock tube - shock
Chapman-Jouquet Detonation - det
Assigned Entropy and Pressure - sp
Assigned Entropy and Volume - sv

Reactant Fuel-Oxidant Mixture if not specified in React Dataset

Select ONE Fuel-Oxidant Mixture:

Percent fuel by weight - %f
Oxid-to-fuel weight ratios - o/f
Equ ratios in terms of f/o - phi,eq.ratio
Chem equ ratios in terms of valences - r,eq.ratio

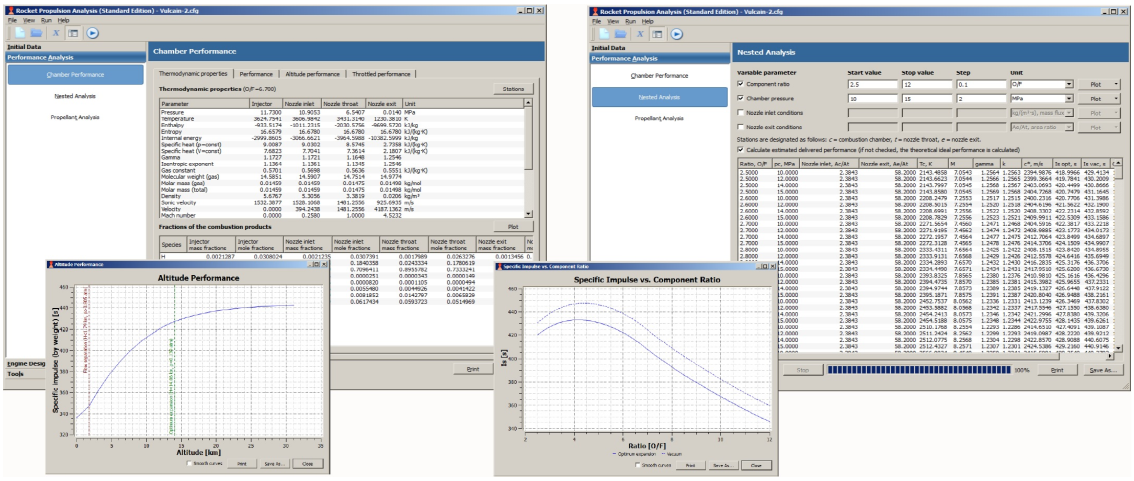
Figure 3-4: Example of the CEA program interface.

The calculation of chemical equilibrium forms the basis of characterizing any type of combustion. The techniques presented and utilized by CEA (2) are widely used in many types of problems which includes combustion. This makes these methods ideal for the implementation in a general propulsion system solver. The input (pressure, reactants concentration) and output data (species concentration, temperature, transport properties) allows for a smooth integration in a fluid system.

3-2-2 Rocket Propulsion Analysis

The Rocket Propulsion Analysis computer program (RPA), is a top level propulsion system design tool developed by Ponomarenko in 2010 (37). The initial program focused solely on solving of combustion equilibrium. In order to achieve this, RPA incorporates the same method of calculating chemical equilibrium as CEA. Later version started including more thrust chamber specific design tools. These tools include, combustion chamber cooling design (38), performance parameters estimation (39), rocket engine cycle performance analysis (40), and rocket engine mass estimations (41). As of 2015 RPA has become a versatile and robust top level propulsion system design tool capable of obtaining thermodynamic properties of combustion products, combustion chamber and nozzle design and sizing, combustion chamber thermal analysis, engine cycle analysis and propulsion system mass estimations (1, 40).

The combustion module of RPA uses the same methods as used in CEA. Combustion equilibrium properties are obtained by assigning two thermodynamic states and minimizing Gibbs energy (37). The combustion chamber and nozzle performance estimations are obtained through one dimensional thrust coefficient equation including estimations of the turbulent boundary layer (39, 40). RPA allows the designer to optimize the contour of the combustion chamber and nozzle for maximum thrust by optimizing this performance estimation.



Chamber Performance

Thermodynamic properties (O/F=6.700)

Parameter	Injector	Nozzle inlet	Nozzle throat	Nozzle exit	Unit
Pressure	11.7300	10.9053	6.5407	0.0140	MPa
Temperature	3624.7541	3606.9842	3431.3140	1230.3810	K
Enthalpy	-933.5174	-1011.2315	-2030.5756	-9699.5720	kJ/kg
Entropy	16.6579	16.6780	16.6780	16.6780	kJ/(kg·K)
Internal energy	-2999.8605	-3066.6621	-3964.5988	-10382.5999	kJ/kg
Specific heat (p=const)	9.0087	9.0302	8.5745	2.7358	kJ/(kg·K)
Specific heat (V=const)	7.6823	7.7041	7.3614	2.1807	kJ/(kg·K)
Gamma	1.1727	1.1721	1.1648	1.2546	
Isentropic exponent	1.1364	1.1361	1.1345	1.2546	
Gas constant	0.5701	0.5698	0.5636	0.5551	kJ/(kg·K)
Molecular weight (gas)	14.5851	14.5907	14.7514	14.9774	
Molar mass (gas)	0.01459	0.01459	0.01475	0.01498	kg/mol
Molar mass (total)	0.01459	0.01459	0.01475	0.01498	kg/mol
Density	5.6767	5.3056	3.3819	0.0206	kg/m³
Sonic velocity	1532.3877	1528.1068	1481.2556	925.6935	m/s
Velocity	0.0000	394.2438	1481.2556	4187.1362	m/s
Mach number	0.0000	0.2580	1.0000	4.5232	

Fractions of the combustion products

Species	Injector mass fractions	Injector mole fractions	Nozzle inlet mass fractions	Nozzle inlet mole fractions	Nozzle throat mass fractions	Nozzle throat mole fractions	Nozzle exit mass fractions	Nc mc
H	0.0021287	0.0308024	0.0021235	0.0307391	0.0017989	0.0263276	0.0013456	0.
				0.1840358	0.0243334	0.1780619		
				0.7096411	0.8955782	0.7333241		
				0.0000251	0.0000343	0.0000149		
				0.0000820	0.0001105	0.0000494		
				0.0055480	0.0044926	0.0041422		
				0.0081852	0.0142797	0.0065829		
				0.0617434	0.0593723	0.0514969		

Nested Analysis

Variable parameter	Start value	Stop value	Step	Unit
☑ Component ratio	2.5	12	0.1	O/F
☑ Chamber pressure	10	15	2	MPa
□ Nozzle inlet conditions				kg/(m²·s), mass flux
□ Nozzle exit conditions				Ae/At, area ratio

Stations are designated as follows: *c* = combustion chamber, *t* = nozzle throat, *e* = nozzle exit.

☑ Calculate estimated delivered performance (if not checked, the theoretical ideal performance is calculated)

Ratio, O/F	pc, MPa	Nozzle inlet, Ac/At	Nozzle exit, Ae/At	Tc, K	M	gamma	k	c*, m/s	Is opt, s	Is vac, s
2.5000	10.0000	2.3843	58.2000	2143.4858	7.0543	1.2564	1.2563	2394.9876	418.9966	429.4134
2.5000	12.0000	2.3843	58.2000	2143.6623	7.0544	1.2566	1.2565	2399.3664	419.7841	430.2009
2.5000	14.0000	2.3843	58.2000	2143.7997	7.0545	1.2568	1.2567	2403.0693	420.4499	430.8666
2.5000	15.0000	2.3843	58.2000	2143.8580	7.0545	1.2569	1.2568	2404.7268	420.7479	431.1645
2.6000	10.0000	2.3843	58.2000	2208.2479	7.2553	1.2517	1.2515	2400.2316	420.7706	431.3986
2.6000	12.0000	2.3843	58.2000	2208.5015	7.2554	1.2520	1.2518	2404.6196	421.5622	432.1900
2.6000	14.0000	2.3843	58.2000	2208.6991	7.2556	1.2522	1.2520	2408.3302	422.2314	432.8592
2.6000	15.0000	2.3843	58.2000	2208.7829	7.2556	1.2523	1.2521	2409.9911	422.5309	433.1586
2.7000	10.0000	2.3843	58.2000	2271.5654	7.4560	1.2471	1.2468	2404.5916	422.3817	433.2218
2.7000	12.0000	2.3843	58.2000	2271.9195	7.4562	1.2474	1.2472	2408.9885	423.1773	434.0173
2.7000	14.0000	2.3843	58.2000	2272.1957	7.4564	1.2477	1.2475	2412.7064	423.8499	434.6897
2.7000	15.0000	2.3843	58.2000	2272.3128	7.4565	1.2478	1.2476	2414.3706	424.1509	434.9907
2.8000	10.0000	2.3843	58.2000	2333.4311	7.6564	1.2425	1.2422	2408.1515	423.8420	434.8955
2.8000	12.0000	2.3843	58.2000	2333.9131	7.6568	1.2429	1.2426	2412.5578	424.6416	435.6949
	14.0000	2.3843	58.2000	2334.2893	7.6570	1.2432	1.2430	2416.2835	425.3176	436.3706
	15.0000	2.3843	58.2000	2334.4490	7.6571	1.2434	1.2431	2417.9510	425.6200	436.6730
	10.0000	2.3843	58.2000	2393.8325	7.8565	1.2380	1.2376	2410.9810	425.1616	436.4296
	12.0000	2.3843	58.2000	2394.4735	7.8570	1.2385	1.2381	2415.3982	425.9655	437.2331
	14.0000	2.3843	58.2000	2394.9744	7.8573	1.2389	1.2385	2419.1327	426.6448	437.9122
	15.0000	2.3843	58.2000	2395.1871	7.8575	1.2391	1.2387	2420.8040	426.9488	438.2161
	10.0000	2.3843	58.2000	2452.7537	8.0562	1.2336	1.2331	2413.1239	426.3469	437.8302
	12.0000	2.3843	58.2000	2453.5882	8.0568	1.2342	1.2337	2417.5546	427.1550	438.6380
	14.0000	2.3843	58.2000	2454.2413	8.0573	1.2346	1.2342	2421.2996	427.8380	439.3206
	15.0000	2.3843	58.2000	2454.5188	8.0575	1.2348	1.2344	2422.9755	428.1435	439.6261
	10.0000	2.3843	58.2000	2510.1768	8.2554	1.2293	1.2286	2414.6510	427.4091	439.1087
	12.0000	2.3843	58.2000	2511.2424	8.2562	1.2299	1.2293	2419.0987	428.2220	439.9212
	14.0000	2.3843	58.2000	2512.0775	8.2568	1.2304	1.2298	2422.8570	428.9088	440.6075
	15.0000	2.3843	58.2000	2512.4327	8.2571	1.2307	1.2301	2424.5386	429.2160	440.9146

Figure 3-5: Example of the combustion chamber performance analysis module in RPA (1).

The thermal analysis of the thrust chamber and nozzle is done via the implementation of a various number of emperical methods. The hot gas side heat transfer can be calculated using either Levlev's method or Bartz's method (38). The designer can select either of the two methods or the average of both. Equilibrium wall temperature and heat flux is calculated by constructing a heat balance. The designer can include convective and radiative heat fluxes in this balance. RPA can also evaluate cooling of thrust chambers by gas film cooling, liquid film cooling and regenerative cooling (38, 40). Figure 3-5 shows an example of the RPA chamber performance analysis module.

RPA is a state-of-the-art preliminary thrust chamber design tool. The combustion module allows calculation of combustion temperature, and species, by the use of an intuitive and minimal user interface. The results of this combustion module can be used by other modules within RPA for further analyses. The procedure of implementing different tools, to broaden the analysis, is a powerful method to increase the utility of the entire tool.

3-3 Thermophysical Property Tools

3-3-1 CoolProp

Coolprop is an open source fluid thermophysical property calculation tool. The tool was created in 2013 as a collaboration between the University of Liége and the Technical

University of Denmark (3). The aim of the tool is to create an open source thermophysical reference database using the most state-of-the-art methods. Coolprop excels at calculating the thermophysical properties of pure fluids, but lacks accuracy when dealing with fluid mixtures.

Coolprop uses a Helmholtz-energy-explicit formulation, using reduced temperature and density as input, to calculate the state properties (3). This method, also known as a multivariable equation of state, is state-of-the-art and builds on the work of Span (42). In the explicit Helmholtz formulation, a Helmholtz energy function is constructed, as function of two thermodynamic parameters, using a large set of empirical fitting parameters. The thermophysical properties are the derivatives of this Helmholtz function. More information regarding the Helmholtz formulation is presented in section 5-1.

The advantage of such a formulation is that it not only allows for the calculation of the associated pressure temperature or density, but it also provides other thermodynamic variables like c_p, c_v, h, s, etc. These state variables can be obtained by the differentiation of the Helmholtz energy function. Another advantage offered by the Helmholtz energy multivariable equation of state is that it gives accurate state properties in the critical region, whereas most other methods struggle in that region.

Transport properties have to be obtained by implementing more traditional methods depending on both the property and the fluid (3). The three main transport properties which are calculated are: viscosity, thermal conductivity and surface tension. Coolprop implements different types of formulations for these properties depending on the fluid. However, most of the formulations share a similar kind of structure to the Helmholtz energy formulation, where the property is depended on two state properties (e.g. reduced temperature and reduced density) and the property value is a linear combination of a dilute gas component, a residual component, and in some cases a critical enhancement component. More information regarding these techniques are presented in section 5-1.

The state-of-the-art techniques implemented and described by Coolprop (3) are well suited for a general fluid solver. An additional benefit of the models used by Coolprop is the accuracy of the results around the critical region. Fluids used in rocket engines often go through this region and it is therefore important to describe the fluid properties in this region well.

Coolprop openly publishes the sources and constants that are used in the thermophysical functions for each fluid on their website. The source papers also report the uncertainties that one can expect when using their formulation. Coolprop, however, does not implement these uncertainty figures in their tool and only reports the results of the state function.

3-3-2 Reference Fluid Properties

The NIST Reference Fluid Thermodynamic and Transport Properties Database (REFPROP) program is a fluid properties reference program developed by the National Institute of Standards and Technology (NIST). REFPROP can calculate the thermodynamic and transport properties of common industrial fluids and their mixtures (4). The first version of REFPROP was released in 1989 under the name Refrigerant Properties. This initial version, and versions that came out shortly after, were mainly focused on calculating the properties of refrigerants and refrigerant mixtures. from version 7 onwards, released in 2002, the program started to include other types of commonly used industrial fluids and the program was renamed to Reference Fluid Properties. Figure 3-6 shows an example of the REFPROP user interface.

REFPROP implements the following three methods for the calculation of the thermodynamic properties of fluids:

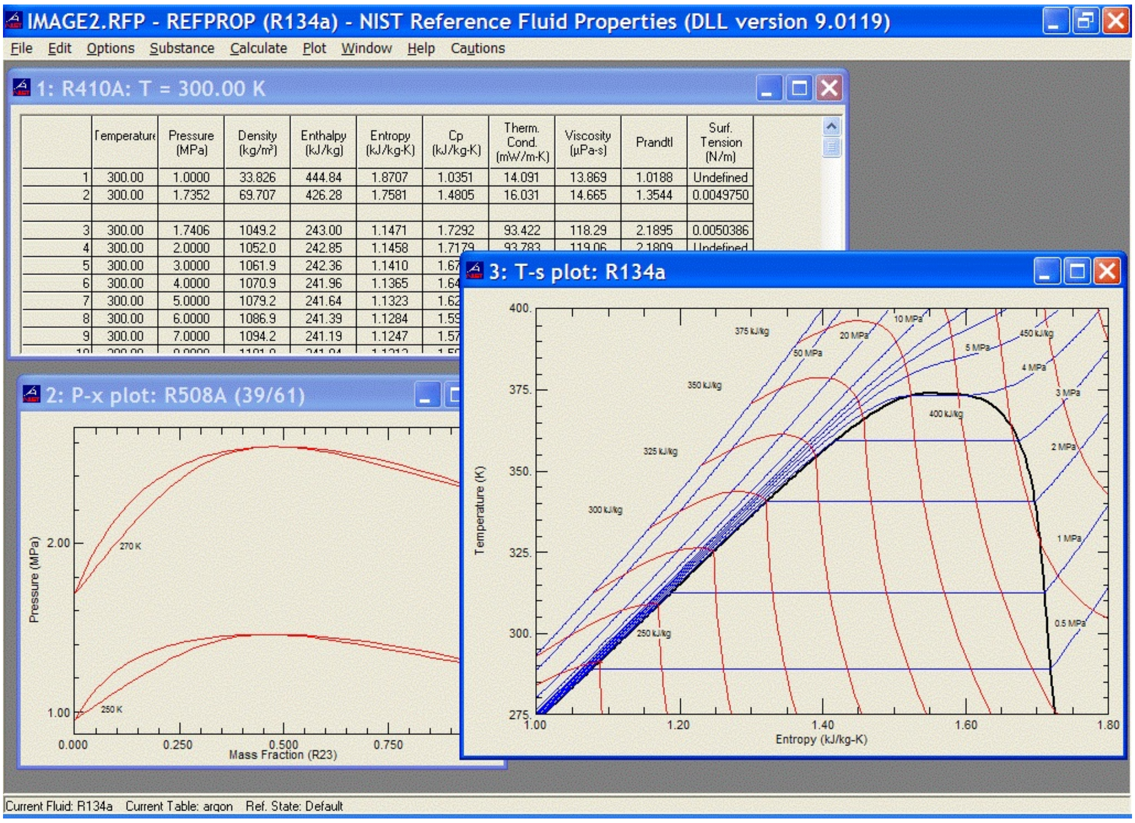


IMAGE2.RFP - REFPROP (R134a) - NIST Reference Fluid Properties (DLL version 9.0119)

1: R410A: T = 300.00 K

	Temperature	Pressure (MPa)	Density (kg/m³)	Enthalpy (kJ/kg)	Entropy (kJ/kg-K)	Cp (kJ/kg-K)	Therm. Cond. (mW/m-K)	Viscosity (μPa-s)	Prandtl	Surf. Tension (N/m)
1	300.00	1.0000	33.826	444.84	1.8707	1.0351	14.091	13.869	1.0188	Undefined
2	300.00	1.7352	69.707	426.28	1.7581	1.4805	16.031	14.665	1.3544	0.0049750
3	300.00	1.7406	1049.2	243.00	1.1471	1.7292	93.422	118.29	2.1895	0.0050386
4	300.00	2.0000	1052.0	242.85	1.1458	1.7179	93.783	119.06	2.1809	Undefined
5	300.00	3.0000	1061.9	242.36	1.1410	1.67				
6	300.00	4.0000	1070.9	241.96	1.1365	1.64				
7	300.00	5.0000	1079.2	241.64	1.1323	1.62				
8	300.00	6.0000	1086.9	241.39	1.1284	1.59				
9	300.00	7.0000	1094.2	241.19	1.1247	1.57				

2: P-x plot: R508A (39/61)

3: T-s plot: R134a

Current Fluid: R134a Current Table: argon Ref. State: Default

Figure 3-6: Example of theREFPROP program interface.

- Explicit Helmholtz energy formulation.
- Extended corresponding states model.
- Modified Benedict-Webb-Rubin equation of state.

The explicit Helmholtz energy formulation is a method which is also used in Coolprop, see section 3-3-1. A Helmholtz energy function is constructed which is a function of reduced temperature and reduced density. The thermodynamic properties are then derived from this Helmholtz function. The explicit helmholtz energy formulation is the most accurate multiparameter equation of state if correctly implemented (42).

The corresponding states model is derived from the van der Waals equation of state. The model assumes that any two dimensional equation of state in reduced form can be assumed to be a universal function independent of fluid. This means that if one is able to characterize one fluid in great detail, other fluids can be characterized without additional characterization. Real fluids deviate from this corresponding states model due to differences in molecular parameters compared to the ideal parameters assumed in the van der Waals model.

The extended corresponding states model improves the regular corresponding states model by including fluid specific terms. This means that an improvement in accuracy is traded for a loss in convenience. However, the extended corresponding states model still outperforms most other real fluid equations of state when it comes to accuracy and implementation convenience, especially when it comes to fluids which are close in composition.

The Benedict-Webb-Rubin equation of state (BWR) equation of state is one of the more simple real fluid equations of state. The original form of this real fluid equation of state was first proposed by Benedict, Webb and Rubin in 1940 (43). The BWR equation finds its roots in the Beattie-Bridgeman equation of state (43). Both forms share the property that the pressure is an explicit function of temperature and density. In the original form of the BWR equation of state, the fluid specific constants, used in the calculation of the pressure, are obtained by finding the fitting constants of several isometrics (43).

The modified Benedict-Webb-Rubin equation of state (mBWR), originally proposed by Jacobsen and Steward (44), adds to the original BWR equation of state by introducing more fitting constants. The original BWR equation of state had about 8 fitting constants (43) whereas the mBWR equation of state often contains around 32 fitting coefficients (45, 44). Other thermodynamic properties (e.g. entropy, enthalpy, specific heat at constant pressure, etc.) are obtained by integrating the mBWR equation of state in combination with the equation of the ideal gas heat capacity, the vapour pressure equation and the melting curve (44).

REFPROP represents the industry standard fluid property calculator. The methods used to calculate the properties of real fluids are state-of-the-art and the results are often used as reference data against which other tools are validated (42). Implementing these methods in a general fluid system solver will greatly increase their confidence level and utility.

3-4 Conclusion

This chapter provided a brief overview of the different available computer programs for the the design of propulsion system elements at different stages of the design process. The goal of this literature survey is to gather a number of common practices and find a good combination of design practices which can be implemented in the top level design of IPSAT. The following list of recommendations can be made with respect to the top level design of IPSAT:

- The fluid system design method as used by GFSSP and ESPSS blend in very well with a modular design tool. The construction of a fluid system by connecting separate components allows the designer to be free to construct a fluid system of arbitrary complexity. Each fluid system component can be part of a distinct module in the program.
- The approach of the AUTOCOM to the implementation of models gives the designer the freedom to customize the design approach. Each module can contain multiple models from which the designer can choose. This approach of implementing models creates a flexible program which is able to be easily extended.

Next to the general propulsion system design tools there are several examples of specialized state-of-the-art modeling tools which are suitable to be made into modules. The following list of recommendations can be made with respect module design of IPSAT:

- Both CEA and RPA uses a minimization of Gibbs free energy approach in order to calculate the chemical equilibrium of a set of reactions. This technique can be used to model a wide number of combustion related problems and should form the basis of a chemical reactions module.

- The approach to solving the set of conservation equations as implemented by GFSSP forms a good basis to approach the solving of a flexible fluid system. The usage of multiple solving methods simultaneously is also something that should be considered when constructing the solver module.
- Both REFPROP and Coolprop implement a state-of-the-art Helmholtz energy function to calculate the thermodynamic state properties of pure fluids for a large range of states. Since IPSAT operates across a number of fluid phases (e.g. liquid, gas, supercritical) it is recommended to implement such a Helmholtz energy function to calculate the main thermodynamic state variables.

Chapter 4

IPSAT Computer Program

The design of the program is guided by the program requirements set out in section 2-4 in combination with common practices found in literature, which was covered in chapter 3. The following project goals influence the design of the top level program architecture:

- The computer program shall be designed to be modular (requirements 1.1 - 1.2 in table 2-1).
- The computer program shall be designed to be transparent (requirements 3.1 - 3.4 in table 2-3).
- The computer program shall be designed to be flexible (requirements 4.1 - 4.2 in table 2-4).

This chapter shows how these requirements are incorporated into the IPSAT computer program architecture. Furthermore, this chapter describes the underlying philosophy and data structure of the IPSAT computer program and provides the reader with a guideline to interact with IPSAT.

The underlying fluid system model used by IPSAT is presented in section 4-3. Section 4-4 explains the definition of a module and presents universal variables which are exchanged between modules. The computer program architecture is explained in section 4-2. The data structure, which is used in IPSAT and the options for the designer to interact with the variables used in the computer program, are presented in section 4-5.

4-1 Program Requirement Implementation

The first step in the development of the computer program is to analyze the project goals as presented in section 2-3 and the derived program requirements as stated in section 2-4. The following section explains how the project goals and computer program requirements are implemented in the core structure of IPSAT.

4-1-1 Program Modularity

One of the main project goals is to create a program which is modular. This goal ties directly with the research objective. For the top level program architecture the following list of derived sub requirements are relevant:

- 1.1.1 The computer program shall be constructed out of distinct modules.
- 1.1.2 The computer program modules shall be able to communicate to each other by universal variables.
- 1.1.3 The computer program modules shall be able to be added to and removed from the computer program.
- 1.1.4 The computer program modules shall be able to operate independently from each other.

Requirement 1.1.1 is identified to be the most influential requirement with respect to the top level program design, because it directly influences the fundamental data structure within the program. In IPSAT a module is defined as a part of the program that fulfills a specific function. The program can call on internal subroutines in order to achieve the functional requirement. This is inspired by programs like AUTOCOM and GFSSP, see chapter 3. The advantage of using a modular architecture is that a complex program can be systematically structured and managed. In the current version of IPSAT this requirement is met by dividing the program into 5 modules. These modules are:

1. A Thermophysics module.
2. A Fluid friction module.
3. A Solver module.
4. A System initialization module.
5. A Fluid system initialization module.

These modules represent the basic functionality of a fluid system based on common practices found in literature in programs like GFSSP. These modules are also outlined in the program requirements listed in table 2-2. Section 4-4 describes the definition of the modules in IPSAT in more detail. Information regarding the modules themselves is presented in chapter 5.

Requirement 1.1.2 is essential to ensure that the modular architecture, as presented by requirement 1.1.1, is implemented effectively. It is also inextricably linked to requirement 1.1.3. In IPSAT information is passed using three universal data structures which are submitted to every module:

1. The **Nodes** data structure, containing the information of all the nodes in the fluid system.
2. The **Branches** data structure, containing the information of all the branches of the fluid system.
3. The **Settings** data structure, containing the program settings defined by the designer.

The fluid system definition is further explained in section 4-3. More information regarding the implementation and content of these universal data structures is presented in section 4-5.

Requirement 1.1.3 guarantees that the program can be extended by the designer and that existing modules can be replaced if needed. IPSAT facilitates this process by standardizing the module architecture. Each module has its own folder inside the program architecture. By using a standardized naming convention, new modules can be added in the correct folder and called via the settings without changing anything drastically to the program. More information regarding this process can be found in section 4-4.

Requirement 1.1.4 is aimed at simplifying the verification procedure. It ensures that each module can be verified separately. In the IPSAT program each module has its own internal folder. This creates an isolated environment where all the functions of the module are located. An internal module initialization procedure should be able to generate the required data fields in order to run the module independently. The only exception to this rule is the solver module. The solver module has the ability to call other modules and is therefore not able to function in isolation.

4-1-2 Program Transparency

The fourth project goal states that the program should be transparent to the designer and developer. For the top level program architecture the following list of sub requirements are relevant:

- 3.1.1 A list of available models for each module shall be available to the designer.
- 3.1.2 The models which are used shall be presented in the output file.
- 3.2.1 The sources for the model constants shall be presented in the output file.
- 3.3.1 The uncertainty figures shall be incorporated in the model when available.
- 3.3.2 The uncertainty figures for each model shall be presented in the output file when available.
- 3.4.1 Each part of the program code shall include comments explaining the function it fulfills.

Requirement 3.1.1 will be met by the implementation of a listing file in each of the module folders. This listing file will list the available models and methods for the given module.

Requirement 3.1.2 will be met twofold. First, the **Settings** data structure will provide a comprehensive overview of the modules used because it is the module selection input file for IPSAT itself. In order to provide an extra, easily to understand, overview, the program will return a summary file after completion displaying the modules and models which are used to solve the problem and additional information like the validity range of the data.

Requirement 3.2.1 ties in with requirement 3.1.2. This requirement ensures that the designer knows how the results are obtained and which sources are used. The problem summary file will present the source of the data (where applicable) next to the model which is used to solve the problem. This allows the designer to check if the model that is used to evaluate the problem is valid.

Requirement 3.3.1 and 3.3.2 help the designer interpret the results and the limitations of these results which are provided by the program. Requirement 3.3.1 will be met by the implementation of a uncertainty and validity range into the input data files for the modules which utilize source data (e.g. thermophysics module). Requirement 3.3.2 will be met by the implementation of a summary file after completion.

Requirement 3.4.1 will ensure that the designer is able to easily extend function files. Every piece of code used by IPSAT is documented in a structured way. Furthermore, each part of the program is constructed systematically which will help the designer in identifying the procedure that is described by the code.

4-1-3 Program Flexibility

The third project goal states that the program needs to be flexible. For the top level program architecture the following list of derived sub requirements are relevant:

- 4.1.1 The designer shall be able to select the modules of the program.
- 4.1.2 The designer shall be able to select the models used in each module.
- 4.1.3 The designer shall be able to construct a propulsion system in a systematic way.
- 4.2.1 Model constants shall be able to be accessed and changed by the designer.

Requirement 4.1.1 ties in with requirement 1.1.3. The designer should have control over the parts of the program that are most suitable for the problem at hand. The IPSAT program accommodates this by allowing the modules to be called by the designer in the program settings. Additionally, the order of operations can be altered by the designer in order to provide maximum flexibility over the program.

Requirement 4.1.2 extends requirement 4.1.1 by also allowing the designer to select the desired models and methods that are used in each module. Each module in IPSAT contains a function library of methods and/or models. All the functions in each function library share the same inputs and outputs. The available modules and/or methods can be selected by the designer in the **`Settings`** data structure.

Requirement 4.1.3 ensures that the designer shall be able to construct a wide array of fluid systems in the program. IPSAT implements a Finite Volume Method (FVM) functionality which allows control volumes to be connected to neighboring control volumes. Each control volume can have a different modeling requirement depending on the environment that it models. For example, a fluid tank be differently modeled compared to a combustion chamber. Each environment functionality is captured by a different module of the program.

Requirement 4.2.1 ensures that the model constants are freely accessible to the designer and not coded into the software itself. This makes the program more flexible and transparent to the designer. This requirement will be met by the implementation of model specific input files containing the model specific constants. These files are read and interpreted by the program during the module initialization subroutine.

4-2 Program architecture

The current version of IPSAT is constructed in the MATLAB programming environment. MATLAB offers a wide variety of build-in functions and has a large community base. MATLAB also has build-in tools for function management and complex data structures, both of which are extensively utilized in IPSAT. It is envisioned that future version will be written in an open source programming language to make distribution easier.

The program consists of a main interface offering several functions and a large number of modules, which are used by the program, but can be modified by the designer. The current version of IPSAT does not have a graphical user interface and it is a recommended feature for future versions.

The top level program architecture can be seen in figure 4-1. The first step is for the designer to define the system of nodes and branches using both the functions **`add_node()`** and **`add_branch()`**. These functions are used to input the relevant geometric information, the connectivity of the fluid system, and the module specific information (e.g. the two thermophysical state properties) for each node and branch of the fluid system.

After the fluid system layout is constructed, the program needs to initialize the program using the system initialization module. The initialization procedure will interpret the designer specified program settings and will initialize each of the selected modules. The system initialization module is described in more detail in section 5-5.

After the program is initialized the program starts the execution of the pre-solver modules. This program space can be used for modules which initialize the system before the solver routine is executed. An example of this is the fluid system initialization module described in further detail in section 5-4.

After the pre-solver modules are successfully executed, a solution will be obtained iteratively by the solver module. The solver module will call all in-solver modules which are selected by the designer, and which are part of the iteration scheme. These modules will be called during each iteration. In case of a transient problem, there will be two loops: an inner loop, which will iterate to find a solution at each time step, and an outer loop, which will iterate through every time step until the desired final time has been reached. The solver module is described in more detail in section 5-3. Examples of in-solver modules are the thermophysics module which is described in section 5-1, which calculates the thermophysical state properties of the fluid in each internal fluid node at each iteration, and the fluid friction module which is described in section 5-2, which calculates the the friction force in each of the fluid branches at each iteration.

If a solution is obtained, the program starts the execution of the post-solver modules. This program space can be used for modules which interpret the results, like a results visualization module or a output file construction module.

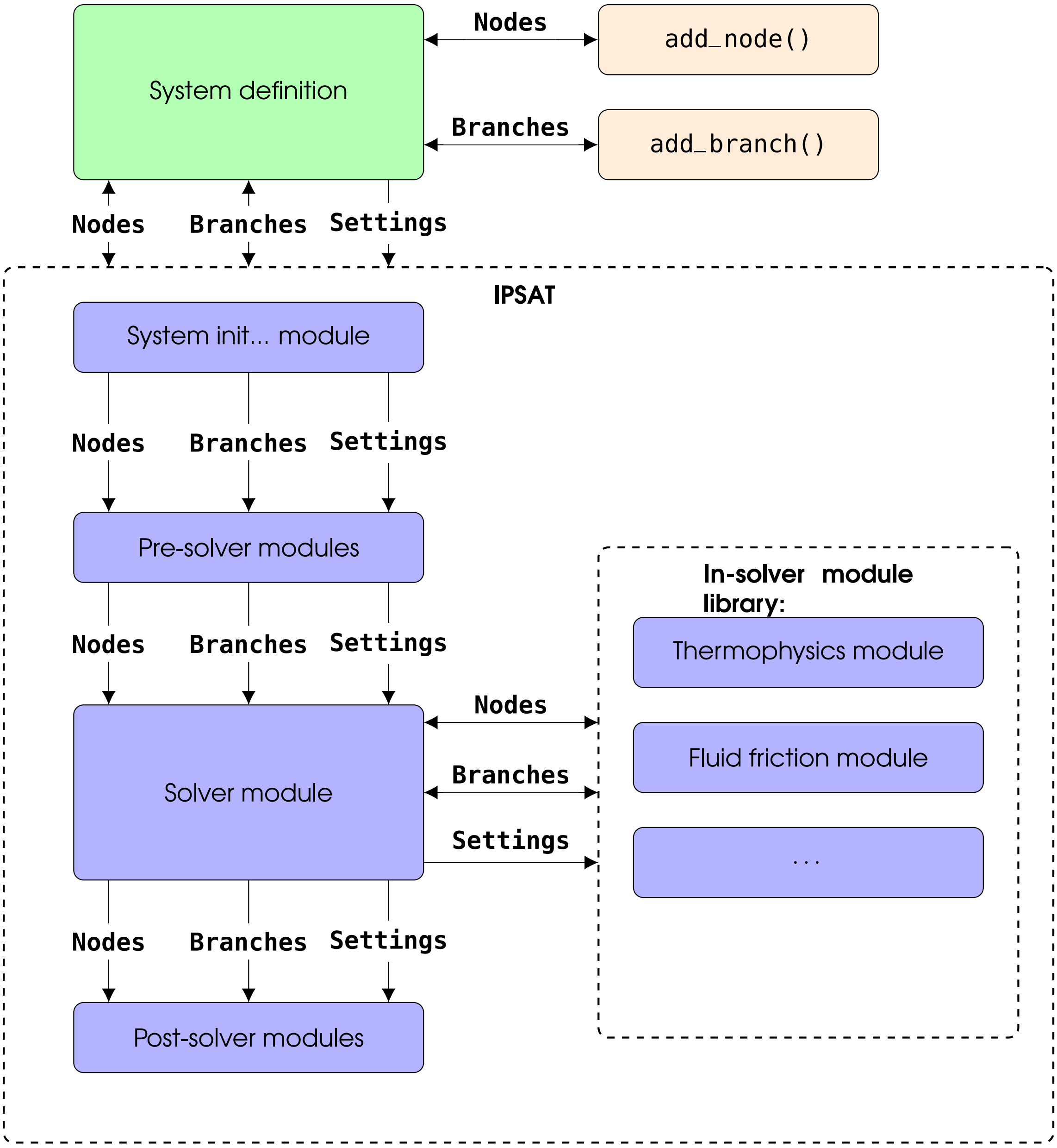


Figure 4-1: Top level program architecture.

4-3 Nodes and branches model

The underlying fluid system model as implemented in IPSAT is based around the nodes and branches method introduced by GFSSP, see section 3-1-2. This method allows for a flexible and intuitive way to build up a fluid system. The fluid nodes and branches method also provides the designer with a possibility of scaling the problem from a simple preliminary design stage to an arbitrarily complex fluid system in later stages of the design. The designer will be able to construct an arbitrarily complex fluid system by simply connecting nodes and branches.

The nodes and branches model is based around solving the fundamental conservation equations in a system containing a number of nodes of finite volume which are connected to each other by branches. Each node contains a fluid, or a mixture of fluids, with each its own thermophysical properties. Each branch utilizes these thermophysical properties to determine the flow properties between two nodes. Figure 4-2 shows the basic concept of two fluid nodes connected by a branch.

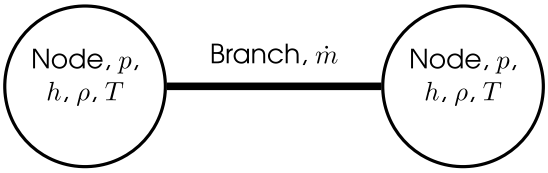


Figure 4-2: The two basic components of the IPSAT fluid system architecture.

Each node can be connected to an arbitrary number of neighboring nodes, making it more flexible compared to structured grid meshes most often used in CFD programs. The Nodes and Branches method allows the designer to map an arbitrarily complex fluid system one-to-one into a computer model which can be analyzed. Figure 4-3 shows the generalized concept of how a feed system can be constructed.

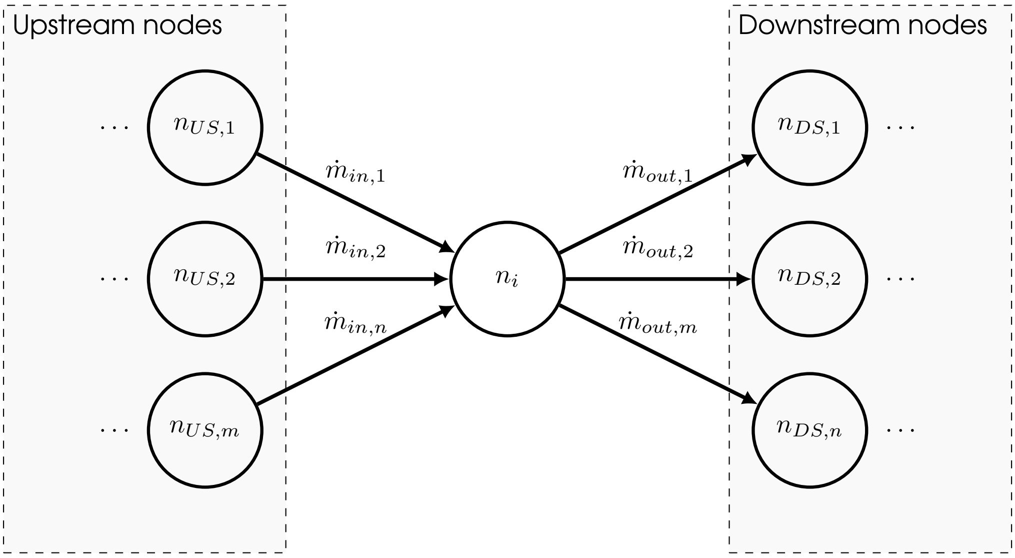


Figure 4-3: The general concept of a fluid node within the IPSAT fluid system architecture. Each node can have an unlimited number of neighboring upstream (US) and downstream (DS) nodes.

The Nodes and Branches method forms the basis of the IPSAT computer program. All modules are constructed around this fundamental concept.

4-4 Module Definition

IPSAT interacts with various modules, which are selected by the designer, in order to solve a system of equations. Each module can be seen as its own program with custom functions and custom variables. A developer can freely add functions to a module in order to enhance the functionality of the module or add new modules in order to enhance the functionality of the entire program.

The models and methods in each module are grouped in function libraries. The functions in each function library have a common input and output system and represent alternative options for the designer to achieve the same objective. For example, the solver module contains a root finding method library which contains a list of methods which all have the same objective, i.e. finding the root of a function. A function can be added to a function library by adhering to a simple naming convention. The first part of the function name is a custom name that can be chosen by the designer the last part of the function name is the selected library identifier. This means that a function file inside a function library adheres to the following naming convention: **`(custom name)_(library identfier).m`**.

Each module receives three inputs variables which are considered to be the three universal variables. These three universal variables are; **`Nodes`** which contains all the information about all nodes in the fluid system, **`Branches`** which contains all the information about all branches in the fluid system, and **`Settings`** which contains the program setting defined by the designer. The last variable can be used to transfer model specific settings which determine which module to run and which functions in a module should be used to analyze the problem. With these inputs, the module can change and/or add fields in the existing data structure. Each module will return the **`Nodes`** and **`Branches`** variables as output. A visual representation of this process can be seen in figure 4-4.

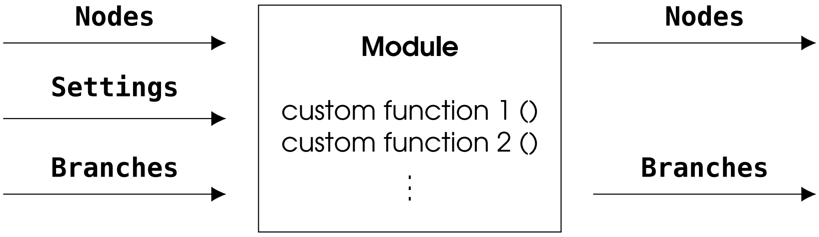


Figure 4-4: Definition of a module in IPSAT with the standardized inputs and outputs.

Modules can be executed in different parts of the program. This module hierarchy is captured in the module settings which are specified by the designer. The designer is free to change this hierarchy for existing modules if required. However, the module hierarchy is of particular use when creating new modules. The designer can create modules to be executed at specific location in the program. In the current version of IPSAT the following list of module types exist:

- **`system:`** Specifies that the module is fixed in the structure of the IPSAT program and cannot be changed.
- **`presolver:`** Modules of this type are executed before the solver, and is used for the initialization of certain variables before the solver subroutine is executed.
- **`postsolver:`** Modules of this type are executed after the solver, and is used for the refinement of data, visualization of data and any other output related actions.
- **`in_solver:`** Modules of this type are executed inside the solver, and will be called in each successive iteration of the solver and are required to update variables throughout the solver process.

4-5 Program data structure

The IPSAT program uses relational data structures in Matlab, called structs, to pass information. A struct is a high level variable which can store different types of data under field names specified by the designer. This allows for an intuitive method of conveniently passing large volumes of variables. The three main structs that need to be defined and initialized by the designer are **`Nodes`**, **`Branches`** and **`Settings`**.

The individual fields inside a struct are accessed by putting a dot (.) in between the field and the subfield. For example, **`A.B.C`** gives the value of variable **`C`** in subfield **`B`** in struct **`A`**. Sections 4-5-1, 4-5-2 and 4-5-3 describes the structure and all callable fields of the structs **`Nodes`**, **`Branches`** and **`Settings`** respectively.

4-5-1 Nodes

The information of all the nodes in the feed system are stored in the structure **`Nodes`**. The **`Nodes`** struct is a universal variable which is passed to all modules. Figure 4-5 shows the complete top level data structure for the nodes structure. Each feed system node has a unique identifier field name, or **`(ID)`**, in which data is stored for each node. A fluid system node can be added with the function **`add_node()`**.

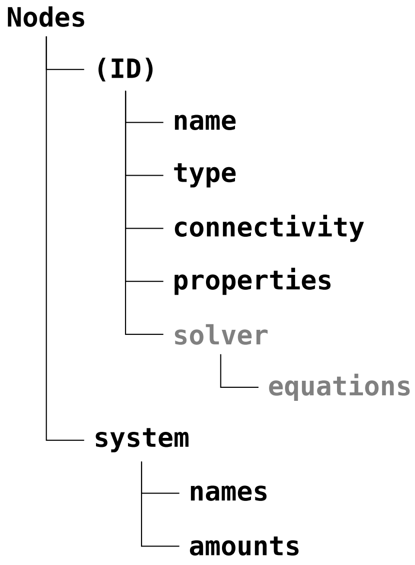


Figure 4-5: Node data structure, the gray text indicates that the data is only available for fluid nodes.

Node types

The field **`Node.(ID).type`** contains the type of node. In the current version of IPSAT there are four different types of nodes from which the designer can choose:

- **`internal`** An internal node is a node for which IPSAT will resolve the thermophysical properties. When a transient analysis option is selected, the program will require the designer to specify the initial conditions for an internal node. For the steady state analysis, IPSAT will guess a solution according to a fluid system initialization method. The designer is not required to specify an initial guess.

- **boundary** A boundary node is a node with specified conditions which do not change over time. The designer will have to specify the pressure, temperature and species.
- **solid** A solid node is a solid material which is connected to a fluid node. The designer must specify a material and its heat conductive properties.
- **ambient** An ambient node is a special kind of boundary node where the conditions are ambient conditions.

Node connectivity

The field **Node.(ID).connectivity** is a cell which contains a list of branch identifiers, **(ID)**, which are connected to the specific node. This list is not sorted in any way, however, each element in the list is unique.

Node properties

The field **Node.(ID).properties** contains the values of a list of properties which are resolved by IPSAT. Each property contains 1 entry in case of a steady state analysis or 2 entries in case of a transient analysis, where the first entry is the value at time t and the second entry is the value at time $t - \Delta t$. The properties which are resolved by the current version of IPSAT for each fluid node are:

- **pressure** (static)
- **delta**
- **vapor_pressure**
- **saturated_vapor_density**
- **viscosity** (dynamic)
- **density**
- **internal_energy** (specific)
- **enthalpy** (specific)
- **isobaric_heat_capacity** (specific)
- **dp_drho**
- **temperature** (static)
- **tau**
- **saturated_liquid_density**
- **surface_tension**
- **thermal_conductivity**
- **compressibility_factor**
- **entropy** (specific)
- **isochoric_heat_capacity** (specific)
- **speed_of_sound**
- **drho_dp**

Node solver

The field **Node.(ID).solver** contains the results of the solving process. this field contains one subfield **Node.(ID).solver.equations** which list the properties of each equation which is solved in the node. For each internal fluid node the conservation of fluid mass (**Node.(ID).solver.equations.fluid_mass**) and the conservation of fluid energy (**Node.(ID).solver.equations.fluid_energy**) is solved.

For each of these equations the residual is stored in the field **Nodes.(ID).solver.equations.(equation).residual** The unit of the residual is dependent on the equation and is equal to the units used in the respective equation. For example the residual of the fluid mass equation is kg·s^{-1}. Each equation is constructed out of a set of terms. The value of each term is stored in the field **Nodes.(ID).solver.equations.(equation).(term)**. The unit of each equation term is dependent on the equation and the same as the unit of the residual.

Node system

Node.system is a field which describes the quantities of the whole system. It contains two different fields: **names** and **amount**. The initialization function categorizes and creates this field during the fluid system initialization subroutine. These fields are therefore only available after the initialization of the system.

The field **names** contains the identifiers, **(ID)**, of nodes of a specific type. For example, the field **Nodes.system.names.internal** contains a list of identifiers of all the internal nodes in the system. The field **amounts** contains the number of nodes of a specific type. For example, the field **Nodes.system.amounts.internal** contains the number of internal nodes in the system.

4-5-2 Branches

The information of all the branches in the feed system is stored in the structure **Branches**. Figure 4-6 shows the top level data structure for the **Branches** structure. Each branch has a unique identifier field name, **(ID)**, in which data is stored for each branch.

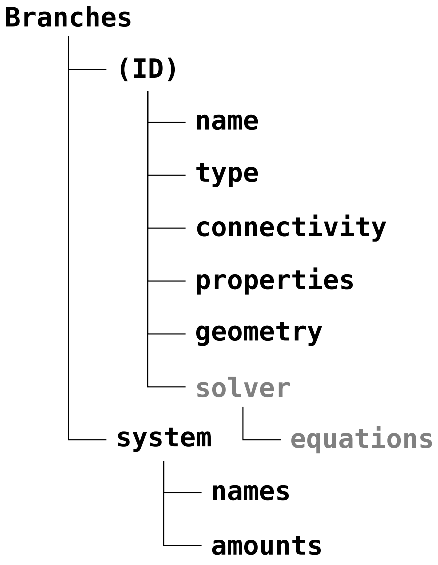


Figure 4-6: Branches data structure.

Branch types

The field **Branches.(ID).type** contains the type of branches. In the current version of IPSAT there are four different types of branch from which the designer can choose:

- **pipe**
- **restriction**
- **non_circular_duct**
- **thin_orifice**
- **thick_orifice**

Branch connectivity

The field **`Branches.(ID).connectivity`** is a cell which contains the identifiers, **`(ID)`**, of the two nodes which are connected by the branch. These node identifiers are not ordered in a particular way. The upstream or downstream node is determined based on the pressure in the two nodes which can change between solver iteration steps.

Branch properties

The field **`Branches.(ID).properties`** contains the values of a list of properties which are resolved by IPSAT for each branch. Similarly to the **`Nodes`** properties, each property contains 1 entry in case of a steady state analysis or 2 entries in case of a transient analysis where the first entry is the value at time t and the second entry is the value at time $t - \Delta t$. The current properties which are determined for each fluid branch are:

- **`massflow`**
- **`velocity`** (of the flow)
- **`reynolds_number`**
- **`delta_p`**

Branch geometry

The field **`Branches.(ID).geometry`** contains a number of geometric parameters that define the branch based on its type. It includes things like area, diameter, friction coefficient, etc. Not all of this information needs to be available for every branch, it depends on the branch type.

Branch solver

The field **`Branches.(ID).solver`** contains the results of the solving process. this field contains one subfield **`Branches.(ID).solver.equations`** which list the properties of each equation which is solved in the node. For each fluid branch the conservation of fluid momentum (**`Branches.(ID).solver.equations.fluid_momentum`**) is solved.

The residual is stored in the field **`Branches.(ID).solver.equations.(equation).residual`** Each equation is constructed out of a set of terms. The value of each term is stored in the field **`Branches.(ID).solver.equations.(equation).(term)`**.

Branch system

Similarly to the nodes structure, the struct **`Branches.system`** contains two different fields: **`names`** and **`amount`**. The field **`names`** contains the identifiers, **`(ID)`**, of branches of a specific type. For example, the field **`Branches.system.names.fluid_branch`** contains a list of identifiers of all the fluid branches in the system.

The field **`amounts`** contains the number of nodes of a specific type. For example, the field **`Branches.system.amounts.fluid_branch`** contains the number of fluid branches in the system.

4-5-3 Settings

All the data related to the settings of the computer program is stored in the struct **Settings**. Each subfield in the struct **Settings** contains information for a module in the computer program.

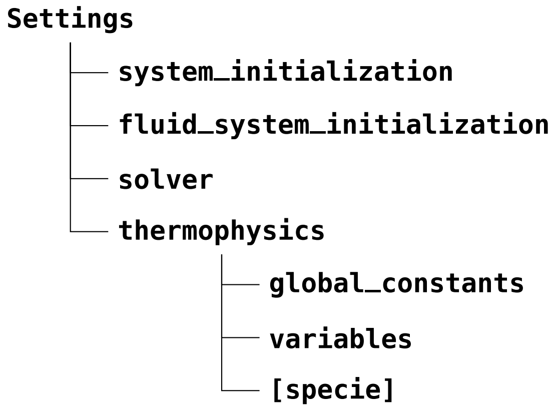


Figure 4-7: Settings data structure.

System initialization

The field **Settings.system_initialization** contains the settings for the system initialization module. This field contains three subfields: **settings**, **modules** and **initialization**. Figure 4-8 shows the data structure of the **Settings.system_initialization** struct.

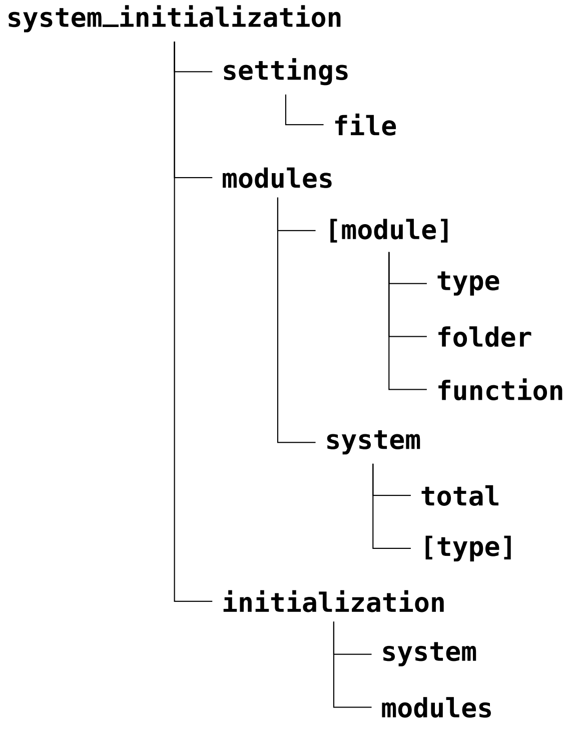


Figure 4-8: System initialization settings data structure.

The subfield **Settings.system_initialization.settings** contains the name of the custom settings file, unique to each analysis, which is stored in the settings folder.

The subfield **Settings.system_initialization.modules** contains the data of each module used in the analyis. Currently, there are 5 modules:

- **system_initialization_module**
- **fluid_system_initialization_module**
- **solver_module**
- **fluid_friction_module**
- **thermophysics_module**

For each module, three different subfields are available:

- **Settings.system_initialization.modules.(module).type**. The module type specifying in which part of the program the module is executed. Currently the following types of modules are available in IPSAT:
  - **fixed**, this module is fixed in the IPSAT in a pre-allocated location.
  - **pre-solver**, this module runs before the solver.
  - **in-solver**, this module runs inside the solver loop.
  - **post-solver**, this module runs after the solver.
- **Settings.system_initialization.modules.(module).folder**. The name of the folder containing the module functions.
- **Settings.system_initialization.modules.(module).function**. The function handle of the main module function.

The subfield **Settings.system_initialization.initialization** lists the initialization functions which need to be executed by the system initialization procedure. These initialization functions are grouped in two different function libraries:

- **Settings.system_initialization.initialization.system**. A cell containing the system initialization function names which are to be executed by the system initialization module.
- **Settings.system_initialization.initialization.modules**. A cell containing the list of modules which are to be initialized by the system initialization module.

Fluid system initialization

The field **Settings.fluid_system_initialization** contains the settings for the fluid system initialization module. This field contains two subfields: **method** and **function**. Figure 4-9 shows the data structure of the **Settings.fluid_system_initialization** struct.

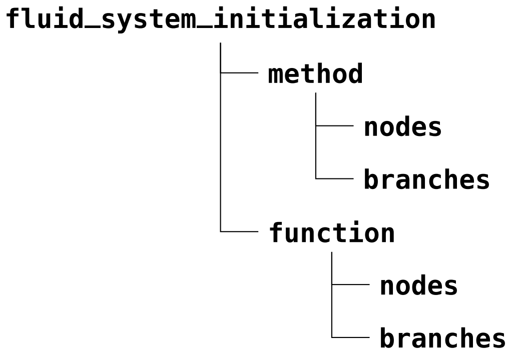


Figure 4-9: Fluid system initialization settings data structure.

The subfield **Settings.fluid_system_initialization.method** contains the initialization methods which are used to initialize the different elements of the fluid system.

The subfield **Settings.fluid_system_initialization.function** contains the function calls for the fluid system initialization methods which are used for each element of the fluid system.

Solver

The field **Settings.solver** contains six subfields, **mode**, **iteration**, **equations**, **variables**, **methods** and **system**. The first subfield, **Settings.solver_module.mode** specifies if the problem is a steady state problem or a transient problem by either specifying **steady_state** or **transient**.

The subfield **Settings.solver_module.iteration** contains the minimum and maximum number of iterations that the program needs to perform in the inner loop. These values are contained in the fields **Settings.solver_module.iteration.max** and **Settings.solver_module.iteration.min**.

The subfield **Settings.solver_module.equations** contains the settings of the selected list of coupled conservation equations which are to be solved by IPSAT for the specific analysis. The current list of available conservation equations is:

- **fluid_mass**
- **fluid_momentum**
- **fluid_energy**

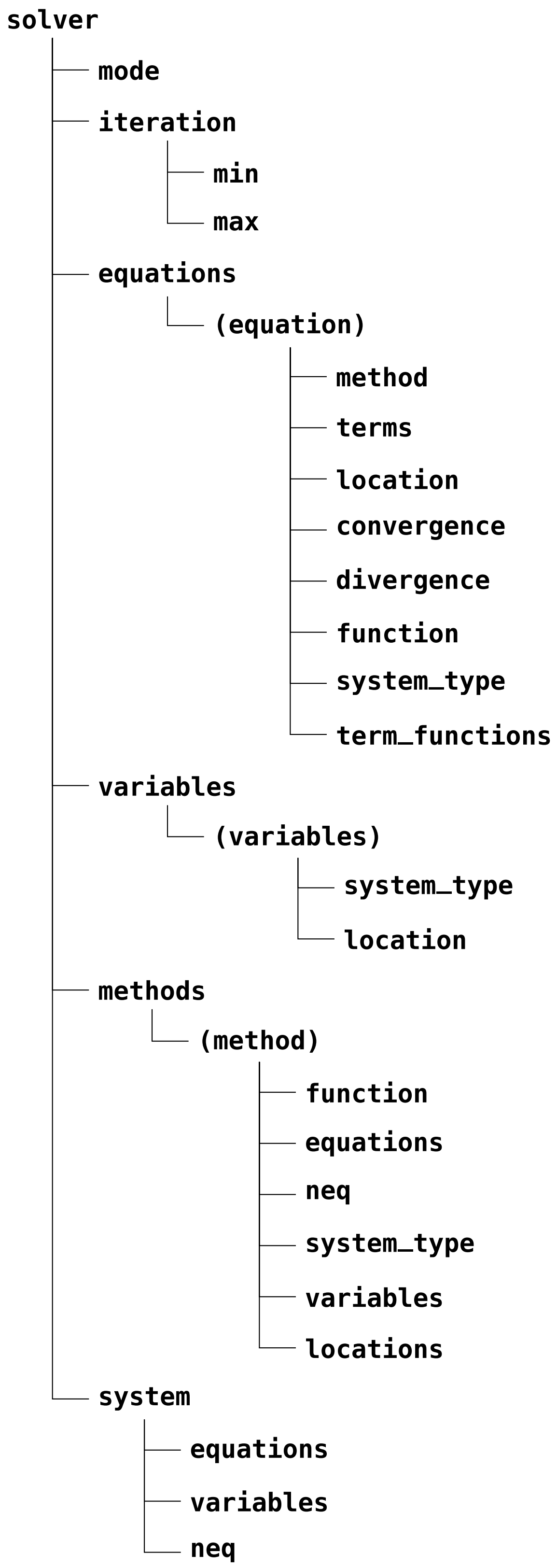


Figure 4-10: Solver module settings data structure.

For each equation, eight different subfields are available:

- **`Settings.solver.equations.(equation).method`**. The solving method that is used to solve this equation.
- **`Settings.solver.equations.(equation).terms`**. A cell containing the equation terms that are active for this equation.
- **`Settings.solver.equations.(equation).location`**. The location in the fluid system where this equation is solved.
- **`Settings.solver.equations.(equation).convergence`**. The convergence limit for the residual of this equation.
- **`Settings.solver.equations.(equation).divergence`**. The divergence limit for the residual of this equation.
- **`Settings.solver.equations.(equation).function`**. The function handle for this equation.
- **`Settings.solver.equations.(equation).system_type`**. The data structure which is associated with this equation.
- **`Settings.solver.equations.(equation).term_functions`**. A cell containing the function handles of all the equation terms.

The subfield **`Settings.solver_module.variables`** contains a list of fundamental variables from which the values are obtained by IPSAT by solving the aforementioned conservation equations. The current list of available variables is:

- **`pressure`**
- **`density`**
- **`massflow`**
- **`enthalpy`**

Note that these are not all the variables which are obtained by IPSAT. All other variables are obtained by consulting the various modules. For example, the fluid temperature can be obtained by the thermophysics module by entering the fluid pressure and fluid enthalpy. For each variable, two different subfields are available:

- **`Settings.solver.variables.(variable).location`**. The location in the fluid system where this variable is obtained.
- **`Settings.solver.variables.(variable).system_type`**. The data structure which is associated with this variable.

The subfield **`Settings.solver_module.methods`** contains a list of solving methods which are available in IPSAT in order to solve the aforementioned conservation equations. The current list of available methods is:

- **`newton_raphson`**
- **`broydens`**

- **modified_broydens**
- **brents**

For each method, six different subfields are available:

- **Settings.solver.methods.(method).function**. The function handle for this method.
- **Settings.solver.methods.(method).equations**. A cell containing the equations that are solved by this method.
- **Settings.solver.methods.(method).neq**. A vector containing the number of equations to be solved by this method for each equation type to be solved by this method.
- **Settings.solver.methods.(method).system_type**. A cell containing the name of the data structures that are associated with each equation type to be solved by this method.
- **Settings.solver.methods.(method).variables**. A cell containing the names of the variables that are obtained by the solving of the equation types solved by this method.
- **Settings.solver.methods.(method).locations**. A cell containing the locations of the equation types that are solved by this method.

Thermophysics

The field **Settings.thermophysics** contains the settings for thermophysics module. This field contains three subfields: **variables**, **global_constants** and **(specie)**. Figure 4-11 shows the data structure of the **Settings.thermophsyics** struct.

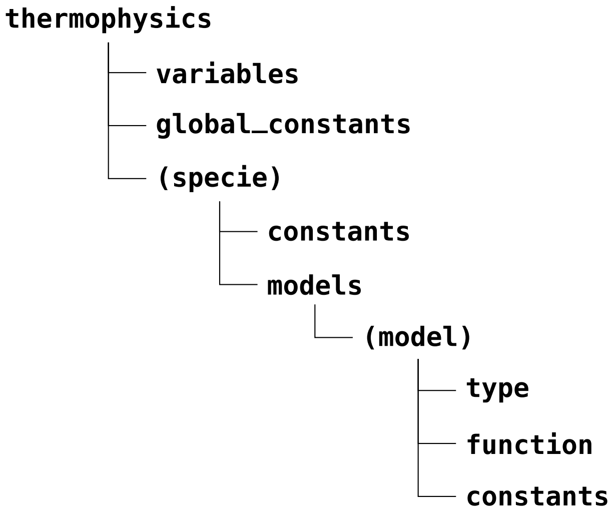


Figure 4-11: Thermophysics module settings data structure.

The field **Settings.thermophysics.variables** contains the two variables which will be used to calculate the other thermophysical properties using the available equation of state. More information on this process is presented in appendix A.

The field **Settings.thermophysics.global_constants** contains the global constants that are used by the various procedures. The following global constants are stored in this field.

- **g0** The gravitational acceleration at sea level.
- **R** The universal gas constant.
- **Na** Avogadro's number.
- **k** The Stefan Boltzmann constant.

The field **Settings.thermophysics.(specie)** contains the specie specific constants and model settings. This field has two subfields, **Settings.thermophysics.(specie).constants** and **Settings.thermophsyics.(specie).models**.

The field **Settings.thermophysics.(specie).constants** contains a list of fluid specific constants which are required for the different types of thermophysical models. The current list of constants include:

- **P_c**, Pressure at the critical point.
- **T_c**, Temperature at the critical point.
- **Rho_c**, density at the critical point.
- **P_tp**, Pressure at the triple point.
- **T_tp**, Temperature at the triple point.
- **M**, Molar mass.
- **P_0**, Reference pressure.
- **T_0**, Reference temperature.
- **H_0**, Reference enthalpy.
- **S_0**, Reference entropy.

The field **Settings.thermophysics.(specie).models** contains information about the models that are used to model the various thermophysical variables. The following list of models to model the thermophysical variables are available for a single species:

- **variable_transform**
- **melting_pressure**
- **vapor_pressure**
- **saturated_liquid_density**
- **saturated_vapor_density**
- **equation_of_state** (determines p,T,ρ,u,h,s,c_v,c_p and w)
- **surface_tension**
- **viscosity**
- **thermal_conductivity**

Note that the **`variable_transform`** and **`equation_of_state`** model do not provide a number of variables and not a single variable. For each model the following three fields are available:

- **`Settings.thermophysics.(specie).models.type`**. This field contains the name of the method which is used to calculate the specified variable. This name is used to construct the function handle.
- **`Settings.thermophysics.(specie).models.functions`**. This field contains the function handle for the model which is used to calculate the specified variable.
- **`Settings.thermophysics.(specie).models.constants`**. This field contains the model constants which are used by the model in order to calculate the specified variable.

Chapter 5

Modules

Modules are at the core of IPSAT. By providing a modular structure for delivering the functions, the program can easily be extended and modified. There is a common mechanism with which various modules interact with the program, see section 4-4. This chapter presents the modules currently available in IPSAT and how each is implemented in the software tool. In the current version of IPSAT, the following five modules are available:

1. The thermophysics module, section 5-1, determines the thermophysical properties of a fluid in a fluid node.
2. The fluid friction module, section 5-2, determines the fluid friction coefficient in a fluid branch.
3. The solver module, section 5-3, implements mathematical tools in order to minimize the residuals of the conservation equations.
4. The fluid system initialization module, section 5-4, provides initial values for the internal fluid nodes and branches.
5. The system initialization module, section 5-5, initializes the program settings.

These five modules are the core set of modules in accordance to the project goals outlined in section 2-3. Each module will be explained in further detail in the following sections.

5-1 Thermophysics Module

Accurately modeling the thermophysical parameters of fluids, and the change thereof, is essential in order to determine other derived parameters in the fluid system. This section describes the thermophysical models which are currently available in the thermophysics module in IPSAT. The thermophysics module determines the thermodynamic state, and transport, variables as function of two other state variables for all fluid nodes.

The thermophysics module requires the following fields to exists:

- **`Settings.thermophysics.global_constants`**, contains the global constants data.
- **`Settings.thermophysics.variables`**, contains the two thermophysical variables that are used as input, the same variables that are specified in the **`Settings.solver.variables`** field and are linked to a fluid node.
- **`Settings.thermophsyics.(specie)`**, contains the fluid specific information. This field is created in the thermophysics module initialize subroutine. The field is subdivided into the following three fields:
  - **`Settings.thermophysics.(specie).constants`**, contains the fluid specific constants.
  - **`Settings.thermophysics.(specie).models`**, contains the model types, the model specific constants, and the model function calls.
- **`Nodes.(ID).properties`**, contains the name of the species in the node and the values of the two state properties which are known. All properties calculated by the thermophysics module will be submitted to this field.

Due to the inherently empirical nature of modeling thermophysical data, the modeling process is heavily dependent on the source data. It is therefore chosen to maximize the flexibility of the designer by allowing the designer to select the available models according to their requirements. The designer is also free to edit or add to the available source data where required. It is also possible to change or extend the formulation used in each model.

Section 5-1-1 describes the equation of state models used in IPSAT. The vapour pressure curve and melting pressure line are described by the models presented in sections 5-1-2 and 5-1-3 respectively. Section 5-1-6 presents the model used to calculate the surface tension. The models for viscosity and thermal conductivity are presented in sections 5-1-7 and 5-1-8 respectively. Lastly, the models for the saturated liquid and saturated vapour density are given in sections 5-1-4 and 5-1-5 respectively.

Table 5-1 provides an overview of the fluids which are available in the IPSAT program and the models which are available for each fluid. Table 5-2 further clarifies the available model types for each variable. All the models which are presented in table 5-2 are further explained in sections 5-1-1 to 5-1-4.

Table 5-1: Overview of pure fluids and the available models to choose from in IPSAT. The letters designate the type of model used to model the respective property. The models which correspond to the letters can be found in table 5-2.

Fluid	**EoS**	p_m	p_v	σ	η	λ	$\rho_{l,sat}$	$\rho_{v,sat}$
Air	A (46)				A (47)	A (47)		
Argon	A (48)	A (48)	A (48)	A (49)	A (47)	A (47)	A (48)	A (48)
Ethane	A (50)	B (50)	A (50)	A (49)	A (51)		A (50)	A (50)
Ethanol	A (52)	A (53)	A (52)	A (49)		C (54)	B (52)	A (52)
Helium	A (55)	D (56)	A (55)	A (49)			A (55)	A (55)
Hydrogen	A (57)	D (56)	A (57)	A (49)		C (58)		
Methane	A (59)	B (59)	A (60)	A (49)			A (59)	A (59)
Nitrogen	A (61)	B (61)	A (61)	A (49)	A (47)	A (47)	A (61)	A (61)
Nitrous Ox.	A (62)		A (63)	A (49)			A (63)	A (63)
Oxygen	A (64)	C (64)	A (64)	A (49)	A (47),B	A (47)	B (64)	A (64)

The selection of fluids given in table 5-1 represent fluids which are currently in use within DARE (e.g. nitrous oxide, ethanol, ethane, nitrogen) or are planned to be used by DARE in the near future (e.g. oxygen, methane, helium) plus a small number of additional fluids commonly associated with propulsion system design.

Table 5-2: Overview of the different thermophysics models available in IPSAT.

Property	Model A	Model B	Model C	Model D
EoS	Helmholtz			
p_m	Simon-original	Simon-general	Polylogarithmic	Kechin
p_v	Polylogarithmic			
σ	REFPROP			
η	Lemmon	Laesecke		
λ	Lemmon	Laesecke	Assael	
$\rho_{l,sat}$	Polylogarithmic	Polynomial		
$\rho_{v,sat}$	Polylogarithmic	Polynomial		

5-1-1 Equation of State

The equation of state forms the basis of the thermophysical property model. Depending on the form, the equation of state can determine the thermophysical properties accurately for a single or multiple fluid states. The equation of state determines the pressure, density and/or temperature from one or two of the other variables. In order to increase the accuracy of the model over all regimes of the fluid, IPSAT makes use of a multiparameter equation of state. The derivatives of a multiparameter equation of state will also be able to provide other thermodynamic properties which are used in the program.

Helmholtz Energy Multiparameter Equation of State

The most state-of-the-art method of getting accurate thermophysical data, in a large regime crossing different fluid states, is to use the multiparameter equation of state written as a function of a fundamental thermodynamic parameter. There are different types of multiparameter equations of state. Four typical examples are: equations of state in terms of internal energy $u(s, \rho)$, enthalpy, $h(s, p)$, Helmholtz energy, $\alpha(\rho, T)$ and Gibbs energy, $g(T, P)$ (55). Usually the first two types are not used, because it is impossible to directly measure entropy from experiments. Out of the last two forms, the Gibbs energy formulation is more convenient to use, because pressure and temperature can be directly measured from experiments. However, the Gibbs energy formulation suffers from a discontinuity in the slope at the saturation line (55). This discontinuity becomes a problem when analyzing the thermodynamic parameters derived from the derivatives of the Gibbs energy close to the saturation line. Therefore, most commonly a formulation is given in terms of the Helmholtz energy (42). This is also the formulation which is used in IPSAT and described in this section.

A multiparameter equation of state is constructed by fitting different measurement points from many different sources. When properly designed, the multiparameter equation of state can represent the thermophysical data within the accuracy given by the measured data. This means that multiparameter equations of state are used to generate the reference table data of thermophysical properties for most pure fluids (42). The strength of the multiparameter equation of state is its accurate description of the critical region, which is where most other models experience large inaccuracies.

Due to the recent increase in computational power, it has become more suitable to directly use the results from the multiparameter equation of state instead of the interpolation of tabular data. Programs like REFPROP(4) and CoolProp (3) are examples of programs which utilize a multiparameter equation of state to calculate thermophysical data for a given set of inputs.

A Helmholtz multivariable equation of state can provide accurate thermodynamic data across different states of a fluid. A phase diagram of a typical fluid is shown in figure 5-1. The typical validity region of a Helmholtz equation of state model is illustrated by the colored region. The minimum temperature is determined by the melting line or is equal to the triple point temperature, in case an equation for the melting line is not specified. The maximum pressure and temperature are usually specified by the source of the fitting constants.

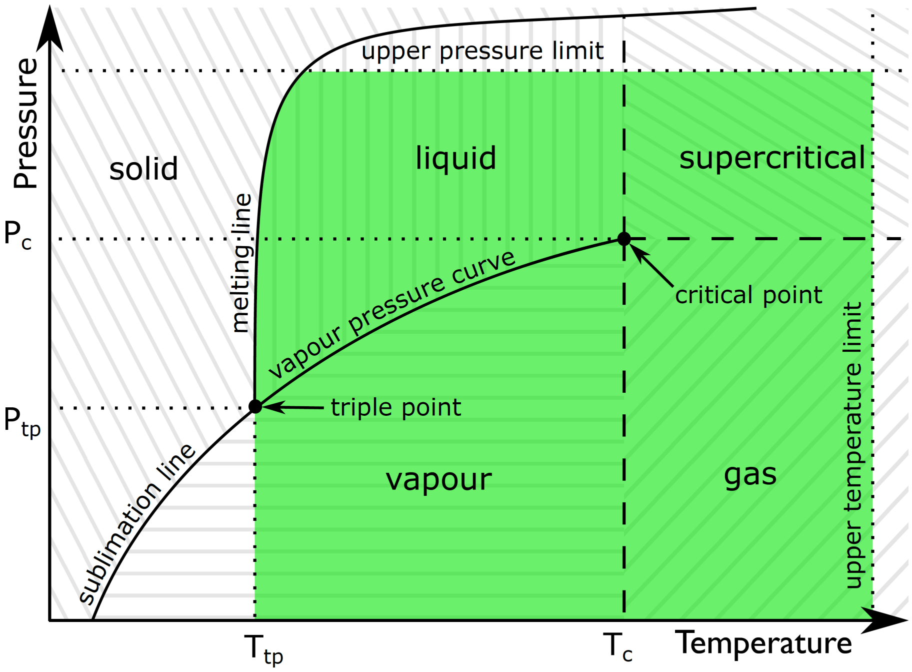


Figure 5-1: Phase diagram for a typical fluid depicting the most important parameters. The dashed area denotes the region of each state. The colored area denotes the region which can be modeled by IPSAT.

To get the relevant thermophysical properties, the Helmholtz energy is described as function of the reduced parameters, δ and τ. Thermophysical properties at both the critical point and the triple point serve as convenient reduction constants:

$$\alpha(\delta,\tau) = \alpha^0(\delta,\tau) + \alpha^r(\delta,\tau), \qquad \delta = \rho/\rho_c, \qquad \tau = T_c/T \tag{5-1}$$

Where α^0 is the reduced Helmholtz energy of an ideal gas and α^r is the real gas, or residual, contribution of the reduced Helmholtz energy. The exact formulation for both α^0 and α^r are dependent on the fluid. The ideal-gas component of the reduced Helmholtz energy is given by:

$$\alpha^0 = \frac{H^\circ \tau}{RT_c} - \frac{S^\circ}{R} - 1 + \ln\left[\frac{\delta}{\delta^\circ}\frac{\tau^\circ}{\tau}\right] - \frac{\tau}{R}\int_{\tau_0}^{\tau}\frac{c_p^\circ}{\tau^2}d\tau + \frac{1}{R}\int_{\tau_0}^{\tau}\frac{c_p^\circ}{\tau}d\tau \tag{5-2}$$

Where H°, S°, δ°, τ° and C_p° are respectively: the enthalpy, entropy, reduced density, reduced temperature and specific heat capacity at constant pressure at reference conditions where, usually, $T_0 =$ 298.15 K and $p_0 =$ 0.101325 MPa. Often, equation 5-2 is also presented in the following simpler form:

$$\alpha^0 = \ln(\delta) + \sum_{i=1}^{n} C_{1,i}\tau^{C_{2,i}} \ln(\tau)^{C_{3,i}} \ln(1 - \exp(-C_{4,i}\tau))^{C_{5,i}} \tag{5-3}$$

Where $C_{1,i}$, $C_{2,i}$ and $C4, i$ are fitting coefficients and $C_{3,i}$ and $C_{5,i}$ are equal to 1, if the term is used, or equal to 0, if the term is not used. As an example, the coefficients for nitrogen are shown in table 5-3.

Table 5-3: Ideal-gas Helmholtz energy coefficients for nitrogen (61).

i	$C_{1,i}$	$C_{2,i}$	$C_{3,i}$	$C_{4,i}$	$C_{5,i}$
1	2.5	0	1	0	0
2	-1.276 952 708·10^{1}	0	0	0	0
3	-0.007 841 63	1	0	0	0
4	-1.934 819·10^{-4}	-1	0	0	0
5	-1.237 742·10^{-5}	-2	0	0	0
6	6.678 326·10^{-8}	-3	0	0	0
7	1.012 941	0	0	26.657 88	1

If equation 5-3 is not available for a given fluid, equation 5-2 will be used to analytically evaluate the derivatives. The residual part of the reduced Helmholtz energy is often written in the following empirical structure:

$$\alpha^r = \sum_{i=1}^{n} C_{1,i}\delta^{C_{2,i}}\tau^{C_{3,i}} \exp(-C_{4,i}\delta^{C_{5,i}}) \exp(-C_{6,i}(\delta - C_{7,i})^2 - C_{8,i}(\tau - C_{9,i})^2) \tag{5-4}$$

Where $C_{1,i}$, $C_{2,i}$, $C_{3,i}$, $C_{5,i}$, $C_{6,i}$, $C_{7,i}$, $C_{8,i}$ and $C_{9,i}$ are fitting coefficients and $C_{4,i}$ is equal to 1 if the term is used, or equal to 0, if the term is not used. These coefficients are obtained from a bank of terms using an optimization procedure as described by Setzmann and Wagner (65). As an example the coefficients for nitrogen are shown in table 5-4.

Table 5-4: Residual helmholtz energy coefficients for nitrogen (61).

i	$C_{1,i}$	$C_{2,i}$	$C_{3,i}$	$C_{4,i}$	$C_{5,i}$	$C_{6,i}$	$C_{7,i}$	$C_{8,i}$	$C_{9,i}$
1	0.924 803 575 275	1.0	0.25	0	0	0	0	0	0
2	-0.492 448 489 428	1.0	0.875	0	0	0	0	0	0
3	0.661 883 336 938	2.0	0.5	0	0	0	0	0	0
4	-0.192 902 649 201·10^{1}	2.0	0.875	0	0	0	0	0	0
5	-0.622 469 309 629·10^{-1}	3.0	0.375	0	0	0	0	0	0
6	0.349 943 957 581	3.0	0.75	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
30	-0.441 513 370 350·10^{-2}	5.0	7.0	1	4	0	0	0	0
31	0.133 722 924 858·10^{-2}	6.0	4.0	1	4	0	0	0	0
32	0.264 832 491 957·10^{-3}	9.0	16.0	1	4	0	0	0	0
33	0.196 688 194 015·10^{2}	1.0	0.0	0	0	20	1	325	1.16
34	-0.209 115 600 730·10^{2}	1.0	1.0	0	0	20	1	325	1.16
35	0.167 788 306 989·10^{-1}	3.0	2.0	0	0	15	1	300	1.13
36	0.262 767 566 274·10^{4}	2.0	3.0	0	0	25	1	275	1.25

The relevant thermophysical properties of the fluid are represented by the partial derivatives of α^0 and α^r. The equations for the main thermophysical properties are shown in table 5-5. IPSAT uses the analytical derivative of equations 5-3 and 5-4, see Appendix C, to calculate the values of the partial derivatives listed in table 5-5.

Table 5-5: List of equations to calculate the various thermodynamic state properties.

Property	Equation
Compressibility factor	$$z = \frac{p}{\rho RT} = 1 + \delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau \qquad (5\text{-}5)$$
Pressure	$$p = \rho RT\left[1 + \delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau\right] \qquad (5\text{-}6)$$
Specific internal energy	$$u = \tau RT\left[\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta + \left(\frac{\partial\alpha^r}{\partial\tau}\right)_\delta\right] \qquad (5\text{-}7)$$
Specific enthalpy	$$h = \tau RT\left[\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta + \left(\frac{\partial\alpha^r}{\partial\tau}\right)_\delta\right] + \delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau + 1 \qquad (5\text{-}8)$$
Specific entropy	$$s = \tau R\left[\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta + \left(\frac{\partial\alpha^r}{\partial\tau}\right)_\delta\right] - \alpha^0 - \alpha^r \qquad (5\text{-}9)$$
Specific heat capacity at constant volume	$$c_v = -\tau^2 R\left[\left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta + \left(\frac{\partial^2\alpha^r}{\partial\tau^2}\right)_\delta\right] \qquad (5\text{-}10)$$
Specific heat capacity at constant pressure	$$c_p = c_v + R\frac{\left[1 + \delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau - \delta\tau\left(\frac{\partial^2\alpha^r}{\partial\delta\partial\tau}\right)\right]^2}{\left[1 + 2\delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau + \delta^2\left(\frac{\partial^2\alpha^r}{\partial\delta^2}\right)_\tau\right]} \qquad (5\text{-}11)$$
Speed of sound	$$w = \sqrt{RT\frac{c_p}{c_v}\left[1 + 2\delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau + \delta^2\left(\frac{\partial^2\alpha^r}{\partial\delta^2}\right)_\tau\right]} \qquad (5\text{-}12)$$

Equations 5-3 and 5-4 are written explicitly as function of density and temperature. In the case that one or more of the thermodynamic state variables listed in table 5-5 are the independent variables, the other state variables can be found by implementing an iterative procedure. This is explained in more detail in appendix A.

5-1-2 Vapour Pressure

The vapour pressure line determines the transition of the fluid between liquid and vapour state from the triple point to the critical point. The vapour pressure is used to determine the state of the fluid between the liquid and vapour region. The state is then used to determine an appropriate guess for the density when evaluating the multiparameter equation of state when density or temperature are dependent variables. The vapour pressure, at a given temperature, is an output of the thermophysics module. The default unit of the vapour pressure is Pa.

Polylogarithmic Formulation

In a study published by Wagner (66) a new method for establishing vapour pressure equations was proposed. In this method a preliminary vapour pressure equation is constructed with a large bank of terms. For each fluid the terms are reduced iteratively according to their statistical significance. This means that the equation does not have a fixed amount of terms. The general form of the equation in polylogarithmic form is shown in equation 5-13.

$$\ln\left(\frac{p_v}{p_c}\right) = \tau\sum_{i=1}^{n} C_{1,i}\left(1-\frac{1}{\tau}\right)^{C_{2,i}} \qquad \tau = T_c/T \tag{5-13}$$

Where p_v is the vapour pressure in Pa, p_c is the critical pressure in Pa, T is the temperature in K, T_c is the critical temperature in K and $C_{1,i}$ and $C_{2,i}$ are fitting constants. It must be noted that when $T > T_c$ this equation becomes invalid, and p_v is automatically set to zero. The same holds for $T < T_{tp}$

5-1-3 Melting Pressure

The melting line determines the transition of the fluid between solid and liquid state from the triple point. The melting line serves as a lower temperature boundary for the multiparameter equation of state in case a formulation for the melting line is available. When an equation for the melting line is not available, IPSAT will assume a lower temperature limit equal to the triple point temperature for all values of pressure. The melting pressure, at a given temperature, is an output of the thermophysics module. The default unit of the melting pressure is Pa.

Original Simon-Glatzel Formulation

Simon and Glatzel formulated a melting pressure relation in 1928 (67) which is still commonly used to describe the melting line. This formulation is shown in equation 5-14.

$$T_m = T_r\left(\frac{p_m - p_r}{a} + 1\right)^{\frac{1}{c}} \tag{5-14}$$

Where p_m and T_m are the melting pressure and melting temperature respectively, a and c are fitting constants and p_r and T_r are a reference pressure and reference temperature respectively. The reference temperature and pressure are usually taken to be the triple point pressure and temperature.

Rewriting equation 5-14, gives the following usable form to calculate the melting pressure:

$$p_m = C_1 \left[\left(\frac{T}{T_r} \right)^{C_2} - 1 \right] + p_r \tag{5-15}$$

General Simon-Glatzel Formulation

Often, a more general form of the Simon-Glatzel formulation is used to describe the melting line to accommodate more terms. The general form can be seen in equation 5-16.

$$\frac{p_m}{p_{tp}} = 1 + \sum_{i=1}^{n} C_{1,i} \left[\left(\frac{T}{T_{tp}} \right)^{C_{2,i}} - 1 \right] \tag{5-16}$$

Where p_m is the melting pressure $C_{1,i}$ and $C_{2,i}$ are fitting constants and p_{tp} and T_{tp} are the triple point pressure and temperature respectively.

Polylogarithmic Formulation

A polylogarithmic formulation of the melting line is proposed by Watson (64) and is used for the calculation of the melting line for oxygen. This polylogarithmic function has the following form:

$$\ln \left(\frac{p_m}{p_{tp}} \right) = \sum_{i=1}^{n} C_{1,i} \left(\frac{T}{T_{tp}} - 1 \right)^{C_{2,i}} \tag{5-17}$$

Where p_m is the melting pressure, $C_{1,i}$ and $C_{2,i}$ are fitting constants and p_{tp} and T_{tp} are the triple point pressure and temperature respectively.

Kechin Formulation

Kechin proposed an improved version of the original Simon-Glatzel formulation (68) which allows for the existence of maxima in the melting curve. The formulation is given in the following form:

$$T_m = T_r \left(1 + \frac{p_m}{C_1} \right)^{C_2} \exp(C_3 p_m) \tag{5-18}$$

Where T_r is a reference temperature (melting temperature at $p = 0$) and C_1, C_2 and C_3 are fitting constants. Note that C_1 and C_2 have dimensions which are Pa and Pa^{-1} respectively. It can be seen that the formulation is gives T_m as function of p_m. In order to obtain p_m as function of T the following function needs to be minimized:

$$T_r \left(1 + \frac{p_m}{C_1} \right)^{C_2} \exp(C_3 p_m) - T = 0 \tag{5-19}$$

5-1-4 Saturated Liquid Density

The saturated liquid density is the density of the fluid on the liquid side of the saturation line. The value of the saturated liquid density can aid in the the search for an initial guess for the fluid density in case that density is not an independent variable. The density of the saturated liquid, at a given temperature, is an output of the thermophysics module. The default unit of the saturated liquid density is kg·m^{-3}.

Polylogarithmic formulation

An often used formulation for the saturated liquid density is a bank of polylogarithmic fitting terms optimized using the method developed by Setzmann and Wagner (65). A generalized form of the result of the optimization process is shown in equation 5-20.

$$\ln\left(\frac{\rho_{l,sat}}{\rho_c}\right) = \sum_{i=1}^{n} C_{1,i}\left(1-\frac{1}{\tau}\right)^{C_{2,i}} \qquad \tau = T_c/T \tag{5-20}$$

Where $C_{1,i}$ and $C_{2,i}$ are fitting constants. The designer also has the option to switch terms on or off by setting $C_{2,i}$ to zero.

Polynomial formulation

In some cases the saturated liquid density is given in a polynomial form. This form is shown in equation 5-21.

$$\frac{\rho_{l,sat}}{\rho_c} = \sum_{i=1}^{n} C_{1,i}\left(1-\frac{1}{\tau}\right)^{C_{2,i}} \qquad \tau = T_c/T \tag{5-21}$$

Where $C_{1,i}$ and $C_{2,i}$ are fitting constants.

5-1-5 Saturated Vapour Density

The saturated vapour density is the density of the fluid on the vapour side of the saturation line. The value of the saturated vapour density can aid in the search for an initial guess for the fluid density in case the density is not an independent variable. The density of the saturated vapour, at a given temperature, is an output of the thermophysics module. The default unit of the saturated vapour density is kg·m^{-3}.

Polylogarithmic formulation

Similar to the saturated liquid density, the saturated vapour density formulation is often constructed by an optimized bank of terms using the method of Setzmann and Wagner (65). A general form of the polylogarithmic formulation is shown in equation 5-22.

$$\ln\left(\frac{\rho_{v,sat}}{\rho_c}\right) = \sum_{i=1}^{n} C_{1,i}\tau^{C_{2,i}}\left(1-\tau\right)^{C_{3,i}}\left(1-\frac{1}{\tau}\right)^{C_{4,i}} \qquad \tau = T_c/T \tag{5-22}$$

Where $C_{1,i}$ and $C_{2,i}$ are fitting constants. The designer also has the option to switch the temperature dependent term on or off by setting $C_{2,i}$, $C_{3,i}$ and $C_{4,i}$ to zero.

Polynomial formulation

In some cases the saturated liquid density is given in a polynomial form. This form is shown in equation 5-23.

$$\frac{\rho_{v,sat}}{\rho_c} = \sum_{i=1}^{n} C_{1,i} \left(1 - \frac{1}{\tau}\right)^{C_{2,i}} \tag{5-23}$$

Where $C_{1,i}$ and $C_{2,i}$ are fitting constants.

5-1-6 Surface Tension

The surface tension is a thermophysical property which is used in the characterization of droplet formation. It is also used as an indirect parameter in the instability analysis for the analysis of droplet and acoustic coupling. The value of surface tension is currently used in both the atomization model and the instability model. The default unit of the surface tension is N·m^{-1}.

REFPROP Formulation

In a study done by Mulero (49), the coefficients for the determination of the surface tension of 81 fluids were redetermined using the REFPROP correlation. This study improved the results, which were given by REFPROP, for a set of 37 fluids. More importantly, this study openly published the fitting coefficients. The formulation used by REFPROP is shown in equation 5-24

$$\sigma(T) = \sum_{i=1}^{n} C_{1,i} \left(1 - \frac{1}{\tau}\right)^{C_{2,i}} \qquad \tau = T_c/T \tag{5-24}$$

Where $C_{1,i}$ and $C_{2,i}$ are fitting coefficients. It has to be noted that this formulation assumes a correlation only in temperature, thus the effect of pressure is ignored. This assumption might result in values that become significantly erroneous at extreme conditions. This limitation should be taken into account when doing an analysis which is heavily dependent on the surface tension values. Needless to say, the surface tension relation is only valid in the liquid region of the state diagram. For all other states of the fluid, the surface tension is set to zero. The study done by Mulero also provides the temperature range for which the formulation is valid. This temperature range should be taken into account when the value of the surface tension is used when solving a problem.

5-1-7 Viscosity

Viscosity is a thermophysical property describing the resistance of the fluid against deformation caused by shear stress. The viscosity which is mentioned in this report is always assumed to be the dynamic viscosity. In IPSAT this thermophysical property is used in the determination of the Reynolds number and the calculation of the fluid friction factor. The default unit of viscosity is Pa·s.

Lemmon Formulation

Lemmon and Jacobsen developed a formulation for viscosity of Nitrogen, Oxygen, Argon and Air (47). This formulation is constructed similarly to how the Helmholtz multiparameter equation of state is constructed and is based on the generalized formulation proposed by Vesovic and Wakeham (69). The viscosity has two separate components, the dilute gas viscosity (zero density), η^0, which is only dependent on temperature and the residual viscosity, η^r, which is dependent on density and temperature:

$$\eta(\delta,\tau) = \eta^0(\tau) + \eta^r(\delta,\tau) \qquad \delta = \rho/\rho_c, \qquad \tau = T_c/T \tag{5-25}$$

The dilute viscosity is obtained from kinetic theory and is given by equation 5-26. For more information regarding the derivation of this equation the reader is referred to (70).

$$\eta^0(\tau) = \frac{5}{16}\sqrt{\frac{kMT_c/\tau}{\pi N_A}}\frac{1}{\sigma_{LJ}^2\Omega(\tau^*)}, \qquad \tau^* = T_c/(\tau\epsilon/k) \tag{5-26}$$

Where k is the Boltzmann constant, M is the molar mass, N_A is Avogadro's constant, σ_{LJ} is the Lennard-Jones size parameter, Ω is the collision integral and ϵ/k is the Lennard-Jones energy parameter. The collision integral is given by equation 5-27.

$$\Omega(T^*) = \exp\left(\sum_{i=0}^{n} C_i \ln(T^*)^i\right) \tag{5-27}$$

Where C_i are fitting coefficients.

The residual viscosity component is constructed by optimizing a bank of terms similarly to how the relation of residual Helmholtz energy is constructed in equation 5-4. The residual viscosity relation according to Lemmon is constructed according to the following bank of terms:

$$\eta^r(\delta,\tau) = \sum_{i=1}^{n} C_{1,i}\tau^{C_{2,i}}\delta^{C_{3,i}}\exp(-C_{4,i}\delta^{C_{5,i}}) \qquad \delta = \rho/\rho_c, \qquad \tau = T_c/T \tag{5-28}$$

Where δ and τ are the reduced density and temperature respectively, $C_{1,i}$, $C_{2,i}$, $C_{3,i}$ and $C_{5,i}$ are fitting coefficients and $C_{4,i}$ is equal to 1 if the term is used, or equal to 0, if the term is not used.

5-1-8 Thermal Conductivity

Thermal conductivity is a measure of how well a fluid/material conducts heat. It is used when evaluating transient heat conduction using Fourier's law of heat conduction. The default unit for the thermal conductivity is W·m^{-1}·K^{-1}.

Lemmon Formulation

Lemmon and Jacobsen developed a new formulation for the thermal conductivity of Nitrogen, Oxygen, Argon and Air (47). The thermal conductivity has three components: the dilute-gas (zero density) thermal conductivity, λ^0, the residual fluid thermal conductivity, λ^r and the critical enhancement thermal conductivity, λ^c. Similarly to the formulation of the fluid viscosity, the formulation of the thermal conductivity is based on the generalized formulation proposed by Vesovic and Wakeham (69).

$$\lambda = \lambda^0(\tau) + \lambda^r(\delta, \tau) + \lambda^c(\delta, \tau) \qquad \delta = \rho/\rho_c, \qquad \tau = T_c/T \tag{5-29}$$

In the formulation proposed by Lemmon, the dilute-gas thermal conductivity is given by the following fitting function:

$$\lambda^0(\tau) = C_1\eta^0(T) + C_2\tau^{C_3} + C_4\tau^{C_5} \qquad \tau = T_c/T \tag{5-30}$$

Where η^0 is the dilute viscosity and C_1 to C_5 are fitting constants.

The residual thermal conductivity relation is constructed according to the following bank of terms:

$$\lambda^r(\delta, \tau) = \sum_{i=1}^{n} C_{1,i}\tau^{C_{2,i}}\delta^{C_{3,i}} \exp(-C_{4,i}\delta^{C_{5,i}}) \qquad \delta = \rho/\rho_c, \qquad \tau = T_c/T \tag{5-31}$$

Where δ and τ are the reduced density and temperature respectively, $C_{1,i}$, $C_{2,i}$, $C_{3,i}$ and $C_{5,i}$ are fitting coefficients and $C_{4,i}$ is equal to 1 if the term is used, or equal to 0, if the term is not used.

The critical enhancement term is relevant when the fluid is close to the critical region. For this term the following formulation from Olchowy and Sengers is used (71):

$$\lambda^c(\tau, \delta) = \rho c_p \frac{kR_0T}{6\pi\xi\eta(\delta, \tau)}(\tilde{\Omega} - \tilde{\Omega}_0) \tag{5-32}$$

Where ρ and T is the density and temperature respectively, c_p is the heat capacity at constant pressure, k is the Boltzmann constant, R_0 is the amplitude and ξ is the correlation length given by equation 5-35. The terms $\tilde{\Omega}$ and $\tilde{\Omega}_0$ are given by equations 5-33 and 5-34.

$$\tilde{\Omega} = \frac{2}{\pi}\left[\left(\frac{c_p - c_v}{c_p}\right)\tan^{-1}\left(\frac{\xi}{q_D}\right) + \frac{c_v}{c_p}\left(\frac{\xi}{q_D}\right)\right] \tag{5-33}$$

$$\tilde{\Omega}_0 = \frac{2}{\pi}\left[1 - \exp\left(\frac{-1}{\left(\frac{\xi}{q_D}\right)^{-1} + \frac{1}{3}\left(\frac{\xi}{q_D}\right)^2\left(\frac{\rho_c}{\rho}\right)^2}\right)\right] \tag{5-34}$$

$$\xi = \xi_0\left(\frac{p_c\rho}{\Gamma\rho_c^2}\right)^\nu \left[\left.\frac{\partial\rho(T, \rho)}{\partial p}\right|_T - \frac{T_{ref}}{T}\left.\frac{\partial\rho(T_{ref}, \rho)}{\partial p}\right|_T\right]^\nu \tag{5-35}$$

Where q_D is the maximum cutoff wave number, Γ and ξ_0 are amplitudes, ν is the critical exponent and p_c and ρ_c are the critical pressure and density respectively. T_{ref} is a reference temperature which is usually set equal to twice the critical temperature (71).

As can be seen in equation 5-35, the correlation length is a function of the partial derivative of density with respect to pressure. This means that the function for density needs to be differentiable with respect to pressure for all valid ranges of density and temperature.

Assael Formulation

Assael proposed a new formulation for the thermal conductivity for normal hydrogen and parahydrogen (58). The formulation of Assael is similar to that of Lemmon. The thermal conductivity is composed of three different terms as described in equation 5-29. However, the definition of the dilute-gas and residual thermal conductivity is different from the Lemmon formulation. The dilute-gas thermal conductivity, given by Assael, is:

$$\lambda^0(\tau) = \frac{\sum_{i=0}^{n} C_{1,i}\tau^{-i}}{\sum_{i=0}^{m} C_{2,i}\tau^{-i}} \qquad \tau = T_c/T \tag{5-36}$$

Where $C_{1,i}$ and $C_{2,i}$ are fitting constants. The residual thermal conductivity as given by Assael is:

$$\lambda^r(\delta,\tau) = \sum_{i=1}^{n}(C_{1,i} + C_{2,1}\tau^{-1})\delta^i \qquad \delta = \rho/\rho_c, \qquad \tau = T_c/T \tag{5-37}$$

Where $C_{1,i}$ and $C_{2,i}$ are fitting constants. The enhancement term for close to the critical region, is the formulation proposed by Olchowy and Sengers (71), similar to what is used in the formulation posed by Lemmon. The critical enhancement term is given by equation 5-32.

5-1-9 Thermophysics Module Architecture

Figure 5-2 shows the program architecture of the thermophysics module. The thermophysics module gets called each iteration by the solver module in order to update the thermophyisical variables in each fluid node. The function **`thermophysics_module()`** is the main function that gets called and oversees the correct execution of the thermophysics module. The function **`evaluate_thermophysics()`** submits the thermophysical variables for an individual fluid node in the struct **`Properties`**.

The function **`evaluate_thermophysics()`** uses two different function libraries: the variable transform function library and the thermophysics function library. The variable transform library contains all the functions which are needed to be able to calculate the reduced pressure and temperature (δ and τ) from a set of state variables which are stored in the **`Properties`** data structure. More information about this process can be found in appendix A.

The thermophysics function library contains all the available functions to calculate thermophysical properties for a given density and temperature. These functions encompass the methods which were presented in sections 5-1-1 to 5-1-8. The designer determines the functions which are executed. These preferences are stored in the **`Settings`** data structure.

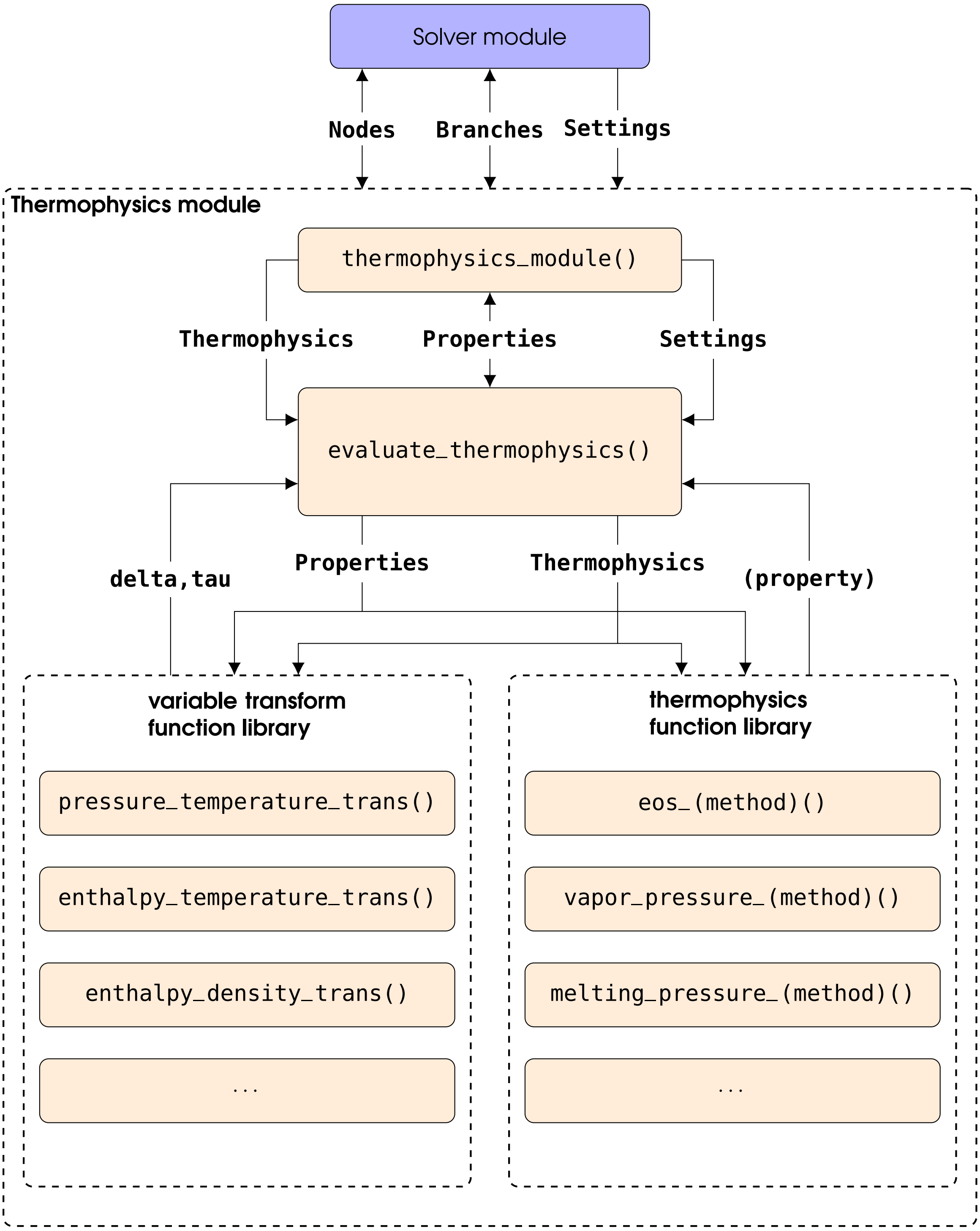


Figure 5-2: Thermophysics module architecture as implemented in IPSAT.

5-2 Fluid Friction Module

The fluid friction module calculates the fluid friction factor, which has the unit of kg^{-1}·m^{-1}, in each fluid branch. The method which is used to evaluate the feed system friction factor is similar to the approach used by GFSSP (10). The friction factor is used in the evaluation of the conservation of momentum, equation 5-79, and the conservation of fluid energy, equation 5-95, when viscous force terms are taken into account.

The fluid friction module requires the following properties to be defined for each upstream fluid node and each fluid branch:

- **`Nodes.(ID).properties`**, contains the thermophysical properties. In particular the property **`density`** is used in the fluid friction module.
- **`Branches.(ID).properties`**, contains the flow properties of the branch. In particular the property **`reynolds_number`** is used in the fluid friction module.
- **`Branches.(ID).geometry`**, contains the geometric properties of the branch. The required geometrical parameters depends on the specific friction model.

The value for the fluid friction coefficient is defined according to equation 5-38. The value of K_f is determined by using empirically models which dependend on the type of branch and the choice of the designer. The following types of models are currently available for the determination of the fluid friction factor:

- pipe flow(circular duct) friction, see section 5-2-1.
- non-circular ducts friction, see section 5-2-2.
- flow through a restriction

The fluid friction factor that is used by IPSAT is defined as follows:

$$K_f = \frac{\Delta p}{\dot{m}^2} \tag{5-38}$$

5-2-1 Pipe flow friction

The original Darcy-Weisbach relation writes the loss in pressure due to friction in a pipe as follows:

$$\Delta p = f \frac{L}{D} \frac{\rho v^2}{2} \tag{5-39}$$

Where f is the dimensionless Darcy friction factor, also known as the flow coefficient, L and D are the length and diameter of the pipe in m, ρ is the density of the fluid in kg·m^{-3}, and v is the mean flow velocity in m·s^{-1}. The mean flow velocity can also be written in the following form (from the continuity equation):

$$v = \frac{\dot{m}}{\rho A} = \frac{4\dot{m}}{\rho \pi D^2} \tag{5-40}$$

Combining equations 5-38, 5-39 and 5-40 gives the following relation for the friction factor in a pipe:

$$K_f = \frac{8fL}{\rho\pi^2 D^5} \tag{5-41}$$

Assuming that the flow in the pipe is fully developed, the Darcy friction factor can be obtained from the Colebrook relation:

$$\frac{1}{\sqrt{f}} = -2\log_{10}\left(\frac{\epsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}\right) \tag{5-42}$$

Where ϵ is the pipe roughness in m, and Re is the Reynolds number, which is defined as:

$$Re = \frac{\dot{m}D}{\mu A} \tag{5-43}$$

Where μ is the fluid viscosity in Pa·s. Note that equation 5-42 is an implicit equation. This means that the value of the Darcy friction factor needs to be obtained iteratively using a root finding method. IPSAT uses a modified brent's method with a one value input guess, see section 5-3-1.This initial guess for the Darcy friction factor can be obtained using an explicit formulation by Haaland (72):

$$\frac{1}{\sqrt{f_{guess}}} = -1.8\log_{10}\left[\left(\frac{\epsilon}{3.7D}\right)^{1.11} + \frac{6.9}{Re}\right] \tag{5-44}$$

In general, the results from equation 5-44 will be very close to the results which are given by the Colebrook formulation. It is therefore possible for the designer to select the results from the Haaland relation as the final result for the Darcy friction factor in order to reduce computational effort.

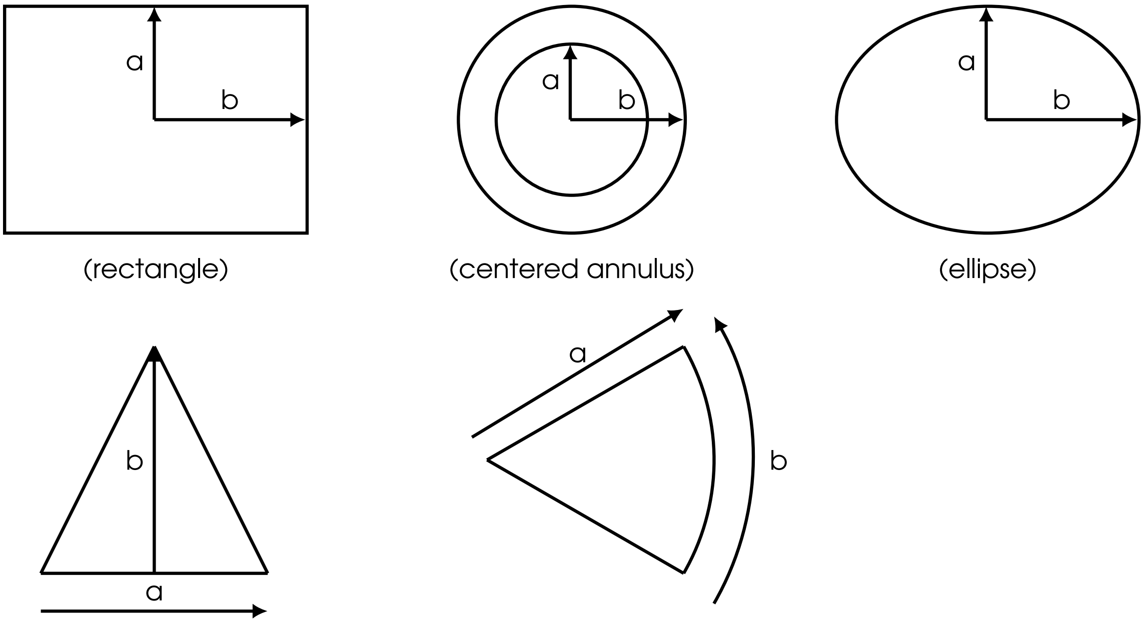
5-2-2 Non-circular ducts friction

The procedure for getting the fluid friction factor for non-circular ducts with constant cross section is similar to the procedure described by White (73). The method proposed by White links the friction factor of a non-circular duct to the friction factor through a pipe as described in the previous section. The method introduces the geometric specific Poiseuille number and the effective diameter.

The Poiseuille number is a dimensionless number which relates the specific cross sectional geometry to the friction factor. It is derived from the Poiseuille law of viscous fluid flow (73). In literature, data can be obtained for the relation between geometry of a cross section and the Poiseuille number. The following fitting equation is often used to relate the geometry of the cross section to the Poiseuille number:

$$Po = \sum_{i=1}^{n} C_i \left(\frac{a}{b}\right)^{i-1} \tag{5-45}$$

Where a and b are the dimensional parameters of the cross section and C_1 to C_n are fitting constants. Figure 5-3 shows various different shapes for which a relation is available. The figure also shows the dimensional parameters for each shape.



(rectangle) (centered annulus) (ellipse)

(isosceles triangle) (circular section)

Figure 5-3: Definition of the short side (a) and long side (b) for different cross sectional shapes for the calculation of the Poiseuille number.

In IPSAT the poiseuille number is modeled using equation 5-45 with a polynomial of degree 4. Figure 5-4 shows the data points from a study done by Shah and London (74) that show the variation of the poiseuille number for different geometries. The trend of these points have been approximated by a best fit curve of degree 4 that satisfies equation 5-45. The constants that are used for these trends are shown in table 5-6.

Once the Poiseuille number has been determined, the friction factor can be calculated using a simple procedure (73, 10):

1. Determine the hydraulic diameter, or equivalent circle diameter, according to the following formula:

$$D_h = \frac{4A}{c} \tag{5-46}$$

   Where A is the area of the cross section and c is the circumference of the cross section.

2. Determine the corresponding Reynolds number:

$$Re_h = \frac{\dot{m}D_h}{\mu A} \tag{5-47}$$

   Depending on the value of Re_h the flow is considered laminar or turbulent:

   - In case $Re_h < 2300$ (laminar flow):

$$f = \frac{4Po}{Re_h} \tag{5-48}$$

   - In case $Re_h \geq 2300$ (turbulent flow): Determine the effective diameter:

$$D_{eff} = \frac{16D_h}{Po} \tag{5-49}$$

Determine the effective Reynolds number:

$$Re_{eff} = \frac{\dot{m}D_{eff}}{\mu A} \tag{5-50}$$

Determine the Darcy friction factor using equation 5-42 with $D = D_{eff}$ and $Re = Re_{eff}$:

$$\frac{1}{\sqrt{f}} = -2\log_{10}\left(\frac{\epsilon}{3.7D_{eff}} + \frac{2.51}{Re_{eff}\sqrt{f}}\right) \tag{5-51}$$

3. Determine the friction factor using equation 5-41:

$$K_f = \frac{8fL}{\rho\pi^2 D_h^5} \tag{5-52}$$

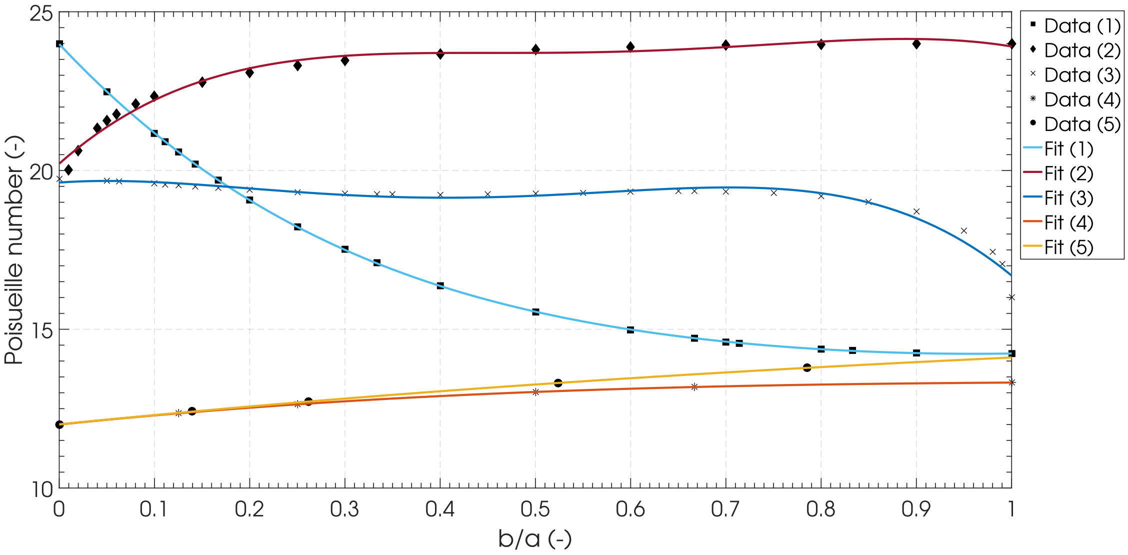


Figure 5-4: The variation of the Poiseuille number with respect to the ratio of the short dimension (a) over the long dimension (b) (see figure 5-3), for a rectangular cross section (1), a centered annular cross section (2), an elliptical cross section (3), a triangular cross section (4) and a circular section cross section (5). The marked points are the data points from Shah and London (74) and the lines are polynomial fitting curves of degree 4 (see equation 5-45 and table 5-6).

Table 5-6: Constants for equation 5-45 to calculate the Poiseuille number with a polynomial of degree 4.

	Rectangle	**Centered annulus**	**Ellipse**	**Isosceles triangle**	**Circular section**
C_1	$2.398{\cdot}10^{1}$	$2.022{\cdot}10^{1}$	$1.962{\cdot}10^{1}$	$1.200{\cdot}10^{1}$	$1.201{\cdot}10^{1}$
C_2	$-3.198{\cdot}10^{1}$	$2.657{\cdot}10^{1}$	$2.132{\cdot}10^{0}$	$3.115{\cdot}10^{0}$	$3.033{\cdot}10^{0}$
C_3	$4.222{\cdot}10^{1}$	$-7.377{\cdot}10^{1}$	$-2.528{\cdot}10^{1}$	$-2.449{\cdot}10^{1}$	$-1.176{\cdot}10^{0}$
C_4	$-2.798{\cdot}10^{1}$	$8.749{\cdot}10^{1}$	$5.720{\cdot}10^{1}$	$6.421{\cdot}10^{-1}$	$2.758{\cdot}10^{-1}$
C_5	$7.985{\cdot}10^{0}$	$-3.660{\cdot}10^{1}$	$-3.698{\cdot}10^{1}$	$1.320{\cdot}10^{-2}$	$-2.790{\cdot}10^{-2}$

5-2-3 Fluid Friction Module Architecture

Figure 5-5 shows the program architecture of the fluid friction module. The fluid friction module is called during each iteration, in case friction is taken into account, in order to update the fluid friction factor in each fluid branch. The function **`fluid_friction_module()`** is the main function that is called and it loops through all fluid branches in order to calculate the fluid friction factor for the given branch and node properties. The function will calculate the fluid friction factor using the methods described in sections 5-2-1 to 5-2-2.

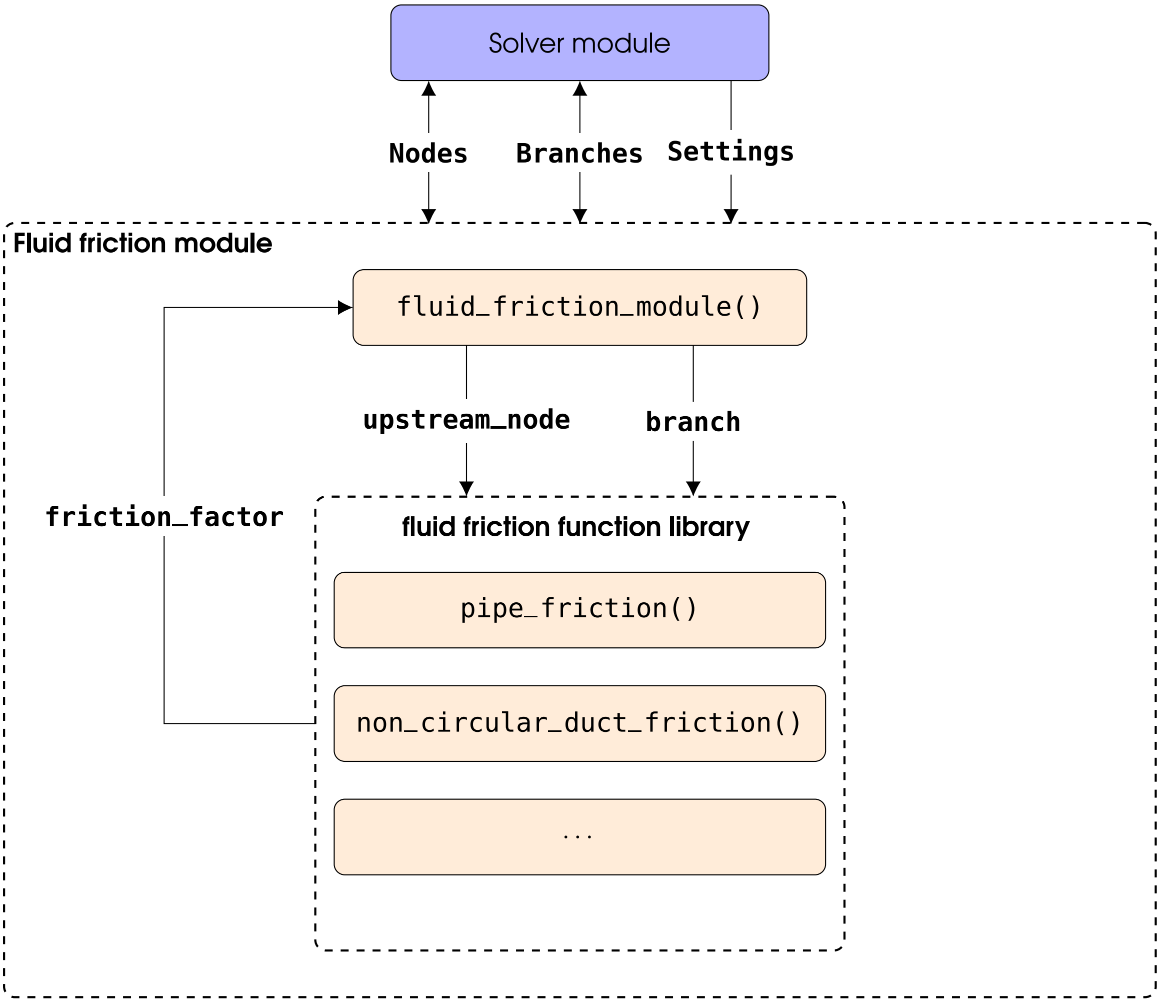


Figure 5-5: Fluid friction module architecture as implemented in IPSAT.

5-3 Solver Module

Due to the large number of unknown variables and coupled equations, the problem has to iterated in order to arrive at a solution. This procedure is inextricably linked with programs using a Finite Element Method (FEM) scheme or a Finite Volume Method (FVM) scheme. The core of this iteration procedure is the implementation of a solving method. Several different solving methods are employed by IPSAT and these methods are discussed in this chapter.

The solver module requires the following fields to be present:

- **`Settings.solver`**, contains the settings for the solver. This field contains the following two subfields:
  - **`Settings.solver.mode`**, contains the mode of the solver (**`steady_state`** or **`transient`**).
  - **`Settings.solver.iteration`**, contains the maximum number of iterations (**`max`**) and the minimum number of iterations (**`min`**) to be executed by the solver module.
  - **`Settings.solver.system`**, contains the summary of all the variables, equations and methods used by the solver module.
  - **`Settings.solver.equations`**, contains the information about all the conservation equations that are used by the solver module, including the function calls and the equation terms that are active.
  - **`Settings.solver.variables`**, contains the information about all the variables that are solved by the solver module.
  - **`Settings.solver.methods`**, contains the information about the methods that are implemented by the solver module.
- All other fields required by the modules which are called by the solver module.

Solving a set of equations can always be brought back to the classical equivalent problem of finding the root of a (set of) function(s). There are a number of well known and proven methods for finding the root of a function. These methods will be further elaborated upon in section 5-3-1.

The problem is considered solved if the residuals of the conservation equations are within the given convergence criteria. The definition of these criteria, and how they are applied in the program, is shown in section 5-3-4.

5-3-1 Root Finding Methods

There are many different approaches to finding a root of a function, where each one has its specific advantages. This section presents the different root finding methods used by IPSAT. A full study of all the different root finding approaches is beyond the scope of this thesis project. The focus of this section will mainly be on explaining the methods used, why they are selected and how they are implemented.

Newton-Raphson Method

The Newton-Raphson method (also known as Newton's method) solves a system of non linear functions by transforming the problem into a system of linear equations based on the local values of the partial derivatives of the function. The premise of this method is that the function can be (numerically) differentiated for the complete range of applicable values.

In IPSAT, the Newton-Raphson solver is a method used in solving the system of coupled conservation equations which describes the fluid system. In the case of a single function with a single variable, the Newton-Raphson method can be written in the following form:

$$\mathbf{x_{i+1}} = \mathbf{x_i} - \frac{f(\mathbf{x_i})}{f'(\mathbf{x_i})} \tag{5-53}$$

Where $\mathbf{x_n}$ is the current variable value, x_{n+1} is the new value of the variable, and $f(\mathbf{x_i})/f'(\mathbf{x_i})$ is the correction factor; where the value of the function, $f(\mathbf{x_i})$, is divided by the derivative of the function, $f'(\mathbf{x_n})$. When solving a system of coupled equations, the Newton-Raphson method can be rewritten to the form presented in equation 5-54. This equation is rewritten to a form where the corrections can be obtained by solving a set of linear equations in order to avoid dividing by $f'(\mathbf{x_n})$.

$$\mathbf{J}(\mathbf{f}(\mathbf{x_i}))\mathbf{x_c} = \mathbf{f}(\mathbf{x_i}) \tag{5-54}$$

Where $\mathbf{x_i}$ is the vector of current values for all relevant variables, $\mathbf{J}(\mathbf{f}(\mathbf{x_n}))$ is the Jacobian of the system of equations, $\mathbf{x_c}$ is the correction vector and $\mathbf{f}(\mathbf{x_n})$ is the vector of function evaluations at the current conditions, which are the residuals of the functions. The new values of the variables can be found by applying equation 5-55.

$$\mathbf{x_{i+1}} = \mathbf{x_i} - \mathbf{x_c} \tag{5-55}$$

In order for a unique solution to exist the system of n variables $\{x_1, \ldots, x_n\}$ needs to be complemented by a system of n independent equations $\{f_1, \ldots, f_n\}$. This means that equation 5-54 can be rewritten in matrix form as:

$$\begin{bmatrix} \frac{\partial f_1(\mathbf{x_i})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x_i})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x_i})}{\partial x_1} & \cdots & \frac{\partial f_n(\mathbf{x_i})}{\partial x_n} \end{bmatrix} \begin{bmatrix} x_{c,1} \\ \vdots \\ x_{c,n} \end{bmatrix} = \begin{bmatrix} f_1(\mathbf{x_i}) \\ \vdots \\ f_n(\mathbf{x_i}) \end{bmatrix} \tag{5-56}$$

The correction vector can be obtained by multiplying the inverse of the Jacobian matrix with the current set of function evaluations $\mathbf{f}(\mathbf{x_n})$:

$$\mathbf{x_c} = \mathbf{J}(\mathbf{f}(\mathbf{x_n}))^{-1} \cdot \mathbf{f}(\mathbf{x_n}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x_i})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x_i})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x_i})}{\partial x_1} & \cdots & \frac{\partial f_n(\mathbf{x_i})}{\partial x_n} \end{bmatrix}^{-1} \cdot \begin{bmatrix} f_1(\mathbf{x_i}) \\ \vdots \\ f_n(\mathbf{x_i}) \end{bmatrix} \tag{5-57}$$

When solving a large number of equations, it is important to structure the grouping of the partial derivatives in the Jacobian matrix. In IPSAT the Jacobian matrix is constructed by sorting the equations, in groups of equation type, in the designated rows. The variables are sorted, in groups of variable type, in the designated columns. This allows for an ordered construction of the matrix where the designer can easily add and remove equations to be solved using the Newton-Raphson method. The general structure used by IPSAT is shown in equation 5-58. In this structure an equation, and its accompanying variable, can be added or removed by adding or removing the designated row and column respectively.

$$
\begin{array}{cc|ccc|ccc|ccc|}
 & & \multicolumn{9}{c}{\text{variables} \longrightarrow} \\
 & & \multicolumn{3}{c}{\overbrace{\qquad\qquad\qquad}^{\text{density}}} & \multicolumn{3}{c}{\overbrace{\qquad\qquad\qquad}^{\text{mass flow}}} & \multicolumn{3}{c}{\overbrace{\qquad\qquad\qquad}^{\text{enthalpy}}} \\
 & & \frac{\partial f_{ma,1}}{\partial \rho_1} & \cdots & \frac{\partial f_{ma,1}}{\partial \rho_n} & \frac{\partial f_{ma,1}}{\partial \dot{m}_1} & \cdots & \frac{\partial f_{ma,1}}{\partial \dot{m}_m} & \frac{\partial f_{ma,1}}{\partial h_1} & \cdots & \frac{\partial f_{ma,1}}{\partial h_n} \\
 & \text{mass} & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
 & & \frac{\partial f_{ma,n}}{\partial \rho_1} & \cdots & \frac{\partial f_{ma,n}}{\partial \rho_n} & \frac{\partial f_{ma,n}}{\partial \dot{m}_1} & \cdots & \frac{\partial f_{ma,n}}{\partial \dot{m}_m} & \frac{\partial f_{ma,n}}{\partial h_1} & \cdots & \frac{\partial f_{ma,n}}{\partial h_n} \\
 & & \frac{\partial f_{mo,1}}{\partial \rho_1} & \cdots & \frac{\partial f_{mo,1}}{\partial \rho_n} & \frac{\partial f_{mo,1}}{\partial \dot{m}_1} & \cdots & \frac{\partial f_{mo,1}}{\partial \dot{m}_m} & \frac{\partial f_{mo,1}}{\partial h_1} & \cdots & \frac{\partial f_{mo,1}}{\partial h_n} \\
\text{equations} \downarrow & \text{momentum} & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
 & & \frac{\partial f_{mo,m}}{\partial \rho_1} & \cdots & \frac{\partial f_{mo,m}}{\partial \rho_n} & \frac{\partial f_{mo,m}}{\partial \dot{m}_1} & \cdots & \frac{\partial f_{mo,m}}{\partial \dot{m}_m} & \frac{\partial f_{mo,n}}{\partial h_1} & \cdots & \frac{\partial f_{mo,m}}{\partial h_n} \\
 & & \frac{\partial f_{en,1}}{\partial \rho_1} & \cdots & \frac{\partial f_{en,1}}{\partial \rho_n} & \frac{\partial f_{en,1}}{\partial \dot{m}_1} & \cdots & \frac{\partial f_{en,1}}{\partial \dot{m}_m} & \frac{\partial f_{en,1}}{\partial h_1} & \cdots & \frac{\partial f_{en,1}}{\partial h_n} \\
 & \text{fluid energy} & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
 & & \frac{\partial f_{en,n}}{\partial \rho_1} & \cdots & \frac{\partial f_{en,n}}{\partial \rho_n} & \frac{\partial f_{en,n}}{\partial \dot{m}_1} & \cdots & \frac{\partial f_{en,n}}{\partial \dot{m}_m} & \frac{\partial f_{en,n}}{\partial h_1} & \cdots & \frac{\partial f_{en,n}}{\partial h_n}
\end{array}
\tag{5-58}
$$

In equation 5-58, n is the number of internal nodes and m is the number of branches.

The partial derivatives, which appear in the Jacobian, are approximated using a two point symmetric derivative:

$$
\frac{\partial f}{\partial x} \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \tag{5-59}
$$

Where f is the residual of the conservation equation, x is the value of the variable at the point where the partial derivative is taken, and Δx is the finite difference term which is taken to be sufficiently small.

In general, the Newton-Raphson method provides the following advantages:

- The Newton-Raphson method converges quadratically, resulting in a significantly lower number of function calls compared to methods which have linear convergence.
- The Newton-Raphson method can be used to simultaneously solve a set of non-linear equations.
- The Newton-Raphson method only needs one initial starting point.

Whereas the method has the following disadvantages:

- The Newton-Raphson method requires the calculation of the Jacobian matrix at every iteration, this means that at every iteration $2n^2$ function calls will be required where n is the number of equations to be solved.

- The Newton-Raphson method required the inverse of the Jacobian matrix to be calculated at every iteration. This becomes more tedious when the number of solved equations becomes large.
- The Newton-Raphson method requires an initial guess which is sufficiently close to the root.
- Local minima and maxima of the function need to be avoided in the area close to the root.
- The function needs to be differentiable.
- If a derivative function is not available, the Newton-Raphson function requires a minimum of 2 function evaluations per derivative.

Broyden's Method

Broyden's method is part of a subsection of methods called quasi-Newton methods. Quasi-Newton methods have the goal of using the strength of the Newton-Raphson method whilst eliminating the downside of computing the Jacobian and/or its inverse at every iteration.

Broyden's method eliminates the need to calculate the Jacobian at every iteration by making an approximation of the new Jacobian based on the new function value using the secant equation (75):

$$\mathbf{J}(\mathbf{x_i} - \mathbf{x_{i-1}}) \simeq \mathbf{f}(\mathbf{x_i}) - \mathbf{f}(\mathbf{x_{i-1}}) \tag{5-60}$$

The Jacobian matrix at the first iteration is constructed similarly to the Newton-Raphson method, see equation 5-58. The Jacobian matrix at each succesive iteration step is defined as a correction to the previous Jacobian (75):

$$\mathbf{J_i} = \mathbf{J_{i-1}} + \frac{\Delta\mathbf{f_i} - \mathbf{J_{i-1}}\Delta\mathbf{x_i}}{||\Delta\mathbf{x_i}||_2^2}\Delta\mathbf{x_i}^{\mathsf{T}} \tag{5-61}$$

Where $\Delta\mathbf{x_i}$ and $\Delta\mathbf{f_i}$ are defined as:

$$\Delta\mathbf{x_i} = \mathbf{x_i} - \mathbf{x_{i-1}} \tag{5-62}$$

$$\Delta\mathbf{f_i} = \mathbf{f}(\mathbf{x_i}) - \mathbf{f}(\mathbf{x_{i-1}}) \tag{5-63}$$

The general procedure of Broyden's method is defined in the following steps:

1. Calculate the first Jacobian matrix, $\mathbf{J_0}$, similarly to the Newton-Raphson method for the starting values of the variables $\mathbf{x_0}$.
2. Calculate the residuals, $\mathbf{f_0}$ using the starting values of the variables, $\mathbf{x_0}$.
3. Calculate the new values of the variables, $\mathbf{x_i}$ using equation 5-55.
4. Caluclate the new values of the residuals, $\mathbf{f_i}$ using the updated variables, $\mathbf{x_i}$.
5. Caluclate the values for $\Delta\mathbf{x_i}$ and $\Delta\mathbf{f_i}$ using equations 5-62 and 5-63 respectively.
6. Calculate the updated Jacobian matrix, $\mathbf{j_i}$ using equation 5-61.

7. Repeat steps 3-6 untill $\mathbf{f_i} < \epsilon$.

Broyden's method has most of the same advantages and disadvantages when compared to the Newton-Raphson method. The main difference is that the Jacobian matrix is obtained without requiring extra function evaluations. The downside is that Broyden's method only converges linearly at best. This means that, in general, more iterations are required in order to arrive at a solution. However, in cases where a large number of equations need to be solved, the time gained by avoiding the evaluation of the Jacobian matrix outweighs the extra iterations that need to be performed. Furthermore, if the starting values of the variables, $\mathbf{x_0}$ is sufficiently far removed from the root, the Jacobian matrix, as calculated by Broyden's method, will differ greatly with respect to the true Jacobian and the solution will diverge.

Modified Broyden's Method

The modified Broyden's method uses a Sherman Morrison formula such that the inverse of the Jacobian is updated instead of the Jacobian itself (75). The advantage of this method is that the matrix inverse operation at each iteration step is avoided. This can potentially save time when dealing with a large number of equations.

The formula of the inverse of the Jacobian at the new iteration step is defined using the Sherman Morrison formula (75):

$$\mathbf{J_i^{-1}} = \mathbf{J_{i-1}^{-1}} + \frac{\mathbf{\Delta x_i} - \mathbf{J_{i-1}^{-1}} \mathbf{\Delta f_i}}{\mathbf{\Delta x_i^T} \mathbf{J_{i-1}^{-1}} \mathbf{\Delta f_i}} \mathbf{\Delta f_i^T} \tag{5-64}$$

Where $\mathbf{\Delta x_i}$ and $\mathbf{\Delta f_i}$ defined by equations 5-62 and 5-63.

Similarly to Broyden's method, the modified Broyden's method requires more iterations to arrive at a solution when comparing it to the Newton-Raphson method. The advantage of the modified Broyden's method is that it avoids the calculation of the inverse of the Jacobian at every iteration.

Regula Falsi Method

The regula falsi method (also known as the false position method) can find the root of any single non-linear continuous function given a bracketed interval. The method assumes that a line connects the two bracketed points. The point where this line equals zero will create a new point and, by doing so, divide the bracketed interval into two. The last step of the regula falsi method is to select which of the two intervals is the next, smaller, bracketed interval.

Given two points $[a, b]$ where a and b bracket the function $f(x)$, the regula falsi iteration scheme defines a new point in between the bracketed region as follows:

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)} \tag{5-65}$$

This gives two new intervals $[a, c]$ and $[c, b]$. The new interval needs to be bracketed to contain a root, therefore the interval which is selected is the interval where either $f(a)f(c) < 1$ or where $f(c)f(b) < 1$. This method is repeated until $f(c) = 0$ or $f(c) < \epsilon$.

The regula falsi method generally has the following advantages:

- Given that the function is continuous and the function can be bracketed, the regula falsi method will always converge to a root.
- Each additional step in the regual falsi method only requires a single function call.

Whereas the method has the following disadvantages:

- The regula falsi method can only solve a single equation at a time.
- The regula falsi method has a slow, linear, rate of convergence.
- The regula falsi method requires the input of two points which need to bracket a root.

It can be concluded that the regula falsi method trades speed for stability when comparing it to the Newton-Raphson method. The method also converges quicker compared to similarly stable methods like the bisection method.

In IPSAT the Regula Falsi method is used in some parts of the thermophysics model where stability is required for a large range of possible functions. More information on this can be found in appendix A. The regula falsi method can be used by the solver module. However, it is not advised since all equations will be solved individually and convergence is not guaranteed.

Brent's Method

Brent's method is a hybrid root finding method, combining different traditional methods into one. The idea behind this combined method is that each (conventional) root finding method is used in their most effective regime. This combines the advantages of each method whilst also negating most disadvantages at the same time. The three different methods which are combined in Brent's method are the bisection method, the secant method and the inverse quadratic interpolation method (76).

Brent's method requires the designer to supply two values $[a, b]$ which brackets the root of the function and the function itself. The first step of Brent's method is to calculate a new value for d. This is done by either using the quadratic interpolation method or the secant method. The quadratic interpolation method used in Brent's method is a Lagrange interpolation function of degree 2:

$$d = \frac{af(b)f(c)}{(f(a)-f(b))(f(a)-f(c))} + \frac{bf(a)f(c)}{(f(b)-f(a))(f(b)-f(c))} + \frac{cf(a)f(b)}{(f(c)-f(a))(f(c)-f(b))} \quad \text{(5-66)}$$

The secant method used in Brent's method is defined as:

$$d = b - f(b)\frac{b-a}{f(b)-f(a)} \quad \text{(5-67)}$$

The quadratic interpolation method, as defined in equation 5-66, will only be used if c is defined and $f(a) \neq f(b \neq f(c)$. In all other cases the secant method will be used to calculate d, as defined by equation 5-67.

The next step is to evaluate if the value of d will be used as is, or if it has to be replaced by the value that is provided by the bisection method which is defined by:

$$d = \frac{a+b}{2} \tag{5-68}$$

To determine which value of d to keep, Brent's method has a specialized heuristic. This heuristic is defined by the following set of decisions:

- If the value of d, given by either the quadratic interpolation method (5-66), or the secant method (5-67), is not between $(3a+b)/4$ and b, then use the value of d given by the bisection method (5-68), otherwise keep the value of d.
- If $|d-b| \geq |c-e|/2$ then use the value of d, given by the bisection method (5-68), otherwise keep the value of d.
- If $|b-c| < |\delta|$ then use the value of d, given by the bisection method (5-68), otherwise keep the value of d.
- If $|c-e| < |\delta|$ then use the value of d, given by the bisection method (5-68), otherwise keep the value of d.

Brent's method is very popular due to its favorable stability and convergence. The philosophy behind Brent's method usually forms the basis for custom methods implemented by different programs. MATLAB has incorporated Brent's method in its preprogrammed `fzero()` function. Brent's method is used in several parts of the IPSAT program to find the root of functions with a single independent variable. Brent's method can also be selected as a method in the solver module, however this is not advised for similar reasons as the regula falsi method.

5-3-2 Conservation Equations

The goal of the solver is to solve a set of equations. The equations that are solved by the solver module are the set of fundamental conservation equations. Each conservation equation determines the remainder of that equation, given that the equation should equal zero, called the residual. These residuals are used by the solver module, which is tasked to minimize these residuals. The conservation equations and the components of each equation can be selected by the designer.

The goal of the minimization of the residuals of the conservation equations is to determine the value of a certain node or branch property. These properties cannot be determined directly by one method/equation and need to be found using an iterative procedure. An overview of these properties and equations is shown in table 5-7.

Table 5-7: The variables solved by the solver module along with the accompanying equations.

Variable	Equation	Solved in
massflow	conservation of fluid mass	internal fluid node
pressure	conservation of fluid momentum	fluid branch
fluid enthalpy	conservation of fluid energy	internal fluid node

The next sections introduce the set of conservations equations and shows how they are implemented in IPSAT.

Conservation of Fluid Mass

The first conservation equation is the conservation of fluid mass. This equation will be solved for every internal fluid node. The conservation of fluid mass of a control volume with a discrete number of input and output flows is given by equation 5-69.

$$\underbrace{\frac{\partial m_{c.v.}}{\partial t}}_{\text{transient term}} = \underbrace{\sum_{\text{branches}} \dot{m}}_{\text{massflow term}} + \underbrace{\sum_{\text{source}} \dot{m}}_{\text{other terms}} \tag{5-69}$$

Where $m_{c.v.}$ is the total mass of the control volume, and $\dot{m}$ is the massflow. In the current version of IPSAT only the transient and the massflow terms are used. Other terms, like a custom source and/or sink term, can be added by creating term function files.

Transient term

The transient term in the conservation of fluid mass equation describes the increase or decrease of fluid mass in the control volume. The increase in mass can be caused by various changes in the fluid and geometric properties of the fluid node. In order to quantify this change, the transient term needs to be rewritten. The total fluid mass in the control volume can be rewritten as a volume integral of the density in the control node:

$$\frac{\partial m_{c.v.}}{\partial t} = \frac{\partial}{\partial t}\left(\oint_{V_{c.v.}} \rho dV\right) \tag{5-70}$$

Where V is the total volume of the control volume. It is assumed that the density distribution inside the control volume is uniform. This means that the density can be taken out of the integral form and can directly be written in differential form:

$$\frac{\partial}{\partial t}\left(\oint_{V_{c.v.}} \rho dV\right) = \frac{\partial \rho}{\partial t} V_{c.v.} \tag{5-71}$$

The equation can be left in this form where density is currently the independent variable which would make the solver a density based solver. In that case the pressure can be obtained by the real equation of state described in section 5-1-1. In a pressure based solver the continuity equation solves for pressure. This can be achieved by substituting the equation of state for a real gas into equation 5-70.

$$\frac{\partial \rho}{\partial t} V_{c.v.} = \frac{\partial}{\partial t}\left(pzT\right)_{c.v.} RV_{c.v.} \tag{5-72}$$

In equation 5-72, z is the compressibility factor, p is the pressure, T is the fluid temperature and R is the specific gas constant. The compressibility factor is obtained from the thermophysics module.

Massflow term

The massflow term determines the mass that is retained by the control volume per unit time due to the mass coming into the control volume and the mass leaving the control volume. The mechanism for mass coming into and leaving the control volume in IPSAT are the fluid branches. The total sum of these masses can therefore be written as:

$$\sum_{\text{branches}} \dot{m} = \sum_{\text{in}} \dot{m} - \sum_{\text{out}} \dot{m} \tag{5-73}$$

Residual form

Rewriting and combining equations 5-72 and 5-73 for discrete time steps and into residual form with density as an independent variable gives:

$$\frac{\rho_{t+\Delta t} - \rho_t}{\Delta t} V - \left(\sum_{\text{in}} \dot{m} - \sum_{\text{out}} \dot{m} \right) = r_{mass} \tag{5-74}$$

The conservation of fluid mass in pressure based form can be written as:

$$\frac{p_{t+\Delta t} z_{t+\Delta t} T_{t+\Delta t} - p_t z_t T_t}{\Delta t} RV - \left(\sum_{\text{in}} \dot{m} - \sum_{\text{out}} \dot{m} \right) = r_{\text{mass}} \tag{5-75}$$

Where the subscript t indicates the value of the property at the current time and Δt is the selected time increment. The value of r_{mass} is equal to the residual of the conservation of mass given the values of the current set of variables. Successive iterations will be necessary to minimize the residual.

Conservation of Fluid Momentum

The second conservation equation is the conservation of fluid momentum. This equation will be solved for every fluid branch in order to determine the massflow in each fluid branch. The conservation of fluid momentum for a control volume is given by equation 5-76.

$$\frac{\partial}{\partial t}(mv) = \sum F \tag{5-76}$$

Where m is the fluid mass, v is the flow velocity and F is a force acting on the control volume. Depending on the preference of the designer, the effect of different kinds of forces on the fluid system can be analyzed. The most common type of force terms in fluid systems, and the ones currently available in IPSAT, are shown in equation 5-77.

$$\underbrace{\frac{\partial}{\partial t}(mv)}_{\text{transient term}} = F_{\text{pressure}} + F_{\text{gravity}} + F_{\text{friction}} + \sum_{\text{other}} F \tag{5-77}$$

This equation can be adjusted depending on the context, similarly to the continuity equation. For example, in case of a horizontal test setup, the force of gravity might play a negligible role compared to other forces. However, in a vertically stacked rocket this might not be the case. In the current version of IPSAT it is assumed that all other forces are zero. However, the developer is free to add more terms which will automatically be included in the solving process.

Transient term

The transient term determines the change in massflow due to a residual force. Note that in a steady state simulation there are no residual forces. The transient term can be rewritten by substituting:

$$v = \frac{\dot{m}}{\rho A}, \quad m = \rho V \tag{5-78}$$

It is assumed that the velocity in the branch is uniform, the density in the branch is uniform and there are no changes in area (the relation $V = AL$ holds) in a branch. Substituting these variables for m and v gives:

$$\frac{\partial}{\partial t}(mv) = \frac{\partial}{\partial t}\left(\rho V \frac{\dot{m}}{\rho A}\right) = \frac{\partial \dot{m}}{\partial t} L \tag{5-79}$$

In equation 5-79, L is the length of the branch, which is assumed to be constant over time.

Pressure term

The pressure force term calculates the pressure force in the fluid branch due to a difference in pressure between the upstream and the downstream node. The pressure force term is written in the following form:

$$F_{\text{pressure}} = (p_{\text{US}} - p_{\text{DS}})\, A \tag{5-80}$$

Where, p_{US} and p_{DS} are the pressure of the upstream and downstream node respectively and A is the cross sectional area of the branch.

Gravity term

The gravity force term calculates the force of gravity in the fluid branch due to a change in elevation between the upstream and the downstream node. The gravity force term is written in the following form:

$$F_{\text{gravity}} = \rho g V \cos(\theta) \tag{5-81}$$

Where ρ is the density (taken from the upstream node), g is the gravitational acceleration and θ is the angle of the flow from horizontal (w.r.t. the gravity field).

Friction term

The friction term calculates the friction force in the fluid branch due to the shape and the surface properties of the fluid branch. The fluid friction force is written in the following form.

$$F_{\text{friction}} = K_f \dot{m}^2 A \tag{5-82}$$

Where K_f is the friction factor of the branch (see section 5-2), A is the cross sectional area of the branch and $\dot{m}$ is the massflow through the branch. It must be noted that the friction force always acts opposite to the direction of the fluid velocity. It is therefore necessary to determine the direction of flow in order to determine the sign of the friction force term.

Residual form

Rewriting equation 5-79 for discrete time steps into residual form, and adding the terms described in equations 5-80 till 5-82, gives:

$$\frac{\dot{m}_{t+\Delta t} - \dot{m}_t}{\Delta t} L - (p_{\text{US}} - p_{\text{DS}})A - \rho g V \cos(\theta) + K_f \dot{m}^2 A = r_{\text{momentum}} \tag{5-83}$$

Where the subscript t indicates the value of the property at the current time and Δt is the selected time increment. The value of r_{momentum} is equal to the residual of the conservation of momentum given the values of the current set of variables. Successive iterations will be necessary to minimize the residual.

Conservation of Fluid Energy

The third conservation equation is the conservation of fluid energy. This equation will be solved for every fluid node in order to calculate the enthalpy in each node. The enthalpy and the density, which is calculated by the continuity equation, will determine all the remaining thermophysical properties of the fluid. This equation is therefore complementary to the determination of the state variables. The conservation of energy in a control volume is given by the first law of thermodynamics:

$$\underbrace{\frac{\partial E}{\partial t}_{c.v.}}_{\text{transient term}} = \underbrace{\sum_{\text{branches}} \dot{E}}_{\text{fluid energy term}} + \underbrace{\sum_{\text{sources}} \dot{Q}_{c.v.}}_{\text{heat flow term}} + \underbrace{\dot{W}_{c.v.}}_{\text{work energy term}} \tag{5-84}$$

Transient term

The total energy of the control volume is defined as:

$$E_{\text{c.v.}} = (H - pV)_{\text{c.v.}} = m_{\text{c.v.}} \left(h_{\text{c.v.}} - \frac{p_{\text{c.v.}}}{\rho_{\text{c.v.}}} \right) \tag{5-85}$$

In equation 5-85, $E_{\text{c.v.}}$ is the total energy of the control volume, $H_{\text{c.v.}}$ is the total enthalpy of the control volume, $p_{\text{c.v.}}$ is the pressure inside the control volume, $V_{\text{c.v.}}$ is the total volume of the control volume, $m_{\text{c.v.}}$ is the mass of the control volume, $h_{\text{c.v.}}$ is the enthalpy per unit mass of the control volume and $\rho_{\text{c.v.}}$ is the density of the control volume. It is assumed that the volume of the control volume does not change (i.e. no work is done by the control volume), that the mass is uniformly distributed (i.e. density is uniform) and that the fluid energy is uniformly distributed in the control volume. Rewriting the transient energy term using equation 5-85 gives:

$$\frac{\partial E}{\partial t}_{\text{c.v}} = \frac{\partial}{\partial t} \left(m \left[h - \frac{p}{\rho} \right] \right) \tag{5-86}$$

Fluid energy term

The fluid energy term describes the energy added to the control volume due to fluid mass added to and subtracted from the control volume. The fluid energy term can be written as a combination of two separate terms:

$$\sum_{\text{branches}} \dot{E}_{\text{fluid energy}} = \sum_{\text{branches}} \dot{E}_{\text{flow}} + \sum_{\text{branches}} \dot{W}_{\text{flow}} \tag{5-87}$$

The flow energy is defined as:

$$\sum_{\text{branches}} \dot{E}_{\text{flow}} = \sum_{\text{in}} \dot{m}e - \sum_{\text{out}} \dot{m}e \tag{5-88}$$

Where $\dot{m}$ is the massflow entering or exiting the control volume through the branches which are connected to the control volume and e is the internal fluid energy per unit mass. The flow work is defined as:

$$\sum_{branches} \dot{W}_{\text{flow}} = \sum_{\text{in}} \dot{m}\frac{p}{\rho} - \sum_{\text{out}} \dot{m}\frac{p}{\rho} \tag{5-89}$$

Where p is the pressure in the control volume and ρ is the density of the control volume. It must be noted that both properties p and ρ for the work done on the control volume are determined from the upstream node.

Combining equations 5-88 and 5-89 into equation 5-87 gives:

$$\sum_{\text{branches}} \dot{E}_{\text{fluid energy}} = \sum_{\text{in}} \dot{m}\left(e + \frac{p}{\rho}\right) - \sum_{\text{out}} \dot{m}\left(e + \frac{p}{\rho}\right) \tag{5-90}$$

Using equation 5-86, the fluid energy term can be rewritten as:

$$\sum_{\text{branches}} \dot{E}_{\text{fluid energy}} = \sum_{\text{in}} \dot{m}h - \sum_{\text{out}} \dot{m}h \tag{5-91}$$

Friction term

The friction work term determines the energy loss in the downstream node due to friction work done by the upstream branch. The friction work term can be written as:

$$\sum_{branches} \dot{W}_{\text{friction}} = \sum_{\text{in}} F_{\text{friction}} v \tag{5-92}$$

Where F_{friction} is defined as $K_f \dot{m}^2 A$. Substituting gives the final form of the friction work term:

$$\sum_{branches} \dot{W}_{\text{friction}} = \sum_{\text{in}} K_f \dot{m}^2 A v \tag{5-93}$$

Where K_f is the friction factor (equal to $\Delta p \dot{m}^{-2}$), v is the velocity of the fluid and A is the area of the branch.

Residual form

Combining all the available terms gives the final form of the conservation of energy relation for unsteady flow in a discretized fluid system currently used in IPSAT:

$$\frac{\partial}{\partial t}\left(m\left[h - \frac{p}{\rho}\right]\right)_{\text{c.v.}} = \sum_{\text{in}} \dot{m}h - \sum_{\text{out}} \dot{m}h + \sum_{in} K_f \dot{m}^2 A v \tag{5-94}$$

Rewriting equation 5-95 for discrete time steps and into residual form gives:

$$\frac{m\left(h - \frac{p}{\rho}\right)_{t+\Delta t} - m\left(h - \frac{p}{\rho}\right)_{t}}{\Delta t} - \sum_{\text{in}} \dot{m}h + \sum_{\text{out}} \dot{m}h - \sum_{in} K_f \dot{m}^2 A v = r_{\text{energy}} \tag{5-95}$$

Where the subscript t indicates the value of the property at the current time and Δt is the selected time increment. The value of r_{energy} is equal to the residual of the conservation of fluid energy equation given the values of the current set of variables. Successive iterations will be necessary to minimize the residual.

5-3-3 Relaxation Factor

A relaxation factor is often used in iterative solving methods in order to control the convergence behavior. This method can be set by the designer in order to change the behavior of the solver. For methods where a variable correction is given as output for each iteration (e.g. Newton-Raphson method) the relaxation factor is defined directly as a factor multiplied by the correction:

$$f^*_{n+1} = f_n + k \cdot \Delta f \tag{5-96}$$

Where f^*_{n+1} is the relaxed variable, f_n is the value of the variable at the current iteration step, Δf is the value of the variable correction step and k is the relaxation factor.

For methods where a variable value is given as output for each iteration (e.g. Brent's method), the relaxation factor is defined as a factor multiplied by the difference between the new value and the previous value:

$$f^*_{n+1} = f_n + k\left(f_{n+1} - f_n\right) \tag{5-97}$$

Where f^*_{n+1} is the relaxed variable, f_{n+1} is the new value of the variable, f_n is the value of the variable at the current iteration step and k is the relaxation factor. The term $f_{n+1} - f_n$ can be seen as the correction step similar to Δf in equation 5-96.

A value of k between 0 and 1 is called under relaxation. This is often required in order to stabilize the solution of a set of non-linear equation between iterations. A relaxation factor between 0 and 1 will prevent the problem to diverge and/or prevent excessive values, of the calculated variables, which cannot be handled by other functions (e.g. the methods used by the thermophysical module).

A value of k larger than 1 is called over relaxation. This used to speed up the convergence and can be used when the problem is shown to have a stable convergence behavior. Note that using a relaxation factor larger than 1 will have the opposite effect on the behavior of the solver compared to using a relaxation factor which is smaller than 1.

5-3-4 Convergence Criteria

The numerical root finding algorithms will, in most cases, only approximate a root. A fixed criteria on convergence provides a clear cutoff value for the root finding methods which avoids unnecessary loop cycles. The convergence criteria defines a non-zero interval in which the root finding problem is assumed to be solved. IPSAT solves multiple equations where each equation needs to meet the convergence criteria. Local convergence is reached when a single variable meets the convergence criteria. This can be simply evaluated given the variable and the convergence criteria of the given variable:

$$|f| < \epsilon \tag{5-98}$$

IPSAT allows the designer to specify the convergence criteria for each of the different equations. The designer can then specify to run the solver until global convergence has been reached, or limit the simulation to keep running for a predetermined number of iterations.

In order to evaluate global convergence for a variable over the entire fluid system, IPSAT evaluates the sum of convergence checks for each node and branch. The convergence for each node specific conservation equation (e.g. conservation of mass, momentum and fluid energy), global convergence is reached if the following condition is met:

$$S_j = \sum_{i=1}^{n} |f_{i,j}| < \epsilon_j \tag{5-99}$$

Where ϵ_j is the convergence criteria for the given conservation equation, n is the number of fluid nodes/branches and $f_{i,j}$ is the residual of equation j in node/branch i. Global convergence over all conservation equations is defined when the following is true:

$$\sum_{j=1}^{n_{eq}} S_j = \sum_{j=1}^{n_{eq}} n_j \tag{5-100}$$

Where n_j is the number of fluid nodes or fluid branches and n_{eq} is the total number of equations which are solved. For a steady state problem n_{eq} is equal to 3. For a transient problem there is an extra transient mass equation added to each node which makes n_{eq} equal to 4.

The default settings for convergence for each variable are listed in table 5-8.

Table 5-8: Standard convergence criteria used in IPSAT.

Equation	ϵ	**Unit**
Mass	10^{-3}	kg·s^{-1}
Momentum	10^{-3}	N
Fluid Energy	10^{-3}	J·s^{-1}

5-3-5 Solver Module Architecture

Figure 5-9 shows the program architecture of the solver module. This module is designed to give full control in the hands of the designer and the developer. The designer can fully customize the solving process by selecting and linking items from each function library. The developer can add new functions to each library without much effort due to the standardized communication of variables.

The **`solver_module()`** function is the main function that controls the solving process. This function runs through the active root finding methods and the **`in_solver`** modules for each iteration. This function also calls **`convergence_check()`** in order he check if convergence has been reached in accordance to the convergence criteria explained in section 5-3-4.

The solver method library contain the root finding methods which can be found in section 5-3-1. These methods get executed each iteration and provide a correction which is applied to the variables.

Each solver method can access the conservation equation library. The conservation equation library contains the three conservation equations (i.e. fluid mass, fluid momentum and fluid energy) which were explained in section 5-3-2. The designer has to specify which solving method gets coupled to which conservation equation.

Each conservation equation is constructed using a set of terms, see section 5-3-2. These terms are collected in the equation terms library. The designer is free to specify which terms to use in which conservation equation.

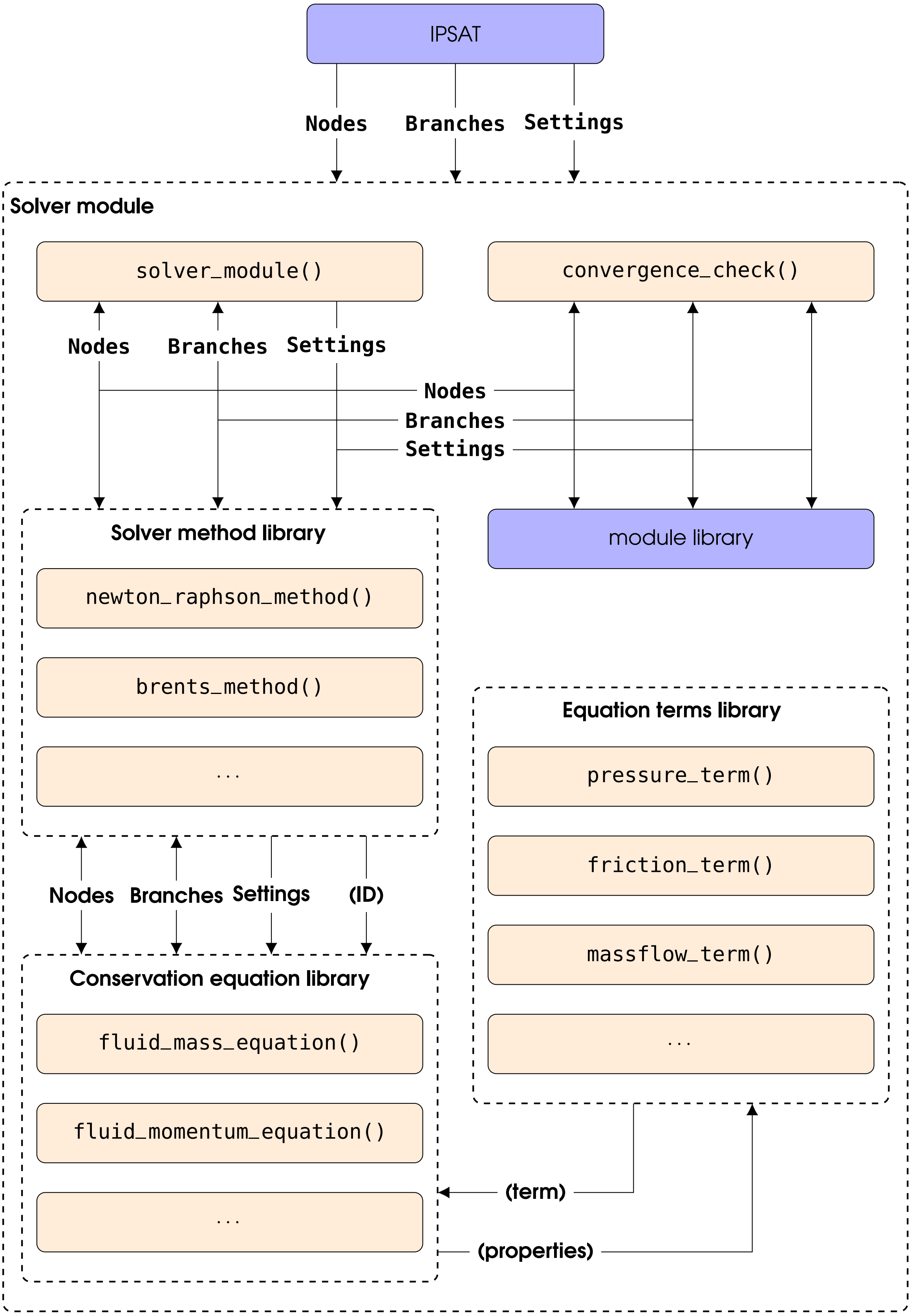


Figure 5-6: Solver module architecture as implemented in IPSAT.

5-4 Fluid System Initialization Module

Solving the system of equations, using the methods presented in section 5-3-1, requires an initial guess to start the procedure in case a steady state solution is required, or an initial value in case a transient solution is required. The fluid system initialization module contains methods which the designer can utilize to provide initial values for the solving process. The initialization process is performed for the two different components of the fluid system, the **`Nodes`** and **`Branches`** data structs.

The fluid system initialization mode requires the following field in the **`Settings`** date structure to be filed:

- **`Settings.fluid_system_initialization`** contains the settings of the fluid system initialization module. This field contains the following two subfields:
  - **`Settings.fluid_system_initialization.methods`**. This field contains the fluid system initialization methods for the **`Nodes`** and **`Branches`** data struct.
  - **`Settings.fluid_system_initialization.functions`**. This field contains the fluid system initialization function calls.

Next to the required settings, there are also other variables that must be specified in order to initialize the fluid system. The following properties might be required depending on the specified initialization method:

- The properties of the boundary fluid nodes have to be submitted in case any of the nodes interpolation methods are used.
- Every fluid node needs to be initialized before the fluid branches can be initialized since the fluid branches initialization method requires the properties of neighboring nodes.

5-4-1 Fluid Nodes Initialization Methods

The function of the fluid nodes initialization methods is to set initial values for the two thermophysical properties in each of the internal fluid nodes. These properties are required to start the solving procedure, as described in section 5-3, and are obtained from the boundary nodes. This section will present the fluid nodes initialization methods which are currently available in IPSAT.

Linear Interpolation Nodes Initialization Method

The linear interpolation nodes initialization method searches for the extremities in pressure and temperature in the boundary fluid nodes and interpolates between those two values to set the values for the internal nodes. The increment in pressure and temperature is calculated using a linear interpolation function given by equation 5-101. From the calculated pressure and temperature, the other thermophysical properties can be determined using the themophysics module. These values in each node are then used as the initial values for the program.

$$\Delta f = \frac{f_{high} - f_{low}}{N_{no} - 1} \tag{5-101}$$

The linear interpolation method will interpolate each of the two initial thermophysical variables separately. All other thermophysical variables will be calculated using the thermophysics module.

Uniform Distribution Nodes Initialization Method

The uniform distribution nodes initialization method allows the designer to set a uniform set of thermophysical variables over a set of internal fluid nodes. The thermophysical values used for this method are taken from an existing boundary node. The designer must specify if the group of nodes are in the same distribution group.

This method is particularly useful for the initialization of transient problems, where the initial state of the system is usually uniform.

Custom Nodes Initialization Method

The designer is also allowed to set the initial values for each of the internal nodes separately. In this case the fluid system initialization method will not determine or adjust any of the values in the nodes. This is initialization method is useful if the designer wants full control of the initial solution. Note that when this method is selected the two thermophysical properties in each internal node needs to be set when adding that node to the fluid system.

5-4-2 Fluid Branches Initialization Methods

The fluid branches initialization methods will set an initial value of the fluid massflow through each of the fluid branches. This section will present the fluid branches initialization methods which are currently available in IPSAT. The fluid massflow is estimated using values of the surrounding nodes. This means that this subroutine needs to be executed after the internal fluid nodes have been initialized.

Bernoulli Branches Initialization Method

The Bernoulli branches initialization method determines the massflow through each branch using the Bernoulli principle for flow through a restriction. The general form of the Bernoulli equation for laminar flow for two points in a straight pipe is given as:

$$p_1 + \frac{1}{2}\rho_1 v_1^2 = p_2 + \frac{1}{2}\rho_2 v_2^2 \tag{5-102}$$

It is assumed that point 1 is the control volume upstream of the branch, and point 2 is taken at the end of the branch next to the downstream node. In this case $v_1 = 0$ and $\rho_2 = \rho_1$ due to the definition of the nodes and branches. The flow velocity in the branch, v_2, can be written as:

$$v = \frac{\dot{m}}{\rho A} \tag{5-103}$$

Where it is assumed that A is uniform over the branch, which is equal to the provided value in the branch geometry variable, and ρ is equal to the density of the upstream node. Substituting equation 5-103 into equation 5-102 gives the following relation for the massflow through the branch:

$$\dot{m} = A\sqrt{2\rho_1(p_1 - p_2)} \tag{5-104}$$

Where 1 is the upstream node and is defined as the node with the highest pressure and 2 is the downstream node.

Custom Branches Initialization Method

Similarly to the custom nodes initialization method, the custom branches initialization method allows the designer to specify the massflow in each fluid branch separately, the fluid system initialization module will not modify or add any values in the fluid branch. In the case that this initialization method is selected, the designer needs to specify the initial value of the massflow in a fluid branch when the fluid branch is added to the fluid system.

5-4-3 Fluid System Initialization Module Architecture

Figure 5-7 shows the program architecture of the fluid system initialization module. The function **`fluid_system_initialization_module()`** is the main function that is called and runs through the fluid system initialization procedure.

There are two function libraries for each type of fluid system element. The fluid nodes initialization function library contains the initialization methods for the fluid nodes. The fluid branches function library contains all the initialization methods for the fluid branches

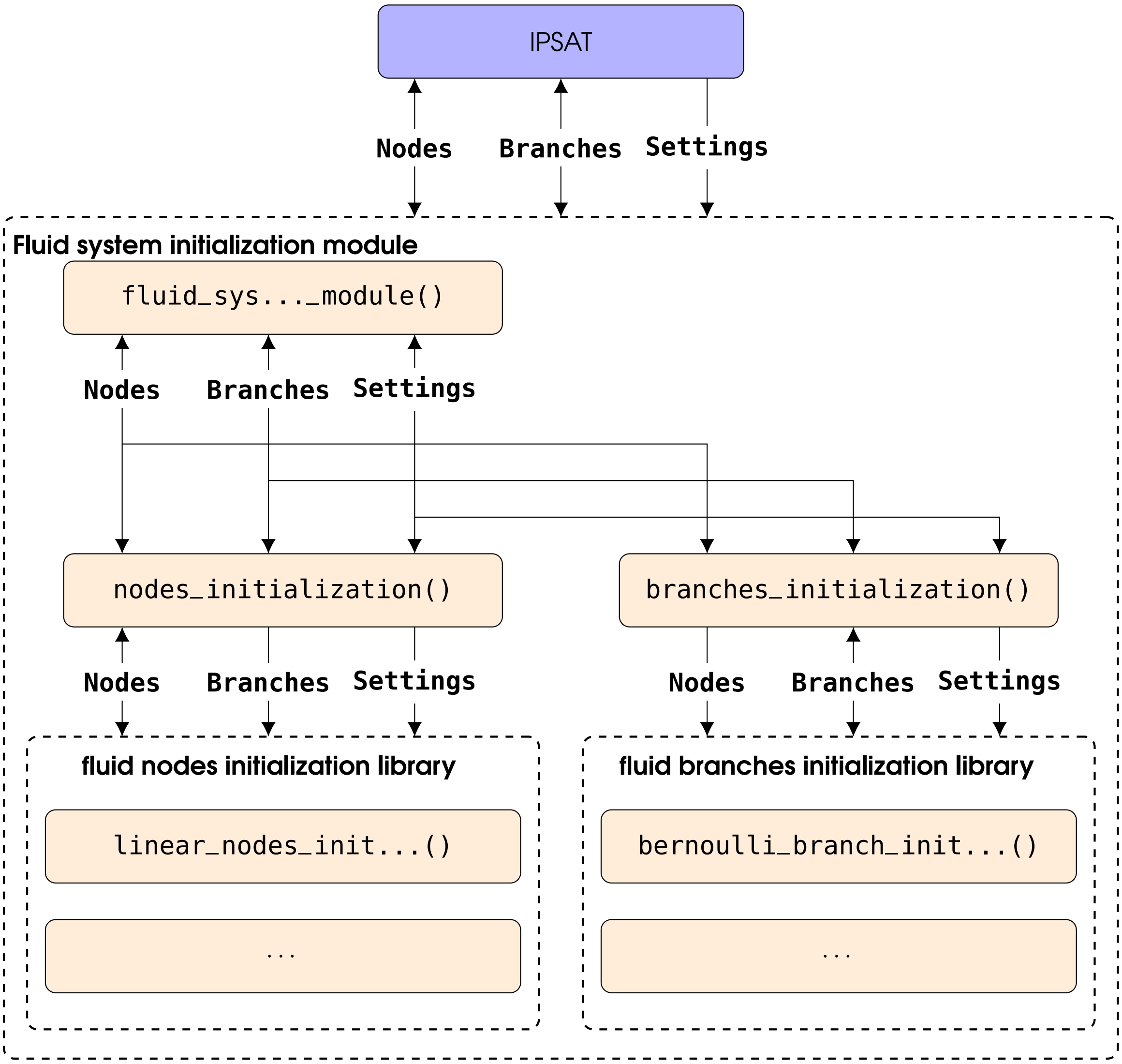


Figure 5-7: Fluid system initialization module architecture as implemented in IPSAT.

5-5 System Initialization Module

The system initialization module is a compulsory module which is executed before all other modules. This module initializes the **`Settings`** data struct and organizes the fluid system described by both the **`Nodes`** and **`Branches`** data structs. The system initialization procedure entails the following list of objectives:

- Loading, interpretation and verification of the provided settings and fluid system architecture.
- Specification of internal folder paths for the specified modules and functions.
- Specification of the order of operations for each of the modules, i.e. the location of execution within the program.
- Specification of the function files in accordance to the methods and models specified by the designer.
- Creation of required data fields used by the selected modules.
- Preloading of the module data files.

The system initialization module also contains the data files which specify the model constants. These files are preloaded and the data content is submitted into a dedicated data field in order to speed up the lookup process. Next to these module specific data files, the system initialization module also contains the data files containing the default settings to be used in case the required settings are not specified by the designer. Lastly, the system initialization module also contains the data file which specifies the program settings defined by the designer.

5-5-1 Load Settings

The settings input file specifies the values to be submitted in the respective data fields of the **`Settings`** data struct. With this file the designer can customize the program to fit the analysis which is being performed. For example, the designer can specify which modules are used in the solver, which solving methods are used by the solver and select the thermophysical models. Initialization of the settings is a procedure that is executed first, because every other procedure in the program depends on the settings. An example of a setting input file is shown in figure 5-8.

The settings initialization file is structured in such a way that it can accommodate a large array of different inputs. The designer can specify the data fields in the **`Settings`** data struct on the left side of the ‘=’ symbol. The values are entered on the right side of the ‘=’ symbol between square brackets ‘(‘ and ‘)’. Multiple entries are entered with a comma ‘,’ as a delimiter. Numbers are automatically registered as numbers whilst text is interpreted as strings. Comments can be added with the ‘%’ symbol put in front of the text. These inputs will not be read by the interpreter.

```
Settings initialization file

%% system initialization module settings

system_initialization-modules-fluid_system_initialization_module-type = (presolver)
system_initialization-modules-solver_module-type = (solver)
system_initialization-modules-fluid_friction_module-type = (in_solver)
system_initialization-modules-thermophysics_module-type = (in_solver)

system_initialization-initialization-system = (modules_system, fluid_system)
system_initialization-initialization-modules = (solver_module, thermophysics_module,
fluid_friction_module)

%% solver module settings

solver-mode = (steady_state)

solver-iteration-min = (0)
solver-iteration-max = (0)

solver-system-equations = (fluid_mass, fluid_momentum, fluid_energy)
solver-system-variables = (pressure, massflow, enthalpy)

% fluid mass settings
solver-equations-fluid_mass-method = (newton_raphson)
solver-equations-fluid_mass-terms = (massflow)
solver-equations-fluid_mass-location = (internal_fluid_node)
solver-equations-fluid_mass-convergence = (1e-3)
solver-equations-fluid_mass-divergence = (1e6)

% fluid momentum settings
solver-equations-fluid_momentum-method = (newton_raphson)
solver-equations-fluid_momentum-terms = (pressure, friction)
solver-equations-fluid_momentum-location = (fluid_branch)
solver-equations-fluid_momentum-convergence = (1e-3)
solver-equations-fluid_momentum-divergence = (1e6)

% fluid energy settings
solver-equations-fluid_energy-method = (brents)
solver-equations-fluid_energy-terms = (massflow, friction)
solver-equations-fluid_energy-location = (internal_fluid_node)
solver-equations-fluid_energy-convergence = (1e-3)
solver-equations-fluid_energy-divergence = (1e6)
```

Figure 5-8: Example of a settings initialization input file.

5-5-2 System Initialization

The system initialization module will run through all the indicated system initialization functions. These functions are essential to the formatting of the data structs and functioning of the program.

Fluid System verification

The fluid system verification procedure verifies the data fields of the **`Nodes`** and **`Branches`** data structs. In case parts of data in the structs are missing the designer is informed and the values are calculated from the existing values where possible, or predetermined default settings are substituted.

The fluid system verification procedure also creates the **system** subfield in the **Nodes** and **Branches** data structs. This field stores the information of the entire fluid system and is required for the program to effectively interact with the fluid system.

module system verification

The module system verification procedure submits the module settings in the **modules** subfield in the **Settings** data structure. The module settings inputs are verified and adjusted where needed. The module system verification function also adds the function name and folder structure to the **Settings.modules** data structure.

5-5-3 Module initialization

The module initialization function library contains all the module specific initialization functions. These functions will check whether the input by the designer is correctly identified and will specify the module specific fields. The module specific initialization functions will also be able to load any required data files. For example, the thermophysics module initialization procedure will load the model constants of the thermophysical models in the thermophysics module initialization function.

In the case that the developer is designing a new module, a module initialization function needs to be added to this library in case the module requires module specific inputs/-data. The developer is allowed to create data folders which are structured according to the developer's preferences.

Each module is allowed to create its own subfield in the **Settings** data structure which is identified by **Settings.(module)**. The contents of this subfield are independent of the other module settings and can be customized by the developer.

5-5-4 System Initialization Module Architecture

The system initialization module program architecture is presented in figure 5-9.
system_initialization_module() is the first function that is called in the system initialization module. This function runs through all the other functions and function libraries. The first function that is called is the **load_settings()** function. This function loads the program settings from a designer defined settings input file, as was presented in section 5-5-1.

The first function library which is called is the system initialization library. This library contains the system initialization functions as presented in section 5-5-2. The second function library which is called is the module initialization library. This includes the functions described in section 5-5-3.

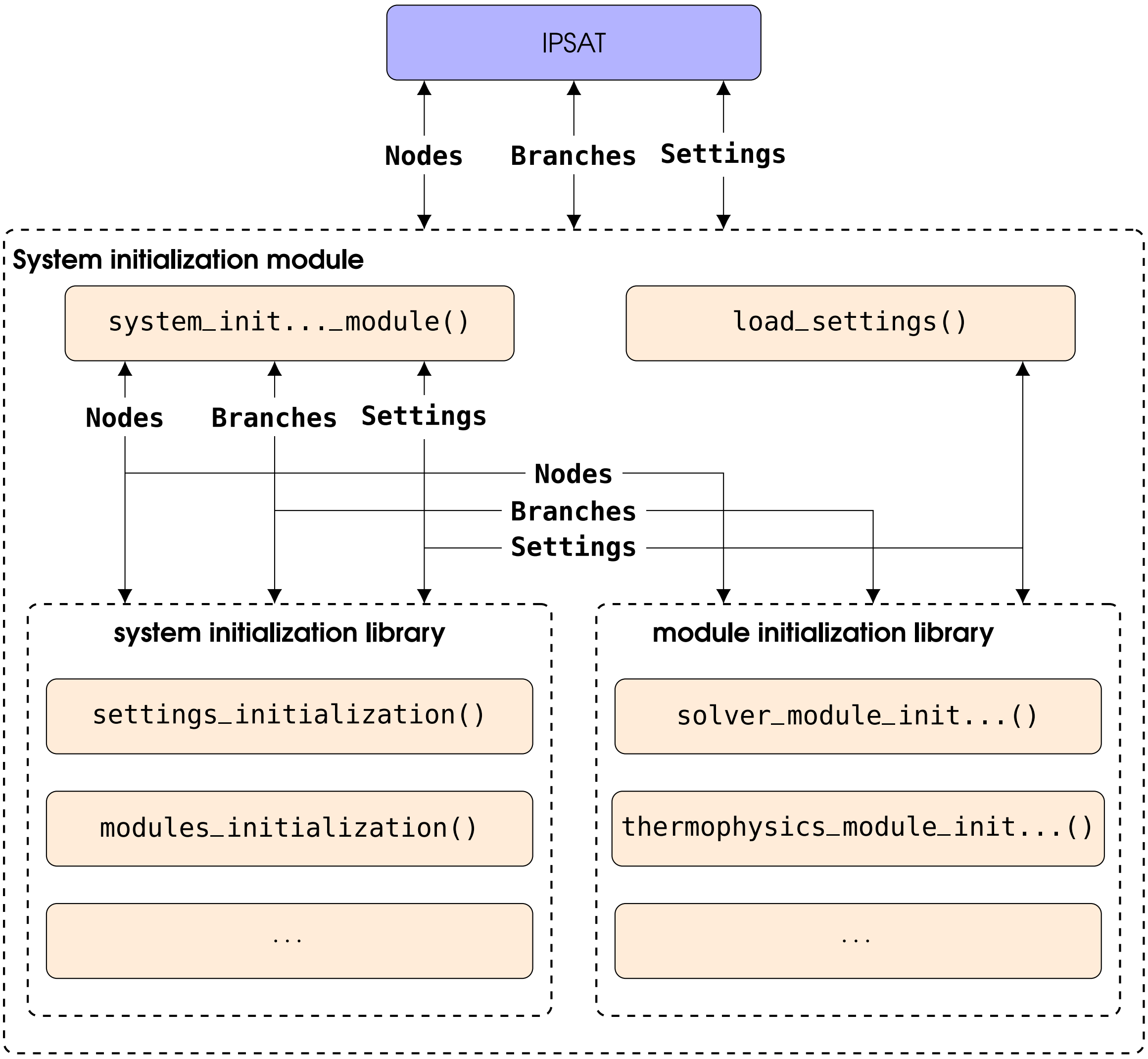


Figure 5-9: System initialization module architecture as implemented in IPSAT.

Chapter 6

Computer Program Verification

The functionality requirement, as presented in section 2-3, states that the program shall be verified. The verification process is divided into two steps corresponding to the verification requirements listed in table 2-5. The first step is to verify each module individually. The second step is to verify the IPSAT program as a whole. This chapter will present the verification process which is performed for the different modules in the program.

6-1 Module Verification

Requirement 5.1.1, presented in section 2-4, dictates that each module shall be verified individually for functionality. This section presents the verification procedure for the following modules:

- Thermophysics module verification, presented in section 6-1-1.
- Fluid friction module verification, presented in section 6-1-2.
- Solver module verification, presented in section 6-1-3.
- Fluid system initialization module verification, presented in section 6-1-4

The process of verifying individual modules is a practice that should be continued when the program is extended and new modules are added.

6-1-1 Thermophysics Module Verification

The goal of the thermophysics verification procedure is to verify that the implemented models, described in section 5-1, are implemented correctly. In order to confirm this, the output of the thermophysics module is compared to the published data for a number of data points. The verification is divided into two parts, the verification of the equation of state and the verification of the other thermophysical properties.

The equation of state part of the thermophysics module is verified on a first degree by comparing the result to the published thermophysical property tables. These tables are generally published in the same source paper as the Helmholtz constants for each fluid and are created using the same process as described in section 5-1. To do this effectively, a predetermined set of input parameters (values for p, and T) is created for each fluid which should form a representable verification data set. The goal is to verify that the program can reproduce the results of the equation of state to within the provided accuracy of the source papers.

The verification of the other thermophysical properties like thermal conductivity and viscosity are more dependent on the available data provided by the source of the model. The verification process for these thermophysical properties are customized to the available, often limited, data set. Often, source papers report a small table with data to aid in implementation of their model. The verification process will directly compare these data and the goal is to verify that the program can reproduce these results to within the provided accuracy of these source papers.

Equation of State Verification Data Set

The input data set for the verification of the equation of state contains data for each fluid region. These regions are: the vapour region, the gas region, the liquid region and the supercritical region. Additionally, the verification data set contains data for two areas on the vapour pressure curve. These areas are: four points close to the critical point and two points close to the triple point. Verifying the results from these points for each implemented fluid will give a good indication if the methodology is correct. Table 6-1 shows the list of data points used to verify the program results compared to a reference source.

Table 6-1: Definition of the data points used to verify the program results.

Point (#)	Phase	Definition
	Data surrounding the critical point	
1	Vapour	$p = 0.9p_c$, $T = 0.9T_c$
2	Liquid	$p = 1.1p_c$, $T = 0.9T_c$
3	Super critical	$p = 1.1p_c$, $T = 1.1T_c$
4	Gas	$p = 0.9p_c$, $T = 1.1T_c$
	Data surrounding the triple point	
5	Liquid	$p = 1.1p_{tp}$, $T = 1.1T_{tp}$
6	Vapour	$p = 0.9p_{tp}$, $T = 1.1T_{tp}$
	Data in each fluid region	
7	Vapour	$p = 0.5p_c$, $T = T_{tp} + 0.5(T_c - T_{tp})$
8	Liquid	$p = 1.5p_c$, $T = T_{tp} + 0.5(T_c - T_{tp})$
9	Super critical	$p = 1.5p_c$, $T = 1.5T_c$
10	Gas	$p = 0.5p_c$, $T = 1.5T_c$
	Data in the far field	
11	Super critical	$p = 10p_c$, $T = 2T_c$
12	Gas	$p = 0.5p_c$, $T = 2T_c$

Reference tables, which are provided at the back of reference fluid papers, have fixed steps in pressure and temperature. The verification procedure will round the data points from table 6-1 to the closest data point in the reference table. Verification of the program with other reference software can be done as is and no rounding will be done.

If there are differences between the calculated data and the reference data which are larger than the reported accuracy of the source paper, the variation of these differences across these data points may help to indicate the source of the discrepancy. This is useful when it comes to adding new fluids and verifying the implementation. Several examples of how the discrepancies in the results can be used to identify the source of these errors are:

- The fluid density matches the reference density at all points, however the other fluid properties show a mismatch. In this case the residual Helmholtz function and its coefficients are implemented correctly. Either the ideal Helmholtz function and/or its coefficients must contain errors.
- The properties vary significantly in one or more points close to the vapour pressure curve (points 1 to 6), but match the reference data in all other points. In this case the program predicts the wrong state of the fluid by submitting a wrong guess for the density. The error can be found in either the implementation of the function for the saturated liquid density, or in the coefficients which are used in this function.
- The fluid entropy shows a significant mismatch compared to the reference data, but all the other fluid properties match the reference data. In this case both the residual and ideal Helmholtz derivatives with respect to density and temperature are correct. However, the actual value of the residual and/or ideal Helmholtz energy is causing the value of the fluid entropy to deviate. In this case the error can be found in the coefficients in the residual and/or the ideal Helmholtz energy function that are a function of neither pressure nor temperature.
- The fluid properties show a significant mismatch close to the critical point and in the liquid and vapour region, and a small mismatch in the far field. This indicates a good implementation of the ideal part of the Helmholtz energy function and its coefficients. The source of the discrepancy can be found in the residual part of the Helmholtz energy formulation and/or coefficients.

Verification Results

For each implemented fluid, the output data at each point listed in table 6-1 is compared to the reference data tables if available. For fluids with no reference table, the data points are compared to the outcome of programs which have implemented the same method. The percentage difference between the calculated data and reference data is tabulated and reported in a verification report for each fluid. An example of this process for the density of nitrogen is given in table 6-2.

Table 6-2: Verification of the calculated density of nitrogen compared to the reference data published by Span et al. (61).

#	p	T	ρ_{calc}	ρ_{ref}	$(\rho_{\text{calc}} - \rho_{\text{ref}})/\rho_{\text{ref}}$
	(MPa)	(K)	(kg·m^{-3})	(kg·m^{-3})	(%)
1	3.0	115	$5.9704\cdot10^{2}$	$5.9705\cdot10^{2}$	$-2.1\cdot10^{-3}$
2	3.5	115	$6.0401\cdot10^{2}$	$6.0400\cdot10^{2}$	$1.4\cdot10^{-3}$
3	3.5	140	$1.2099\cdot10^{2}$	$1.2099\cdot10^{2}$	$5.2\cdot10^{-4}$
4	3.0	140	$9.6409\cdot10^{1}$	$9.6408\cdot10^{1}$	$7.4\cdot10^{-4}$
5	0.1	70	$8.3864\cdot10^{2}$	$8.3864\cdot10^{2}$	$3.3\cdot10^{-4}$
6	0.1	70	$8.3864\cdot10^{2}$	$8.3864\cdot10^{2}$	$3.3\cdot10^{-4}$
7	1.5	95	$7.2294\cdot10^{2}$	$7.2294\cdot10^{2}$	$-1.0\cdot10^{-4}$
8	5.0	95	$7.3804\cdot10^{2}$	$7.3804\cdot10^{2}$	$-9.9\cdot10^{-5}$
9	5.0	190	$1.0098\cdot10^{2}$	$1.0097\cdot10^{2}$	$1.8\cdot10^{-3}$
10	1.5	190	$2.7668\cdot10^{1}$	$2.7668\cdot10^{1}$	$9.3\cdot10^{-4}$
11	25	250	$3.1942\cdot10^{2}$	$3.1941\cdot10^{2}$	$2.6\cdot10^{-3}$
12	1.5	250	$2.0444\cdot10^{1}$	$2.0444\cdot10^{1}$	$-6.0\cdot10^{-5}$

Nitrogen

Table 6-3 shows the equation of state verification results for nitrogen. The results of all properties over all the data points are within the accuracy of provided by the reference paper.

Table 6-3: Percentage difference between the calculated data and reference data published by Span et al. (61) for nitrogen at each data point specified by table 6-1.

#	ρ_{er} (%)	u_{er} (%)	h_{er} (%)	s_{er} (%)	$c_{\text{v,er}}$ (%)	$c_{\text{p,er}}$ (%)	w_{er} (%)
1	$-2.1\cdot10^{-3}$	$1.4\cdot10^{-2}$	$1.9\cdot10^{-2}$	$-5.9\cdot10^{-3}$	$-2.7\cdot10^{-3}$	$-5.7\cdot10^{-3}$	$4.2\cdot10^{-3}$
2	$1.4\cdot10^{-3}$	$1.8\cdot10^{-2}$	$1.9\cdot10^{-2}$	$-2.2\cdot10^{-3}$	$-1.4\cdot10^{-2}$	$-1.1\cdot10^{-2}$	$8.8\cdot10^{-3}$
3	$5.2\cdot10^{-4}$	$-1.5\cdot10^{-2}$	$-1.1\cdot10^{-2}$	$-1.2\cdot10^{-4}$	$-2.4\cdot10^{-2}$	$-3.8\cdot10^{-2}$	$-1.5\cdot10^{-2}$
4	$7.4\cdot10^{-4}$	$-1.4\cdot10^{-2}$	$-1.0\cdot10^{-2}$	$-5.1\cdot10^{-3}$	$-3.5\cdot10^{-2}$	$-1.7\cdot10^{-2}$	$-1.2\cdot10^{-2}$
5	$3.3\cdot10^{-4}$	$2.3\cdot10^{-3}$	$1.3\cdot10^{-3}$	$-1.3\cdot10^{-3}$	$-6.4\cdot10^{-3}$	$-4.5\cdot10^{-3}$	$-1.4\cdot10^{-3}$
6	$3.3\cdot10^{-4}$	$2.3\cdot10^{-3}$	$1.3\cdot10^{-3}$	$-1.3\cdot10^{-3}$	$-6.4\cdot10^{-3}$	$-4.5\cdot10^{-3}$	$-1.4\cdot10^{-3}$
7	$-1.0\cdot10^{-4}$	$5.0\cdot10^{-3}$	$4.2\cdot10^{-3}$	$-2.2\cdot10^{-3}$	$-1.6\cdot10^{-2}$	$-2.6\cdot10^{-3}$	$4.9\cdot10^{-3}$
8	$-9.9\cdot10^{-4}$	$5.2\cdot10^{-3}$	$6.3\cdot10^{-3}$	$-2.4\cdot10^{-3}$	$-6.1\cdot10^{-3}$	$-1.1\cdot10^{-2}$	$-3.1\cdot10^{-3}$
9	$1.8\cdot10^{-3}$	$-1.6\cdot10^{-2}$	$-1.3\cdot10^{-2}$	$-5.5\cdot10^{-4}$	$-1.5\cdot10^{-2}$	$-1.3\cdot10^{-2}$	$2.0\cdot10^{-2}$
10	$9.3\cdot10^{-4}$	$-1.6\cdot10^{-2}$	$-1.1\cdot10^{-2}$	$-5.0\cdot10^{-3}$	$-4.2\cdot10^{-2}$	$-1.4\cdot10^{-2}$	$-5.8\cdot10^{-3}$
11	$2.6\cdot10^{-3}$	$-2.9\cdot10^{-2}$	$-1.8\cdot10^{-2}$	$-4.3\cdot10^{-3}$	$-2.7\cdot10^{-2}$	$-1.9\cdot10^{-2}$	$1.9\cdot10^{-2}$
12	$-6.0\cdot10^{-5}$	$-2.0\cdot10^{-2}$	$-1.4\cdot10^{-2}$	$-1.2\cdot10^{-3}$	$-2.6\cdot10^{-2}$	$-3.0\cdot10^{-2}$	$-5.9\cdot10^{-3}$

Table 6-4 show the verification of the viscosity and thermal conductivity of nitrogen for a data set provided by Lemmon (47). The value of the viscosity and thermal conductivity is shown to be within the accuracy of the given data set.

Table 6-4: Verification of the calculated viscosity (η) and the calculated thermal conductivity (λ) of nitrogen. Both variables are compared to reference data published by Lemmon et al. (47).

#	T (K)	ρ (mol·dm^{-3})	η_{ref} (Pa·s)	η_{er} (%)	λ_{ref} (W·m^{-1}·K^{-1})	λ_{er} (%)
1	100.0	0.0	$6.90349 \cdot 10^{-6}$	$2.2 \cdot 10^{-3}$	$9.27749 \cdot 10^{-3}$	$3.0 \cdot 10^{-3}$
2	300.0	0.0	$1.78771 \cdot 10^{-5}$	$3.8 \cdot 10^{-4}$	$2.59361 \cdot 10^{-2}$	$9.8 \cdot 10^{-4}$
3	100.0	25.0	$7.97418 \cdot 10^{-5}$	$-8.9 \cdot 10^{-3}$	$1.03834 \cdot 10^{-1}$	$-4.5 \cdot 10^{-4}$
4	200.0	10.0	$2.10810 \cdot 10^{-5}$	$-1.0 \cdot 10^{-5}$	$3.60099 \cdot 10^{-2}$	$-6.2 \cdot 10^{-4}$
5	300.0	5.0	$2.07430 \cdot 10^{-5}$	$9.5 \cdot 10^{-5}$	$3.27694 \cdot 10^{-2}$	$-2.3 \cdot 10^{-4}$
6	126.2	11.2	$1.82978 \cdot 10^{-5}$	$-3.6 \cdot 10^{-4}$	$6.75800 \cdot 10^{-1}$	$8.2 \cdot 10^{-3}$

Oxygen

Table 6-5 shows the equation of state verification results for oxygen. It can be seen that the differences between the reference data show slightly higher deviations when comparing it to other fluids, like nitrogen, over the entire data range. It is most likely related to the calculation of the ideal-gas component of the reduced Helmholtz energy. The reference data published by Steward (64) uses an integral form to calculate the value for the idea-gas component of the reduced Helmholtz energy , see equation 5-2. IPSAT evaluated the integral using a MATLAB in-built discrete integrator. The accuracy of this method depends on the selected discretization interval and is not reported by Steward. Furthermore, the derivatives of the ideal-gas component of the reduced Helmholtz energy cannot be calculated directly and are evaluated using a two point symmetric derivative, see equation 5-59. This only aggravates the deviation problem which is created by the integral form.

Table 6-5: Percentage difference between the calculated data and reference data published by Steward et al. (64) for oxygen at each data point specified by table 6-1.

#	ρ_{er} (%)	u_{er} (%)	h_{er} (%)	s_{er} (%)	$c_{\text{v,er}}$ (%)	$c_{\text{p,er}}$ (%)	w_{er} (%)
1	$-7.0 \cdot 10^{-4}$	$2.7 \cdot 10^{-2}$	$2.9 \cdot 10^{-2}$	$-1.5 \cdot 10^{-3}$	$-1.3 \cdot 10^{-2}$	$-3.3 \cdot 10^{-3}$	$1.3 \cdot 10^{-1}$
2	$1.0 \cdot 10^{-3}$	$2.6 \cdot 10^{-2}$	$3.4 \cdot 10^{-2}$	$-5.4 \cdot 10^{-3}$	$-1.2 \cdot 10^{-2}$	$-3.2 \cdot 10^{-3}$	$1.8 \cdot 10^{-1}$
3	$-3.0 \cdot 10^{-3}$	$-1.4 \cdot 10^{-2}$	$-1.0 \cdot 10^{-2}$	$-4.1 \cdot 10^{-3}$	$8.0 \cdot 10^{-3}$	$6.8 \cdot 10^{-3}$	$3.6 \cdot 10^{-1}$
4	$-2.6 \cdot 10^{-3}$	$-1.2 \cdot 10^{-2}$	$-8.4 \cdot 10^{-2}$	$-3.4 \cdot 10^{-3}$	$5.3 \cdot 10^{-3}$	$9.7 \cdot 10^{-3}$	$4.2 \cdot 10^{-1}$
5	$-6.2 \cdot 10^{-4}$	$9.4 \cdot 10^{-3}$	$9.4 \cdot 10^{-3}$	$-9.7 \cdot 10^{-3}$	$4.7 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$3.5 \cdot 10^{-2}$
6	$-6.2 \cdot 10^{-4}$	$9.4 \cdot 10^{-3}$	$9.4 \cdot 10^{-3}$	$-9.7 \cdot 10^{-3}$	$4.7 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$3.5 \cdot 10^{-2}$
7	$-1.3 \cdot 10^{-3}$	$1.1 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$9.9 \cdot 10^{-4}$	$-2.3 \cdot 10^{-2}$	$3.6 \cdot 10^{-3}$	$5.8 \cdot 10^{-2}$
8	$3.0 \cdot 10^{-4}$	$1.0 \cdot 10^{-2}$	$1.0 \cdot 10^{-2}$	$-2.9 \cdot 10^{-3}$	$-3.0 \cdot 10^{-3}$	$7.3 \cdot 10^{-3}$	$4.0 \cdot 10^{-2}$
9	$-2.3 \cdot 10^{-3}$	$-2.8 \cdot 10^{-3}$	$-2.8 \cdot 10^{-3}$	$-3.1 \cdot 10^{-3}$	$2.1 \cdot 10^{-2}$	$1.3 \cdot 10^{-2}$	$3.0 \cdot 10^{-1}$
10	$1.1 \cdot 10^{-3}$	$-2.5 \cdot 10^{-3}$	$-2.5 \cdot 10^{-3}$	$2.0 \cdot 10^{-3}$	$2.7 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$8.4 \cdot 10^{-2}$
11	$-3.1 \cdot 10^{-3}$	$-2.1 \cdot 10^{-3}$	$-2.1 \cdot 10^{-3}$	$2.6 \cdot 10^{-3}$	$2.3 \cdot 10^{-2}$	$-1.8 \cdot 10^{-3}$	$8.1 \cdot 10^{-2}$
12	$-4.1 \cdot 10^{-3}$	$-6.6 \cdot 10^{-4}$	$-6.6 \cdot 10^{-4}$	$1.3 \cdot 10^{-3}$	$1.7 \cdot 10^{-2}$	$1.1 \cdot 10^{-2}$	$1.4 \cdot 10^{-1}$

Table 6-6 show the verification of the viscosity and thermal conductivity of nitrogen for a data set provided by Lemmon (47). The value of the viscosity is shown to be within the accuracy of the given data set. The value of the thermal conductivity is within the accuracy of the given data set for data points 1-5 and show a deviation for data point 6. The source of this discrepancy is most likely linked to the sensitivity of the evaluation of the partial derivative $\partial p/\partial \rho$ at the critical point. The value of this derivative is used to determine the critical enhancement term of thermal conductivity, see section 5-1-8.

Table 6-6: Verification of the calculated viscosity (η) and the calculated thermal conductivity (λ) of oxygen. Both variables are compared to reference data published by Lemmon et al. (47).

#	T	ρ	η_{ref}	η_{er}	λ_{ref}	λ_{er}
	(K)	(mol·dm^{-3})	(Pa·s)	(%)	(W·m^{-1}·K^{-1})	(%)
1	100.0	0.0	$7.70243\cdot 10^{-6}$	$2.9\cdot 10^{-4}$	$8.94334\cdot 10^{-3}$	$3.9\cdot 10^{-3}$
2	300.0	0.0	$2.06307\cdot 10^{-5}$	$8.0\cdot 10^{-4}$	$2.64403\cdot 10^{-2}$	$1.3\cdot 10^{-3}$
3	100.0	25.0	$1.72136\cdot 10^{-4}$	$-1.2\cdot 10^{-2}$	$1.46044\cdot 10^{-1}$	$-6.8\cdot 10^{-3}$
4	200.0	10.0	$2.24445\cdot 10^{-5}$	$-1.5\cdot 10^{-3}$	$3.46124\cdot 10^{-2}$	$-1.3\cdot 10^{-3}$
5	300.0	5.0	$2.37577\cdot 10^{-5}$	$-4.7\cdot 10^{-4}$	$3.25491\cdot 10^{-2}$	$-6.1\cdot 10^{-4}$
6	154.6	13.6	$2.47898\cdot 10^{-5}$	$-2.4\cdot 10^{-3}$	$3.77476\cdot 10^{-1}$	$1.1\cdot 10^{-1}$

Argon

Table 6-7 shows the equation of state verification results for argon. The results of all properties over all the data points are within the accuracy of provided by the reference paper.

Table 6-7: Percentage difference between the calculated data and reference data published by Tegeler et al. (48) for argon at each data point specified by table 6-1.

#	ρ_{er} (%)	u_{er} (%)	h_{er} (%)	s_{er} (%)	$c_{\text{v,er}}$ (%)	$c_{\text{p,er}}$ (%)	w_{er} (%)
1	$-2.0\cdot 10^{-4}$	$-5.9\cdot 10^{-4}$	$-2.0\cdot 10^{-3}$	$-7.1\cdot 10^{-4}$	$2.4\cdot 10^{-4}$	$-2.4\cdot 10^{-3}$	$-4.6\cdot 10^{-4}$
2	$2.9\cdot 10^{-4}$	$6.1\cdot 10^{-4}$	$-2.8\cdot 10^{-3}$	$6.2\cdot 10^{-4}$	$-7.3\cdot 10^{-4}$	$-1.6\cdot 10^{-3}$	$-3.6\cdot 10^{-4}$
3	$2.3\cdot 10^{-3}$	$-4.2\cdot 10^{-3}$	$3.5\cdot 10^{-3}$	$1.7\cdot 10^{-3}$	$-9.0\cdot 10^{-5}$	$4.1\cdot 10^{-4}$	$-1.3\cdot 10^{-3}$
4	$-1.8\cdot 10^{-3}$	$-3.2\cdot 10^{-3}$	$-3.1\cdot 10^{-4}$	$4.4\cdot 10^{-4}$	$-1.3\cdot 10^{-3}$	$-4.1\cdot 10^{-4}$	$-2.6\cdot 10^{-3}$
5	$1.2\cdot 10^{-3}$	$1.3\cdot 10^{-3}$	$-4.1\cdot 10^{-3}$	$4.1\cdot 10^{-4}$	$5.6\cdot 10^{-4}$	$6.0\cdot 10^{-5}$	$-7.4\cdot 10^{-4}$
6	$1.2\cdot 10^{-3}$	$1.3\cdot 10^{-3}$	$-4.1\cdot 10^{-3}$	$4.1\cdot 10^{-4}$	$5.6\cdot 10^{-4}$	$6.0\cdot 10^{-5}$	$-7.4\cdot 10^{-4}$
7	$-1.3\cdot 10^{-4}$	$6.6\cdot 10^{-4}$	$3.2\cdot 10^{-4}$	$-5.9\cdot 10^{-4}$	$-1.6\cdot 10^{-3}$	$2.0\cdot 10^{-3}$	$3.6\cdot 10^{-4}$
8	$-2.8\cdot 10^{-4}$	$-1.2\cdot 10^{-3}$	$-1.5\cdot 10^{-3}$	$1.5\cdot 10^{-3}$	$-1.2\cdot 10^{-3}$	$-4.3\cdot 10^{-3}$	$-7.1\cdot 10^{-4}$
9	$1.2\cdot 10^{-3}$	$-3.3\cdot 10^{-3}$	$4.7\cdot 10^{-4}$	$1.5\cdot 10^{-3}$	$-1.7\cdot 10^{-3}$	$-1.1\cdot 10^{-4}$	$-1.5\cdot 10^{-3}$
10	$-4.0\cdot 10^{-5}$	$-4.3\cdot 10^{-4}$	$6.5\cdot 10^{-4}$	$-8.8\cdot 10^{-4}$	$1.5\cdot 10^{-4}$	$-2.7\cdot 10^{-4}$	$1.3\cdot 10^{-3}$
11	$1.2\cdot 10^{-4}$	$3.7\cdot 10^{-3}$	$-8.4\cdot 10^{-4}$	$-9.6\cdot 10^{-4}$	$-1.5\cdot 10^{-3}$	$-1.0\cdot 10^{-3}$	$8.9\cdot 10^{-4}$
12	$1.2\cdot 10^{-3}$	$-7.0\cdot 10^{-5}$	$7.8\cdot 10^{-4}$	$-7.4\cdot 10^{-4}$	$1.0\cdot 10^{-4}$	$-3.2\cdot 10^{-4}$	$-2.0\cdot 10^{-4}$

Table 6-8 show the verification of the viscosity and thermal conductivity of argon for a data set provided by Lemmon (47). The value of the viscosity is shown to be within the accuracy of the given data set. The value of the thermal conductivity is within the accuracy of the given data set for data points 1-5 and show a deviation for data points 6. The source of this discrepancy is most likely similar compared to the oxygen verification case.

Table 6-8: Verification of the calculated viscosity (η) and the calculated thermal conductivity (λ) of argon. Both variables are compared to reference data published by Lemmon et al. (47).

#	T	ρ	η_{ref}	η_{er}	λ_{ref}	λ_{er}
	(K)	(mol·dm^{-3})	(Pa·s)	(%)	(W·m^{-1}·K^{-1})	(%)
1	100.0	0.0	$8.18940\cdot 10^{-6}$	$9.8\cdot 10^{-4}$	$6.36587\cdot 10^{-3}$	$2.5\cdot 10^{-3}$
2	300.0	0.0	$2.27241\cdot 10^{-5}$	$4.1\cdot 10^{-4}$	$1.78042\cdot 10^{-2}$	$1.3\cdot 10^{-3}$
3	100.0	33.0	$1.84232\cdot 10^{-4}$	$6.9\cdot 10^{-5}$	$1.11266\cdot 10^{-1}$	$-2.6\cdot 10^{-4}$
4	200.0	10.0	$2.55662\cdot 10^{-5}$	$-9.3\cdot 10^{-5}$	$2.61377\cdot 10^{-2}$	$-1.1\cdot 10^{-4}$
5	300.0	5.0	$2.63706\cdot 10^{-5}$	$-9.8\cdot 10^{-5}$	$2.32302\cdot 10^{-2}$	$1.1\cdot 10^{-4}$
6	154.6	13.4	$2.76101\cdot 10^{-5}$	$1.9\cdot 10^{-5}$	$8.56793\cdot 10^{-1}$	$6.6\cdot 10^{-1}$

Methane

Table 6-9 shows the equation of state verification results for methane. The results of all properties over all the data points are within the accuracy of provided by the reference paper.

Table 6-9: Percentage difference between the calculated data and reference data published by Setzmann et al. (59) for methane at each data point specified by table 6-1.

#	ρ_{er} (%)	u_{er} (%)	h_{er} (%)	s_{er} (%)	$c_{\text{v,er}}$ (%)	$c_{\text{p,er}}$ (%)	w_{er} (%)
1	$-1.4{\cdot}10^{-3}$	$6.1{\cdot}10^{-5}$	$-1.2{\cdot}10^{-3}$	$-2.5{\cdot}10^{-4}$	$-2.3{\cdot}10^{-3}$	$-1.1{\cdot}10^{-3}$	$-7.5{\cdot}10^{-4}$
2	$-4.7{\cdot}10^{-4}$	$-4.6{\cdot}10^{-4}$	$4.8{\cdot}10^{-5}$	$-9.6{\cdot}10^{-4}$	$1.2{\cdot}10^{-4}$	$-2.5{\cdot}10^{-4}$	$-6.3{\cdot}10^{-4}$
3	$1.5{\cdot}10^{-3}$	$-6.0{\cdot}10^{-4}$	$-2.0{\cdot}10^{-4}$	$-1.5{\cdot}10^{-3}$	$-2.0{\cdot}10^{-3}$	$5.5{\cdot}10^{-4}$	$9.1{\cdot}10^{-4}$
4	$6.7{\cdot}10^{-4}$	$4.8{\cdot}10^{-4}$	$-1.5{\cdot}10^{-3}$	$7.9{\cdot}10^{-4}$	$-2.3{\cdot}10^{-5}$	$1.3{\cdot}10^{-3}$	$5.5{\cdot}10^{-4}$
5	$9.9{\cdot}10^{-5}$	$-1.4{\cdot}10^{-3}$	$5.4{\cdot}10^{-4}$	$2.6{\cdot}10^{-3}$	$1.9{\cdot}10^{-3}$	$1.6{\cdot}10^{-3}$	$1.5{\cdot}10^{-3}$
6	$9.9{\cdot}10^{-5}$	$-1.4{\cdot}10^{-3}$	$5.4{\cdot}10^{-4}$	$2.6{\cdot}10^{-3}$	$1.9{\cdot}10^{-3}$	$1.6{\cdot}10^{-3}$	$1.5{\cdot}10^{-3}$
7	$-7.8{\cdot}10^{-4}$	$-2.6{\cdot}10^{-4}$	$-1.1{\cdot}10^{-3}$	$-9.4{\cdot}10^{-4}$	$-2.0{\cdot}10^{-3}$	$-7.5{\cdot}10^{-4}$	$2.9{\cdot}10^{-3}$
8	$8.8{\cdot}10^{-4}$	$-6.4{\cdot}10^{-4}$	$-8.2{\cdot}10^{-4}$	$-7.9{\cdot}10^{-4}$	$-2.2{\cdot}10^{-3}$	$-8.4{\cdot}10^{-4}$	$-4.1{\cdot}10^{-3}$
9	$-2.8{\cdot}10^{-5}$	$4.6{\cdot}10^{-4}$	$-4.8{\cdot}10^{-4}$	$8.8{\cdot}10^{-4}$	$-2.7{\cdot}10^{-3}$	$1.3{\cdot}10^{-3}$	$-2.4{\cdot}10^{-4}$
10	$-1.9{\cdot}10^{-4}$	$1.2{\cdot}10^{-3}$	$5.6{\cdot}10^{-5}$	$-2.4{\cdot}10^{-3}$	$-2.2{\cdot}10^{-4}$	$-1.5{\cdot}10^{-3}$	$-3.6{\cdot}10^{-4}$
11	$-9.9{\cdot}10^{-5}$	$1.3{\cdot}10^{-3}$	$-2.2{\cdot}10^{-3}$	$5.3{\cdot}10^{-4}$	$1.1{\cdot}10^{-3}$	$-1.8{\cdot}10^{-3}$	$5.8{\cdot}10^{-4}$
12	$2.5{\cdot}10^{-3}$	$7.2{\cdot}10^{-4}$	$-1.5{\cdot}10^{-3}$	$5.4{\cdot}10^{-4}$	$-1.6{\cdot}10^{-3}$	$-1.5{\cdot}10^{-3}$	$-8.0{\cdot}10^{-4}$

Ethane

Table 6-10 shows the equation of state verification results for ethane. The results of all properties over all the data points are within the accuracy of provided by the reference paper.

Table 6-10: Percentage difference between the calculated data and reference data published by Bücker et al. (59) for ethane at each data point specified by table 6-1.

#	ρ_{er} (%)	u_{er} (%)	h_{er} (%)	s_{er} (%)	$c_{\text{v,er}}$ (%)	$c_{\text{p,er}}$ (%)	w_{er} (%)
1	$-1.1{\cdot}10^{-3}$	$3.4{\cdot}10^{-5}$	$-4.7{\cdot}10^{-4}$	$1.2{\cdot}10^{-4}$	$-2.8{\cdot}10^{-3}$	$-1.6{\cdot}10^{-3}$	$5.4{\cdot}10^{-4}$
2	$2.8{\cdot}10^{-4}$	$2.2{\cdot}10^{-4}$	$-9.7{\cdot}10^{-4}$	$-1.1{\cdot}10^{-3}$	$1.0{\cdot}10^{-3}$	$-1.2{\cdot}10^{-3}$	$-2.9{\cdot}10^{-4}$
3	$7.0{\cdot}10^{-5}$	$-3.6{\cdot}10^{-4}$	$7.5{\cdot}10^{-4}$	$4.5{\cdot}10^{-3}$	$-2.2{\cdot}10^{-3}$	$6.1{\cdot}10^{-4}$	$1.8{\cdot}10^{-3}$
4	$4.7{\cdot}10^{-4}$	$-5.2{\cdot}10^{-4}$	$-2.1{\cdot}10^{-3}$	$7.4{\cdot}10^{-5}$	$7.5{\cdot}10^{-4}$	$-1.1{\cdot}10^{-3}$	$-1.7{\cdot}10^{-3}$
5	$-3.2{\cdot}10^{-4}$	$-5.7{\cdot}10^{-4}$	$-1.1{\cdot}10^{-4}$	$-1.0{\cdot}10^{-3}$	$1.4{\cdot}10^{-3}$	$6.6{\cdot}10^{-4}$	$-1.7{\cdot}10^{-5}$
6	$-3.2{\cdot}10^{-4}$	$-5.7{\cdot}10^{-4}$	$-1.1{\cdot}10^{-4}$	$-1.0{\cdot}10^{-3}$	$1.4{\cdot}10^{-3}$	$6.6{\cdot}10^{-4}$	$-1.7{\cdot}10^{-5}$
7	$-9.2{\cdot}10^{-4}$	$-7.4{\cdot}10^{-4}$	$6.0{\cdot}10^{-4}$	$-1.0{\cdot}10^{-3}$	$-7.3{\cdot}10^{-4}$	$1.4{\cdot}10^{-3}$	$-2.8{\cdot}10^{-4}$
8	$-7.2{\cdot}10^{-4}$	$6.0{\cdot}10^{-4}$	$-3.0{\cdot}10^{-4}$	$1.1{\cdot}10^{-4}$	$-2.1{\cdot}10^{-3}$	$-1.9{\cdot}10^{-3}$	$6.5{\cdot}10^{-5}$
9	$5.0{\cdot}10^{-4}$	$-2.3{\cdot}10^{-3}$	$-9.9{\cdot}10^{-4}$	$1.1{\cdot}10^{-3}$	$-1.5{\cdot}10^{-3}$	$-6.4{\cdot}10^{-4}$	$1.1{\cdot}10^{-3}$
10	$-3.2{\cdot}10^{-4}$	$-1.3{\cdot}10^{-3}$	$4.8{\cdot}10^{-4}$	$2.9{\cdot}10^{-2}$	$-2.1{\cdot}10^{-3}$	$-9.1{\cdot}10^{-5}$	$5.4{\cdot}10^{-4}$
11	$6.0{\cdot}10^{-4}$	$-4.5{\cdot}10^{-4}$	$-5.0{\cdot}10^{-4}$	$1.1{\cdot}10^{-3}$	$3.7{\cdot}10^{-4}$	$1.1{\cdot}10^{-3}$	$2.8{\cdot}10^{-4}$
12	$2.0{\cdot}10^{-3}$	$1.7{\cdot}10^{-5}$	$-2.3{\cdot}10^{-4}$	$5.1{\cdot}10^{-5}$	$-1.6{\cdot}10^{-3}$	$1.3{\cdot}10^{-3}$	$-1.1{\cdot}10^{-3}$

Nitrous Oxide

Table 6-11 shows the equation of state verification results for nitrous oxide. The results of IPSAT were compared with that of Coolprop for all data points. Coolprop uses the same model and source to compute the thermodynamic state properties of nitrous oxide. The results of all properties over all the data points are all within 10^{-3} % which is comparable to other species that are evaluated during this verification procedure.

Table 6-11: Percentage difference between the calculated data and reference data calculated by coolprop (3) for nitrous oxide at each data point specified by table 6-1.

#	ρ_{er} (%)	u_{er} (%)	h_{er} (%)	s_{er} (%)	$c_{\text{v,er}}$ (%)	$c_{\text{p,er}}$ (%)	w_{er} (%)
1	$4.6\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.9\cdot10^{-4}$	$-2.7\cdot10^{-4}$
2	$4.6\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.9\cdot10^{-4}$	$-2.6\cdot10^{-4}$
3	$6.5\cdot10^{-4}$	$-6.0\cdot10^{-4}$	$-6.0\cdot10^{-4}$	$-6.1\cdot10^{-4}$	$-5.4\cdot10^{-4}$	$-4.0\cdot10^{-4}$	$-3.1\cdot10^{-4}$
4	$6.3\cdot10^{-4}$	$-5.9\cdot10^{-4}$	$-5.9\cdot10^{-4}$	$-6.0\cdot10^{-4}$	$-5.5\cdot10^{-4}$	$-4.7\cdot10^{-4}$	$-3.1\cdot10^{-4}$
5	$5{,}8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-2.9\cdot10^{-4}$
6	$5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-2.9\cdot10^{-4}$
7	$4.5\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-2.8\cdot10^{-4}$
8	$4.6\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-2.7\cdot10^{-4}$
9	$5.9\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.9\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.5\cdot10^{-4}$	$-2.9\cdot10^{-4}$
10	$5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.6\cdot10^{-4}$	$-2.9\cdot10^{-4}$
11	$5.3\cdot10^{-4}$	$-5.9\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.9\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-2.3\cdot10^{-4}$
12	$5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.8\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-5.7\cdot10^{-4}$	$-2.9\cdot10^{-4}$

Conclusion

It is demonstrated that IPSAT is able to calculate the state variables for all implemented fluids to within the significant digits of the source papers. Except for oxygen. However, the difference for all the thermophysical state properties of oxygen is still well within 1% of the reported value. It can be concluded that the evaluation of the Helmholtz multiparameter equation of state is correctly implemented. However, when selecting a fluid the designer needs to be aware of the limitations that are reported in this section for certain models.

The verification of the transport properties of nitrogen oxygen and argon have demonstrated that the formulations that are presented in sections 5-1-8 and 5-1-7 have been correctly implemented for those fluids. However, the thermal conductivity at the critical point for both oxygen and argon show deviations which are within 1% of the reference value. The exact source of this deviation is not yet known.

It is recommended that the process as described in this section is repeated for each implemented fluid as it is imperative that all the constants are registered correctly. Experience has shown that a small mistake in interpretation and/or implementation of the constants can have large effects on the outcome of the models.

6-1-2 Fluid Friction Module Verification

This section describes the verification fluid friction module as presented in section 5-2. The verification procedure is performed on the pipe flow friction model, specifically on the implementation of the Colebrook relation. The goal is to verify that the fluid friction module is able to correctly retrieve the Darcy friction factor for the given geometrical inputs.

Colebrook Relation

The calculation of the dimensionless friction factor f, is the most important subroutine in the determination of the fluid friction factor K_f in a pipe. The value of the dimensionless Darcy friction factor f is obtained through the Colebrook relation, see equation 5-42 in section 5-2. The Colebrook relation does not have a closed form, therefore an iterative solver method was implemented to obtain the value of f.

In order to verify this method along with the values of f across a wide range of values of Re and ϵ/D, the Moody diagram is constructed using the implemented solver subroutine. The moody diagram gives the values of f as a function of the Reynolds number Re for a range of values of the relative roughness, ϵ/D. This diagram was used in the past to determine the value of f without having to solve the implicit Colebrook relation. The Moody diagram as constructed from IPSAT is shown in figure 6-1.

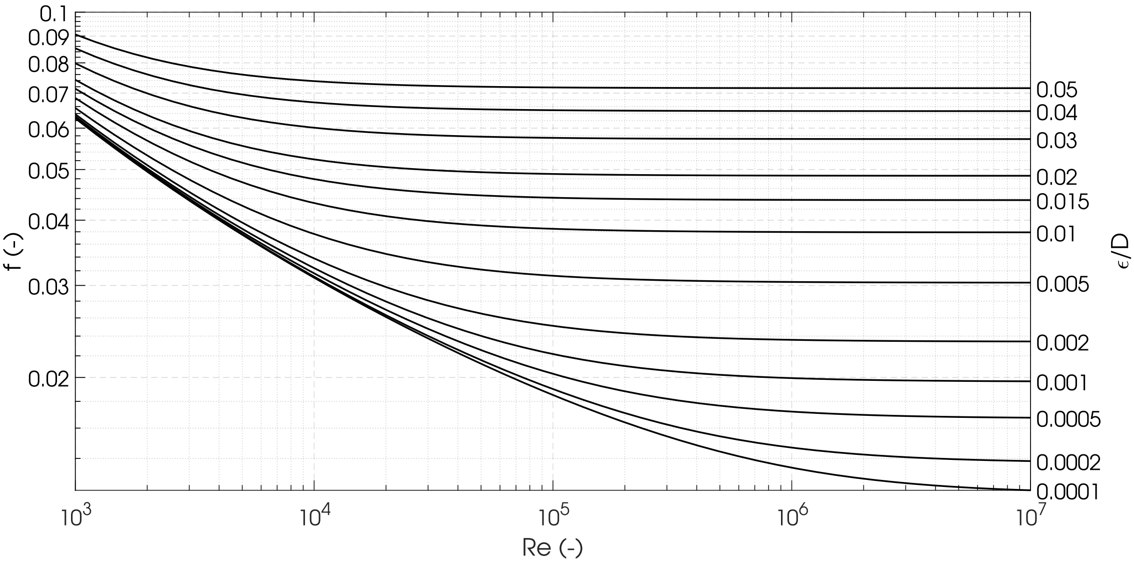


Figure 6-1: Moody diagram generated from the implemented Colebrook relation in IPSAT.

The Moody diagram as computed by IPSAT is identical in shape to the diagram originally published by Moody (77). Moreover, the diagram shows that the value of f varies smoothly even though each point is calculated independently and iteratively. This indicates that the iterative function calculating the Colebrook friction factor is correctly implemented and the behavior of turbulent flow through rough pipes are modeled according to the observations of Moody.

In order to verify the obtained value of the Darcy friction factor, the results of IPSAT are compared to published reference data by Brkic (78). Table 6-12 shows the results of this verification process.

Table 6-12: Verification of the calculated value of the Darcy friction factor f using the Colebrook relation. The reference data is published by Brkic (78).

#	ϵ/D	Re	f_{ref}	f_{calc}	f_{er}
	(-)	(-)	(-)	(-)	(%)
1	$1.0{\cdot}10^{-6}$	$1.0 \cdot 10^{4}$	$3.088{\cdot}10^{-2}$	$3.088{\cdot}10^{-2}$	$1.4{\cdot}10^{-5}$
2	$3.0{\cdot}10^{-3}$	$5.8{\cdot}10^{6}$	$2.617{\cdot}10^{-2}$	$2.619{\cdot}10^{-2}$	$7.6{\cdot}10^{-2}$
3	$4.3{\cdot}10^{-4}$	$3.0{\cdot}10^{7}$	$1.616{\cdot}10^{-2}$	$1.617{\cdot}10^{-2}$	$5.9{\cdot}10^{-2}$
4	$2.0{\cdot}10^{-4}$	$6.0{\cdot}10^{4}$	$2.084{\cdot}10^{-2}$	$2.084{\cdot}10^{-2}$	$9.6{\cdot}10^{-3}$
5	$3.0{\cdot}10^{-2}$	$4.0{\cdot}10^{5}$	$5.719{\cdot}10^{-2}$	$5.725{\cdot}10^{-2}$	$1.1{\cdot}10^{-1}$

It can be seen that for points 1-4 the data can be reproduced to within the accuracy of the reference data. The deviation which can be seen in data point 5 is attributed to the unknown accuracy limit which is set for the iterative procedure used by Brkic.

6-1-3 Solver Module Verification

This section describes the verification of the solver module as presented in section 5-3. The verification procedure is performed on the Newton-Raphson method, Broyden's method, the modified Broyden's method and Brent's method. The goal is to verify that the solver module is able to successfully reduce the residuals of a fluid system with a large number of conservation equations. Next to that, the solver module should be able to correctly apply the designer's choice for the selection of the various solver module parameters.

Verification Setup

The solver module is in charge of minimizing the residuals of a given set of conservation equations. The verification procedure will run through the different solving methods and check the convergence rate and the final converged solution. In order to check the convergence rate, the residuals of each conservation equation are logged over the solving process. The verification procedure is standardized by taking the mean value of the residuals for each equation and by making the residuals non-dimensional.

The normalized mean residuals are defined as the mean absolute residuals divided by the original value of the mean absolute residuals:

$$r_{i,norm} = \frac{\overline{|r_i|}}{\overline{|r_0|}} \tag{6-1}$$

Where the mean absolute residual $\overline{|r_i|}$ is defined as:

$$\overline{|r_i|} = \frac{\sum_{j=1}^{n} |r_{i,j}|}{n} \tag{6-2}$$

Where n is the total number of node or branches where the specific conservation equation is solved.

The solver verification procedure is applied on a predefined simple problem. This is done to focus solely on the behavior of the solver. The problem that is proposed is a 10 nodes and 9 branches model aligned in a single direction, see figure 6-2. The fluid used is nitrogen which is a simple, well understood fluid and is therefore often used in model analysis. The problem is setup in such a way that the fluid is always in gaseous phase in order to eliminate any peculiarities in the solver due to phase changes.

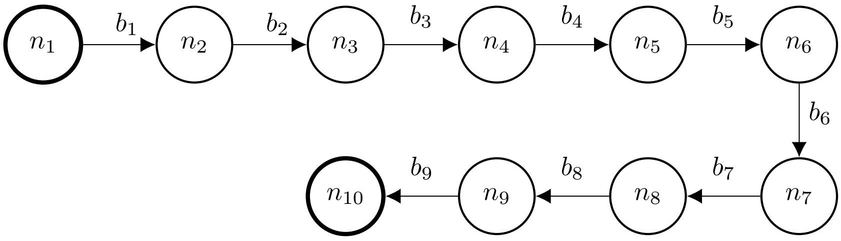


Figure 6-2: Nodes and branches architecture of the solver verification problem.

Each branch has the same geometric parameters, this is done to simplify the problem. Table 6-13 shows the geometric properties of each branch.

Table 6-13: Properties of each Branch in the solver verification problem.

Property	**Value**	**Unit**
Length	$1 \cdot 10^{-1}$	m
Diameter	$1 \cdot 10^{-2}$	m
Roughness	$4.5 \cdot 10^{-5}$	m
Branch type	pipe	-

The properties of the boundary nodes are listed in table 6-14.

Table 6-14: Boundary node properties of the solver verification problem.

Property	**Value**	**Unit**
	Node 1	
Species	nitrogen	-
Pressure	$1 \cdot 10^{6}$	Pa
Temperature	273.15	K
	Node 10	
Species	nitrogen	-
Pressure	101325	Pa
Temperature	273.15	K

The program settings that were used for the solver verification problem are listed in table 6-15.

Table 6-15: Program settings of the solver verification problem.

Option	**Setting**
Fluid system initialization module settings	
Nodes initialization method	linear interpolation
Branches initialization method	Bernoulli
Solver module settings	
Solver mode	steady state
Iteration min	0
Iteration max	(variable)
Equations	fluid mass, fluid momentum, fluid energy
Variables	pressure, massflow, enthalpy
Conservation of fluid energy	
Solver method	(Variable)
Relaxation factor	(Variable)
Convergence limit	N/A (no limit set)
Divergence limit	N/A (no limit set)
Equation terms	massflow
Conservation of fluid momentum	
Solver method	(Variable)
Relaxation factor	(Variable)
Convergence limit	N/A (no limit set)
Divergence limit	N/A (no limit set)
Equation terms	pressure, friction
Conservation of fluid mass	
Solver method	(Variable)
Relaxation factor	(Variable)
Convergence limit	N/A (no limit set)
Divergence limit	N/A (no limit set)
Equation terms	massflow
Thermophysics module settings	
Variable transform model	Helmholtz MEOS
Equation of state model	Helmholts MEOS
Melting pressure model	modified Simon formulation
Vapour pressure model	polylogarithmic formulation
Saturated liquid density model	polylogarithmic formulation
Saturated vapour density model	polylogarithmic formulation
Surface tension model	REFPROP formulation
Viscosity model	Lemmon formulation
Thermal conductivity model	Lemmon formulation

Newton-Raphson Method

Figure 6-3 and 6-4 show the mean normalized residuals of the three conservation equations over the solving process for a relaxation factor of 1 and 0.1 respectively.

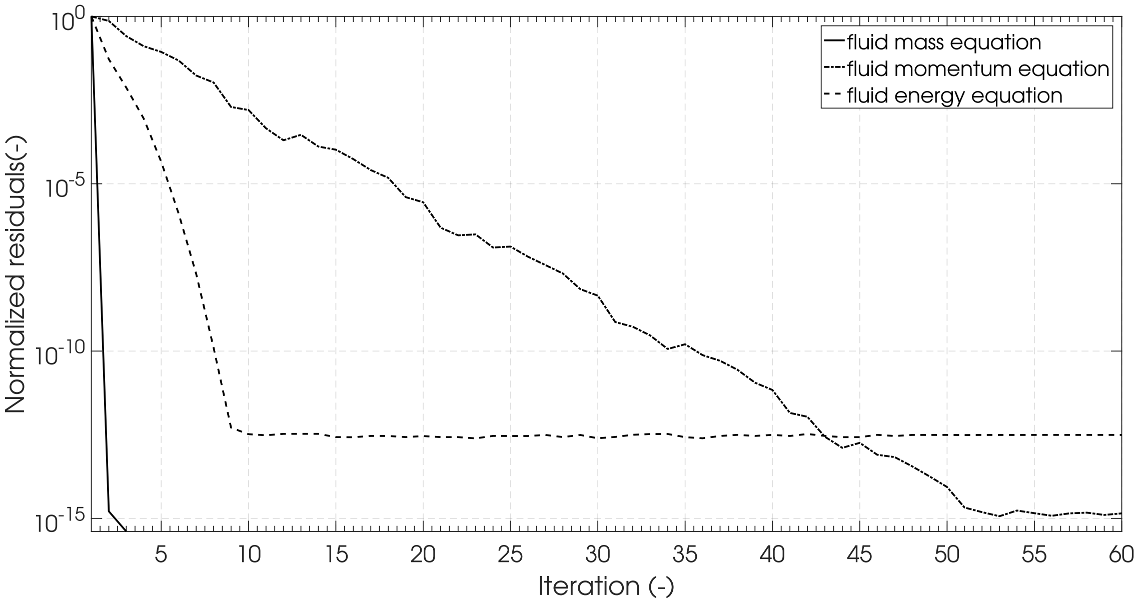


Figure 6-3: The change of the normalized residuals over the iterations using the Newton-Raphson method with a relaxation factor of 1.

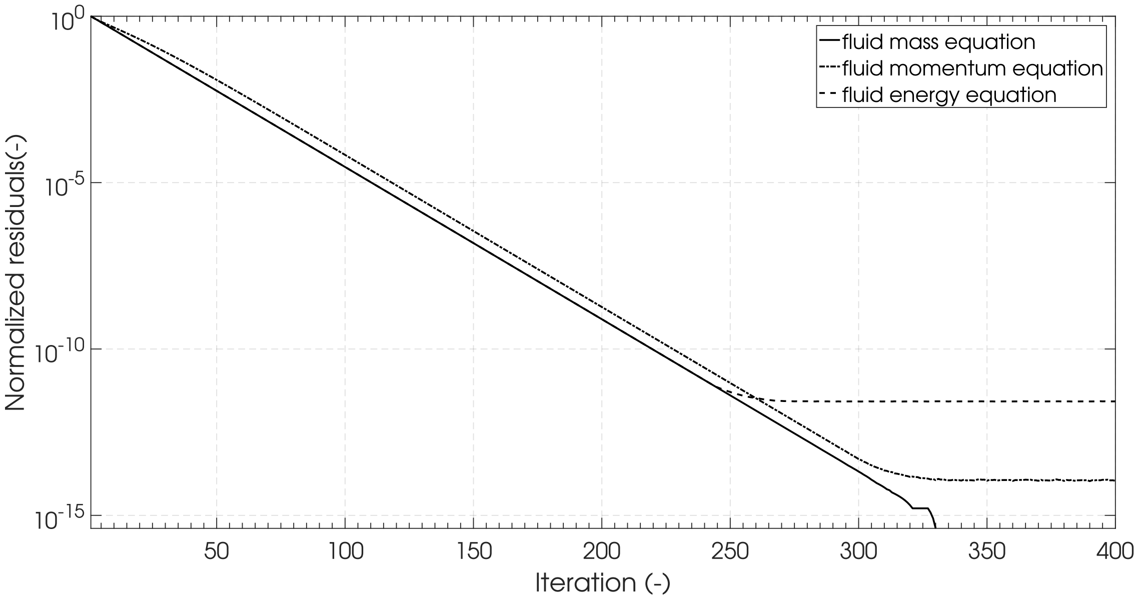


Figure 6-4: The change of the normalized residuals over the iterations using the Newton-Raphson method with a relaxation factor of 0.1.

It can be seen that the conservation of fluid energy equation seems to be converging linearly and the slowest when the relaxation factor is set to 1. The conservation of fluid mass equation and the conservation of fluid momentum equation are both converging quadratically, which is to be expected from the Newton-Raphson method. When the relaxation factor is decreased to 0.1, all the conservation equations converge at an almost equal, linear, rate. This is to be expected, because the Newton-Raphson method solves the system of equations simultaneously. Figure 6-4 also shows that reducing the relaxation factor has a stabilizing effect on convergence. All conservation equations converge much smoother compared to when the relaxation factor is set to 1.

In both cases the convergence stops when the residuals have reached a value smaller than 10^{-13}. This is expected, because the precision limit of 64-bit numbers lies in the order of 10^{-15}. For most problems a precision below 10^{-5} is more than sufficient. The residuals of the conservation of fluid mass actually drops to 0. This is attributed to the simple form of this equation and the lack of dependency with respect to other precision dependent calculations, for example the calculation of the thermophysical properties.

Broyden's Method

Figure 6-5 and 6-6 show the mean normalized residuals of the three conservation equations over the solving process using Broyden's method with a relaxation factor of 0.5 and 0.1 respectively.

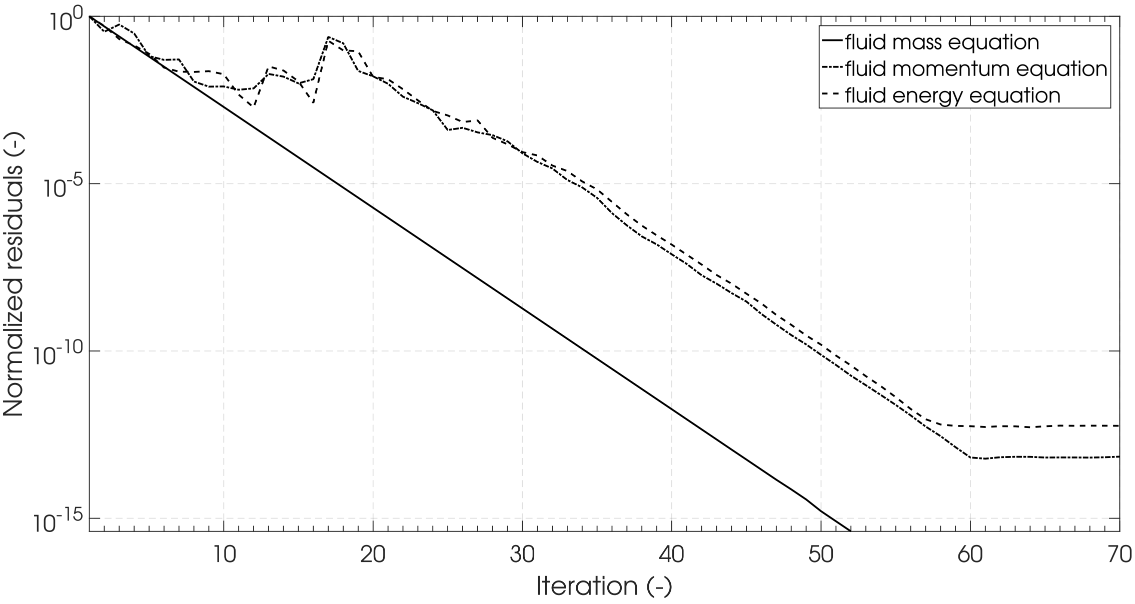


Figure 6-5: The change of the normalized residuals over the iterations using Broyden's method with a relaxation factor of 0.5.

Figure 6-5 shows that Broyden's method has some difficulties converging at the start for a high relaxation factor. Solving the problem using Broyden's method with a relaxation factor of 1 resulted in a diverged solution. Therefore, the highest relaxation factor was taken to be 0.5. However, after a certain number of iterations, the convergence rate seem to stabilize. It can be seen that Broyden's method converges linearly, which is expected.

Figure 6-6 shows that the convergence rate of Broyden's method is similar to that of the Newton-Raphson method for a low relaxation factor. For this problem Broyden's method reduced the time per iteration with a factor of 4.3 compared to the Newton-Raphson method (with a relaxation factor of 0.1).

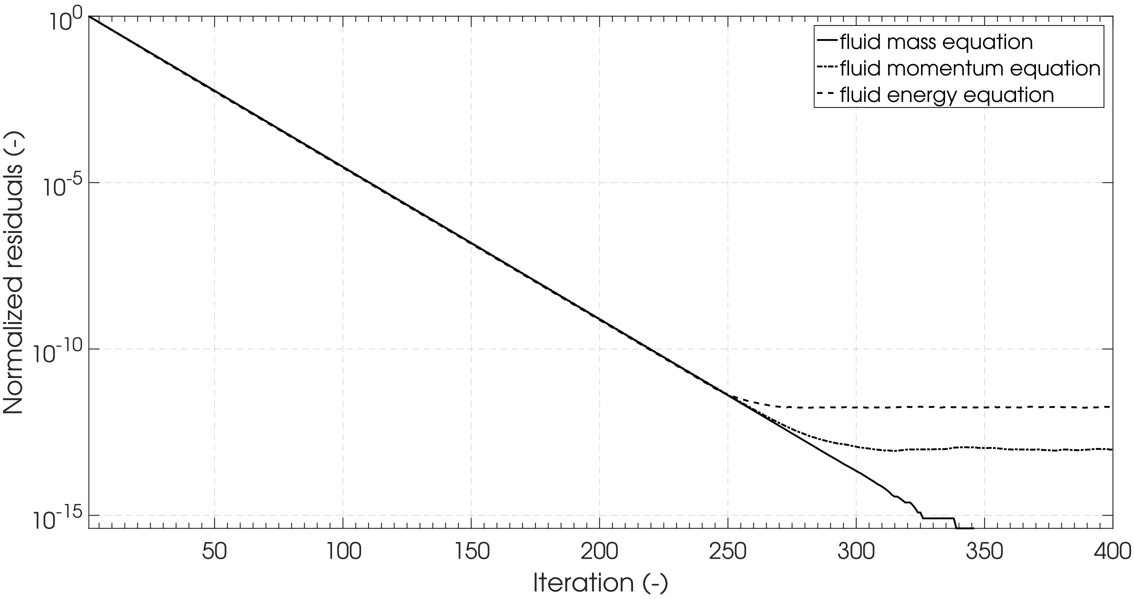


Figure 6-6: The change of the normalized residuals over the iterations using Broyden's method with a relaxation factor of 0.1.

Modified Broyden's Method

Figure 6-7 and 6-8 show the mean normalized residuals of the three conservation equations over the solving process using the modified Broyden's method, using a Sherman Morrison formula, with a relaxation factor of 0.5 and 0.1 respectively.

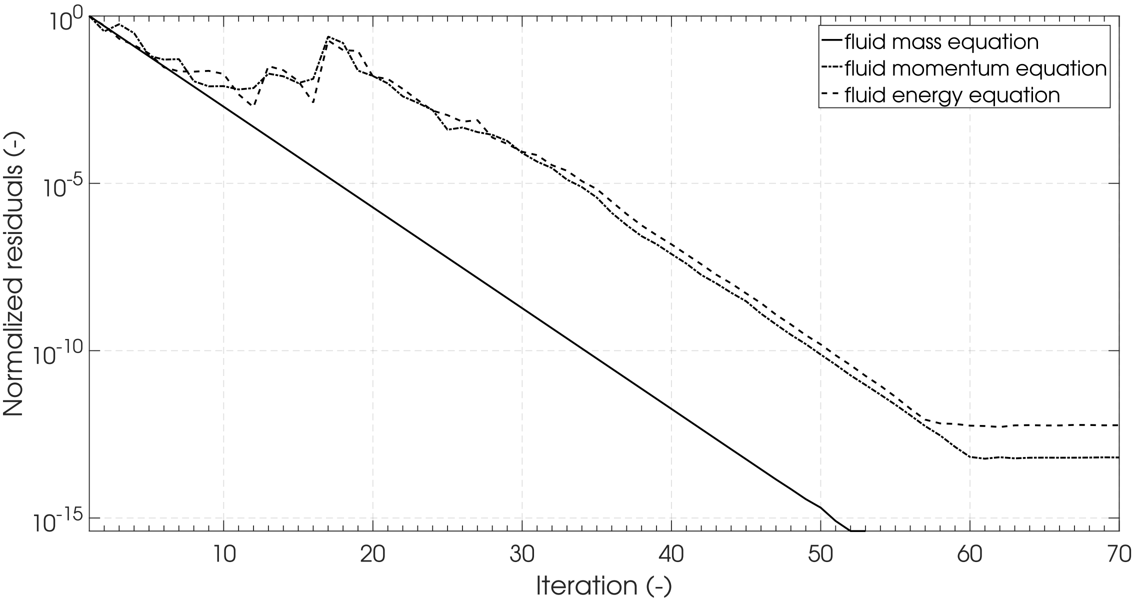


Figure 6-7: The change of the normalized residuals over the iterations using the modified Broyden's method with a relaxation factor of 0.5.

Figure 6-7 and 6-8 show that the modified Broyden's method behaves almost identical, when compared to the regular Broyden's method, for this problem. This was expected because the Jacobian matrix is almost completely symmetrical.

Both Broyden's method and the modified Broyden's method have similar execution times when executing the reference problem with a relaxation factor of 0.1. It was expected that the modified Broyden's method would have a slightly shorter execution time. It is hypothesized that the Jacobian matrix which is constructed in the reference problem is too small to produce a noticeable difference in execution time.

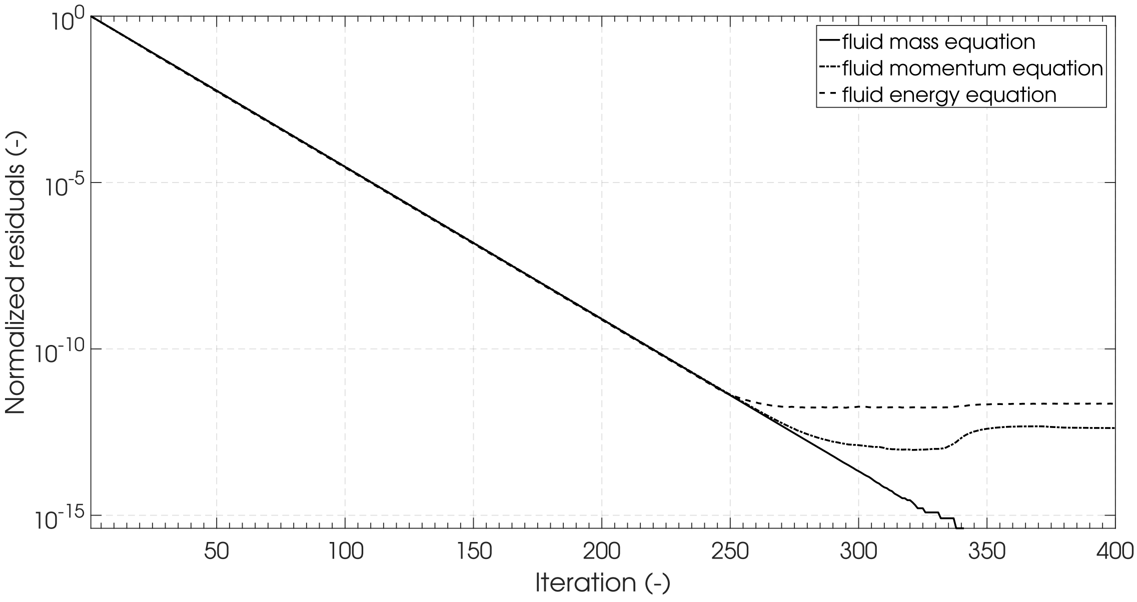


Figure 6-8: The change of the normalized residuals over the iterations using the modified Broyden's method with a relaxation factor of 0.1.

Brent's Method

Figure 6-9 and 6-10 show the mean normalized residuals of the three conservation equations over the solving process using Brent's method with a relaxation factor of 0.1 and 0.01 respectively. Brent's method solves each conservation equation separately during each iteration. It is therefore not expected that it is effective at solving a system of equations.

It can be seen that the convergence of Brent's equation is slow, sub-linear, when using it to solve a system of equations. Similar convergence behavior can be expected from other single equation solving methods (e.g. regula falsi). The convergence behavior also seems very erratic. This is to be expected from a method which solves each equation independent from the other equations.

The final value of the residuals remains relatively high and depends on the relaxation factor. The final value of the residuals scales directly with the value of the relaxation factor, i.e. reducing the relaxation factor with a factor of 10 reduces the final value of the residuals with a factor of 10. It is not recommended to use the single equation solving methods as a main solving method. Brent's method and similar single equation root finding methods can be used to supplement the more effective solving methods.

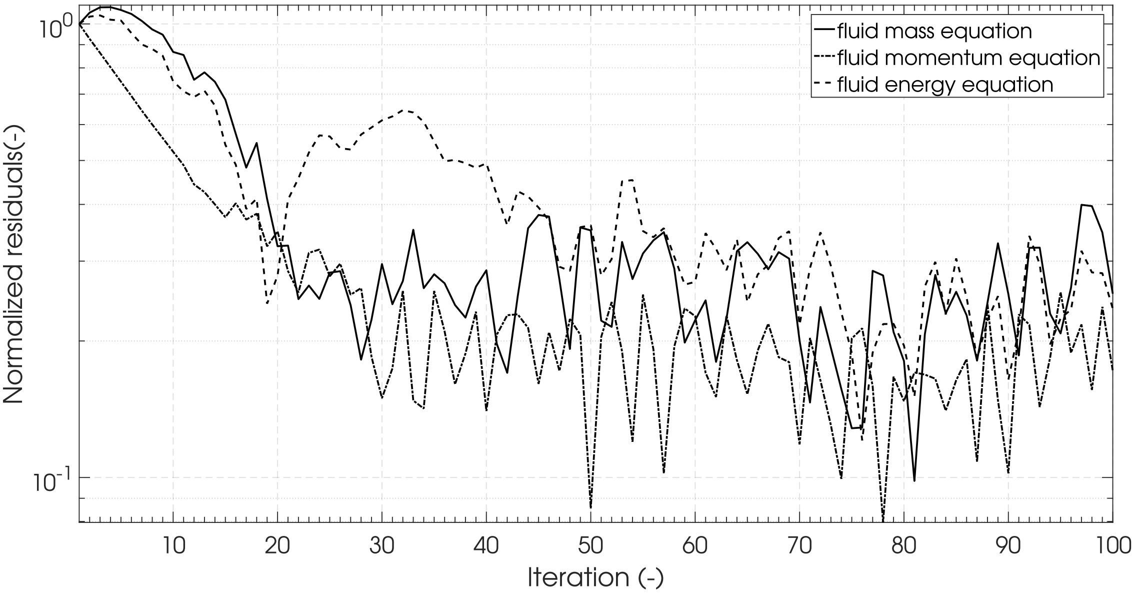


Figure 6-9: The change of the normalized residuals over the iterations using Brent's method with a relaxation factor of 0.1.

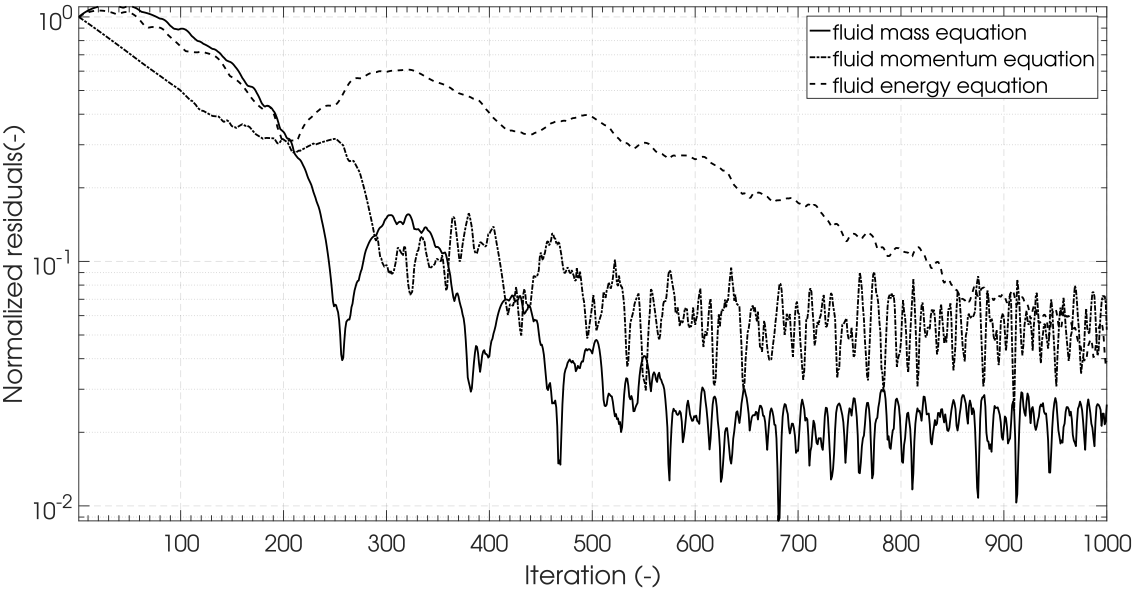


Figure 6-10: The change of the normalized residuals over the iterations using Brent's method with a relaxation factor of 0.01.

Conclusion

The solver module is able to use a number of different methods to minimize the residuals of the specified conservation equations. Each method behaves as is to be expected when applied to the reference problem. The solver module is able to reduce the value of the residuals to within numerical accuracy when using methods which solve equations simultaneously.

The solver module is also able to successfully execute and interact with external modules as part of its routine. These external modules can be successfully selected in the settings file defined by the designer.

6-1-4 Fluid System Initialization Module Verification

This section describes the verification of the fluid system initialization module as presented in section 5-4. The verification procedure is only performed on the linear nodes interpolation method and the Bernoulli branches initialization because those methods require an automated procedure. The goal is to check if the fluid system initialization module provides the correct initial guess according to the methods described in section 5-4.

Verification Setup

This module is applied to two different fluid systems, the results of which are presented in this section. This is done so that the flexibility of the fluid initialization module can be demonstrated. The first fluid system is a simple linear fluid system which can be seen in figure 6-11. The second fluid system in a more complex fluid system with multiple boundary nodes and internal loops. This fluid system can be seen in figure 6-12. For both cases, nitrogen is used as a fluid.

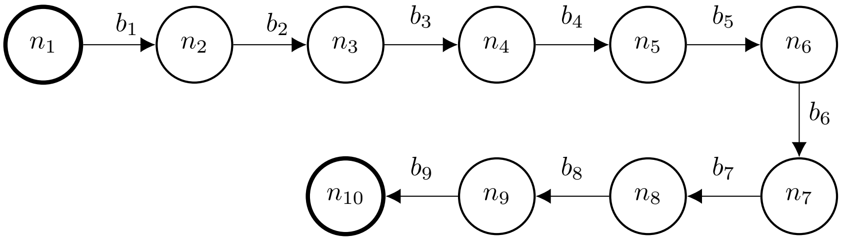


Figure 6-11: Nodes and branches architecture of the fluid system initialization module case 1.

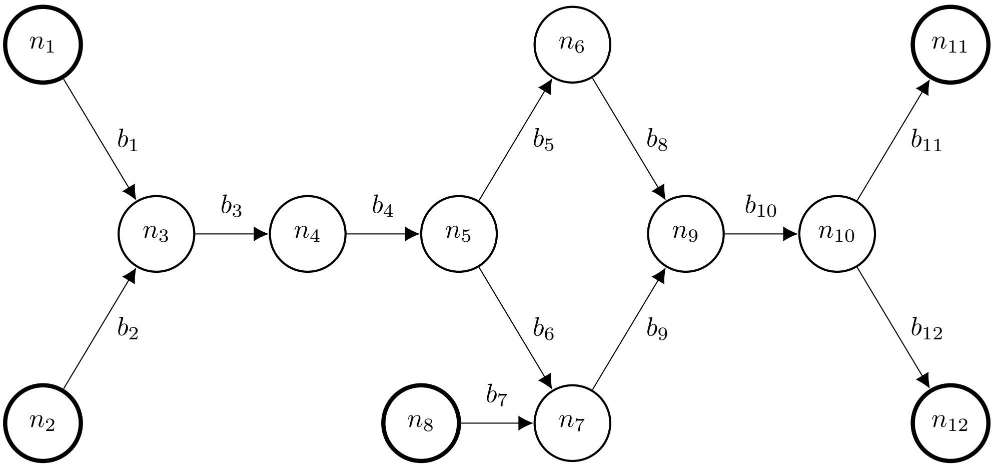


Figure 6-12: Nodes and branches architecture of the fluid system initialization module case 2.

Linear Interpolation Nodes Initialization Method

The results of the linear interpolation nodes initialization method is shown in tables 6-16 and 6-17, where table 6-16 shows the results for case 1 and table 6-17 shows the results for case 2.

For case 1, the linear interpolation method identifies 8 distinct node layers and 9 pressure steps. Node, n_1, is identified to have the highest pressure of 6.0 MPa and node, n_{10} to have the lowest pressure of 0.1 MPa. The decrement in pressure (from node 1), Δp, for each node using the linear interpolation method is equal to 0.656 MPa. Similarly, the increment in temperature (from node 1), ΔT, for each node using the linear interpolation method is equal to 3.33 K.

Table 6-16: Calculated values for the internal nodes using the linear interpolation nodes initialization method for the fluid system given in figure 6-11.

node	type	p (MPa)	T (K)
n_1	boundary	6.000	270.0
n_2	internal	5.344	273.3
n_3	internal	4.689	276.7
n_4	internal	4.033	280.0
n_5	internal	3.378	283.3
n_6	internal	2.722	286.7
n_7	internal	2.068	290.0
n_8	internal	1.411	293.3
n_9	internal	0.756	296.7
n_{10}	boundary	0.100	300

For case 2, the linear interpolation method identifies 6 distinct node layers and 8 pressure steps. Nodes, n_1 and n_2 are identified to have the highest pressure (6.0 MPa), whilest nodes n_{11} and n_{12} are identified to have the lowest pressure (0.1 MPa). The reverse is true for the temperature. The decrement in pressure using the defined pressure steps is calculated to be 0.7375 MPa. The increment in temperature is equal to 3.75 K.

Using the linear interpolation initialization method on case 2 shows that there exist a jump in the value of the estimated thermophysical property if the amount of node layers is smaller than the number of pressure steps - 1. See the difference in pressure from node n_{10} and n_{11} for the pressure and node n_2 and n_3 for the temperature. In other words, in case the fluid system has loops and/or branches the linear interpolation method might return jumps in the value of the estimated thermophysical property which is larger than expected.

Table 6-17: Calculated values for the internal nodes using the linear interpolation nodes initialization method for the fluid system given in figure 6-12.

node	type	p (MPa)	T (K)
n_1	boundary	6.000	270.0
n_2	boundary	6.000	270.0
n_3	internal	5.263	277.5
n_4	internal	4.525	281.3
n_5	internal	3.788	285.0
n_6	internal	3.050	288.8
n_7	internal	3.050	288.8
n_8	boundary	5.000	270.0
n_9	internal	2.313	292.5
n_{10}	internal	1.606	296.3
n_{11}	boundary	0.100	300
n_{12}	boundary	0.100	300

Bernoulli Branches Initialization Method

The results of the Bernoulli initialization method is shown in tables 6-18 and 6-19, where table 6-18 shows the results for case 1 and table 6-19 shows the results for case 2.

It can be seen that in both cases, the massflow decreases gradually due to a decrease in upstream node density. The massflow in all cases match the values that are calculated using the Bernoulli massflow method, see equation 5-104. Note that the decrease in massflow might make this method unsuitable for some steady state analyses.

Table 6-18: Calculated values for the massflow through the fluid branches using the Bernoulli massflow method for the fluid system given in figure 6-11.

branch	Area (m^2)	ρ_{US} (kg·m^{-3})	$\dot{m}$ (kg·s^{-1})
b_1	$7.854{\cdot}10^{-5}$	$7.631{\cdot}10^1$	0.786
b_2	$7.854{\cdot}10^{-5}$	$6.694{\cdot}10^1$	0.736
b_3	$7.854{\cdot}10^{-5}$	$5.787{\cdot}10^1$	0.684
b_4	$7.854{\cdot}10^{-5}$	$4.906{\cdot}10^1$	0.630
b_5	$7.854{\cdot}10^{-5}$	$4.051{\cdot}10^1$	0.572
b_6	$7.854{\cdot}10^{-5}$	$3.220{\cdot}10^1$	0.510
b_7	$7.854{\cdot}10^{-5}$	$2.412{\cdot}10^1$	0.442
b_8	$7.854{\cdot}10^{-5}$	$1.626{\cdot}10^1$	0.363
b_9	$7.854{\cdot}10^{-5}$	$0.859{\cdot}10^1$	0.264

Table 6-19: Calculated values for the massflow through the fluid branches using the Bernoulli massflow method for the fluid system given in figure 6-12.

branch	Area (m^2)	ρ_{US} (kg·m^{-3})	$\dot{m}$ (kg·s^{-1})
b_1	7.854·10^{-5}	7.631·10^1	0.833
b_2	7.854·10^{-5}	7.631·10^1	0.833
b_3	7.854·10^{-5}	6.477·10^1	0.768
b_4	7.854·10^{-5}	5.480·10^1	0.706
b_5	7.854·10^{-5}	4.516·10^1	0.641
b_6	7.854·10^{-5}	4.516·10^1	0.641
b_7	7.854·10^{-5}	4.680·10^1	0.741
b_8	7.854·10^{-5}	3.582·10^1	0.571
b_9	7.854·10^{-5}	3.582·10^1	0.571
b_{10}	7.854·10^{-5}	2.676·10^1	0.493
b_{11}	7.854·10^{-5}	1.796·10^1	0.572
b_{12}	7.854·10^{-5}	1.796·10^1	0.572

Conclusion

It is demonstrated that the linear interpolation initialization method can correctly identify the extreme values of a complex fluid system and interpolate between them. The Bernoulli initialization method is able to correctly estimate an initial massflow figure for a given density, pressure and cross-section geometry according to the methods described in section 5-4-1. It is still the task of the designer to evaluate if the calculated results are able to be used as the initial value for the given design problem.

Chapter 7

Test Setup and Results

The last part of this thesis project is to validate the program which is described in chapters 4 till 6. According to the computer program requirements listed in section 2-4, the program shall be validated using the DHX-200 Aurora engine data. This chapter will present the DHX-200 Aurora engine and the data that was gathered during the conducted engine test campaigns. This data is used to set up and conduct the validation procedure.

The details of the DHX-200 Aurora engine is presented in section 7-1. Section 7-2 presents the part of the test setup of the DHX-200 Aurora engine which is important for the validation process. Lastly, section 7-3 presents the test results, relevant to the validation process, of the DHX-200 Aurora engine which were gathered during the engine test campaigns.

7-1 DHX-200 Aurora Engine

The DHX-200 Aurora engine is a hybrid rocket engine developed by DARE. Some critical design parameters of this engine can be seen in table 7-1 for reference. The engine was designed to fly the Stratos II rocket to an altitude of 50 km. A failed launch campaign in 2014 meant that the engine did not fly on the original Stratos II mission. The engine eventually flew as part of the Stratos II^{+} mission to an altitude of 21.5 km in October 2015. The reason for this offset in altitude is largely contributed to a misinterpretation of the flight simulations done at the time. According to the data gathered during the flight, the engine performed similarly to what was to be expected from the ground tests.

Table 7-1: Main performance parameters of the DHX-200 Aurora engine (79).

Parameter	Value
Thrust (kN)	11.86
Specific impulse (s)	205
Combustion pressure (MPa)	2.08
Oxidizer mass flow (kg·s^{-1})	2.62
O/F mass ratio (-)	3.05

Between May 2013 and July 2015 a total number of 14 static engine firing tests were performed during the development of the DHX-200 Aurora engine. Many of these tests did not achieve the planned test objectives due to a premature failure of the engine. Even though these tests did not meet their objectives, useful performance data was gathered nonetheless for most of these tests. This data includes detailed feed system pressure data over time and combustion pressure data over time and oxidizer tank mass data over time. The feed system is not much affected by changes in the combustion chamber, due to the flow choking at the injector. Therefore, the data gathered from these static firing tests are useful for the validation of the IPSAT that models the feed system behavior.

A photo of the test setup before a test is shown in figure 7-1. The picture shows the thrustbench with the DHX-200 Aurora hybrid engine mounted onto it, the two run tanks behind the engine and the various feed system components.

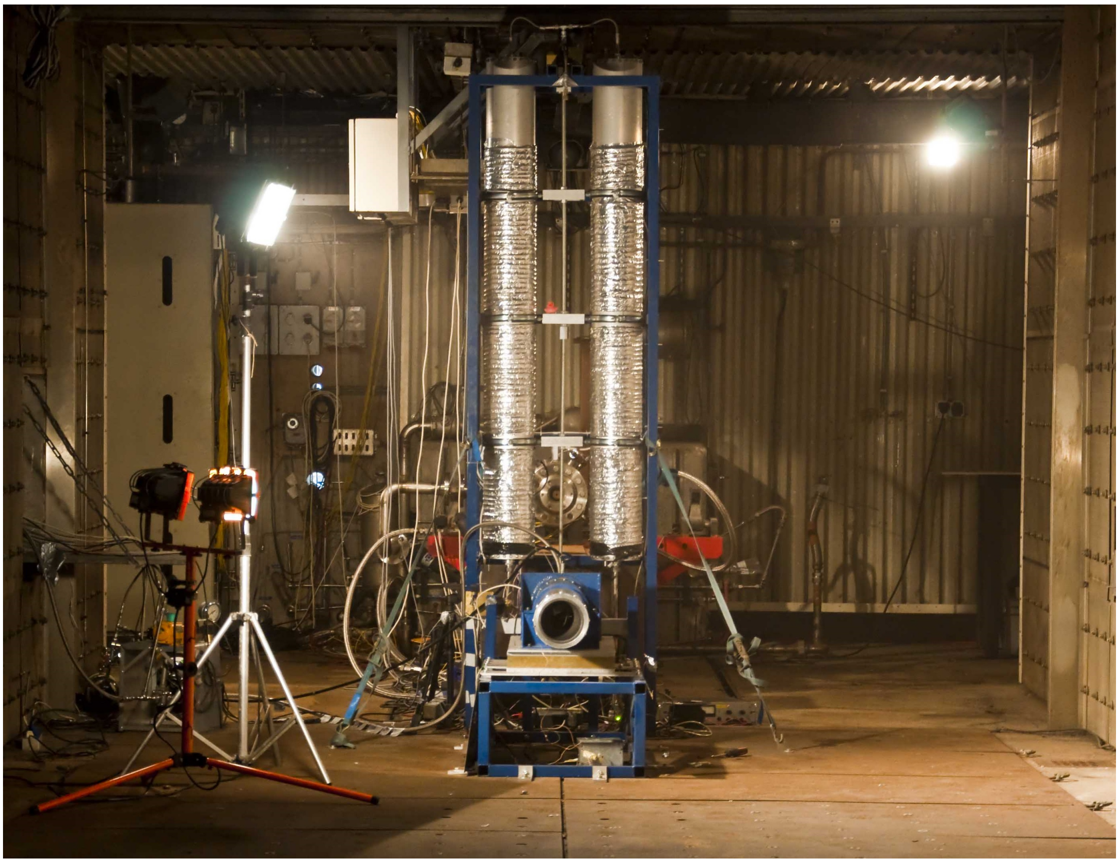


Figure 7-1: DHX-200 Aurora test bench at TNO in Rijswijk. (Image by V.R. Huijsman)

7-2 DHX-200 Aurora Test setup

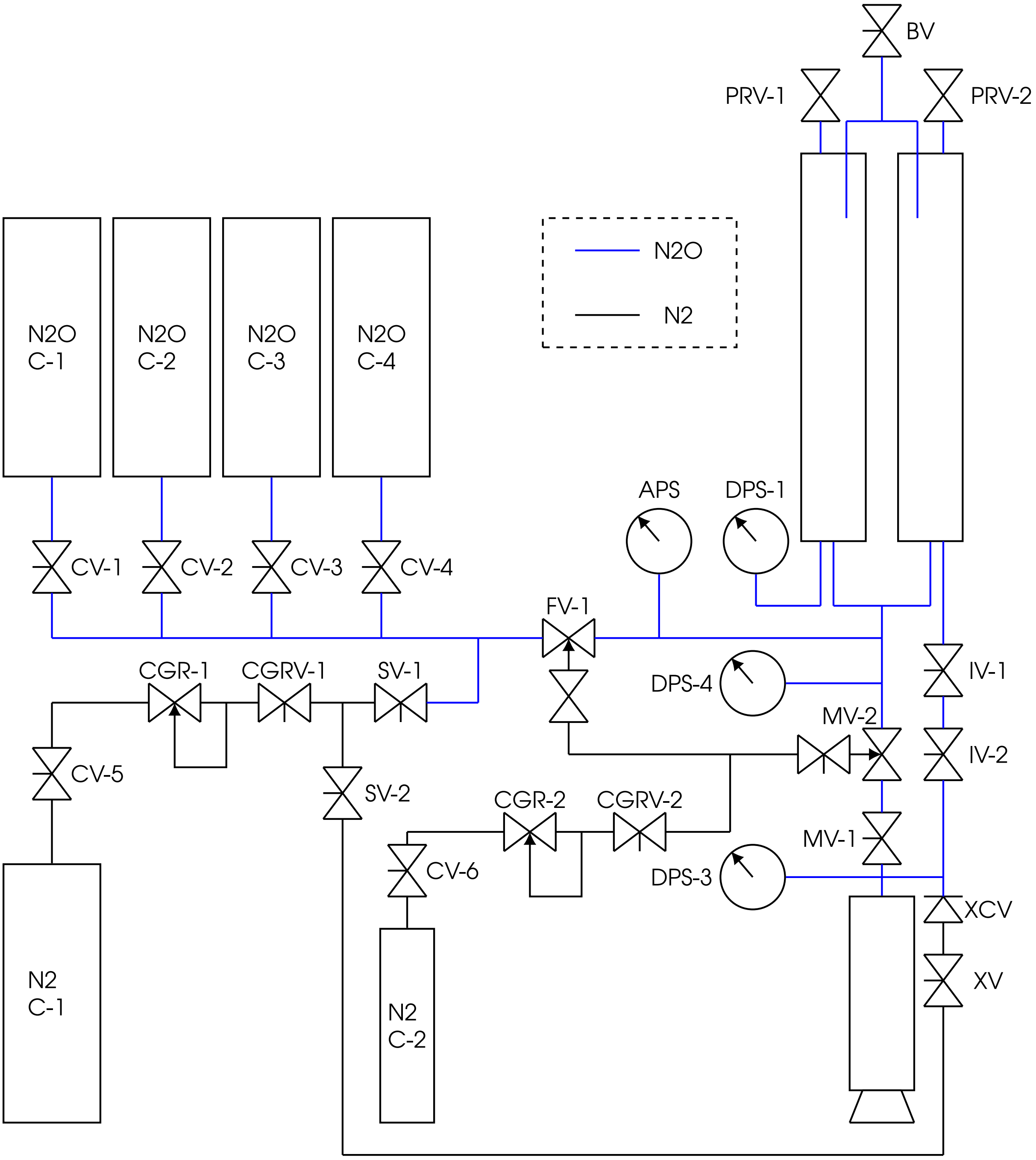
The test setup of the DHX-200 Aurora engine varied slightly over the 14 tests that were conducted. Especially the internal engine configuration changed for almost every test (80, 81). The feed system design and layout remained relatively constant apart from changes made in the tank feed line applied in test 5 onwards.

The test configuration of the DHX-200 Aurora engine consists of: a filling system containing the nitrous oxide gas cyllinders, 2 oxidizer run tanks which store the pressurized liquid oxidizer shortly before a test and the hybrid rocket combustor. A schematic of the DHX-200 test engine feed system layout can be seen in figure 7-2.

The feed system, connecting the three main components, contains several sensors and actuators. The sensors and actuators are logged and actuated by a CompactRIO controller from National Instruments. An overview of all the sensors that are used to log data during each test is shown in table 7-2.

Table 7-2: An overview of the sensors allocated for every test of the DHX-200 Aurora engine (between test 6 and test 14).

Parameter	**Sensor**	**Accuracy**	**Sampling rate**
Thrust force	AST Force Transducer KAS 50 kN	50 N (82)	1 kHz (test 1-10) 2 kHz (test 11-14)
Tank mass	Tedea-Huntleigh 1240 2 kN	0.4 N (83)	1 kHz (test 1-10) 2 kHz (test 11-14)
Combustion pressure	Parker ASIC 0 - 100 bar	0.2 bar (84)	1 kHz (test 1-10) 2 kHz (test 11-14)
Tank pressure	Parker ASIC 0 - 100 bar	0.2 bar (84)	1 kHz (test 1-10) 2 kHz (test 11-14)
Feed system pressure	Parker ASIC 0 - 100 bar	0.2 bar (84)	1 kHz (test 1-10) 2 kHz (test 11-14)
Injector manifold pressure	Parker ASIC 0 - 100 bar	0.2 bar (84)	1 kHz (test 1-10) 2 kHz (test 11-14)
Combustion chamber temperature	K-type thermocouple		50 Hz



APS:	Analog Pressure Sensor	**IV:**	Iginition Valve
BV:	Bleed Valve	**MV:**	Main Valve
CV:	Check Valve	**PRV:**	Pressure Relief Valve
CGR:	Compressed Gas Regulator	**SV:**	Service Valve
CGRV:	Compressed Gas Regulator Valve	**XV:**	Extinguish Valve
DPS:	Digital Pressure Sensor	**XCV:**	Extinguish Check Valve
FV:	Fill Valve		

Figure 7-2: The feed system schematic of the DHX-200 Aurora test engine (between test 6 and test 14). Image based on image published by Knop (80).

The validation of the IPSAT program is focused around the feed system between the run tank and the engine. Since this is only a small part of the overall feed system, as presented in figure 7-2, a more detailed technical drawing of this sub part of the feed system is presented in figure 7-3.This figure shows the inner dimensions of all of the different feed system components that are relevant for validation, except for the curved hoses which are connected to the two oxidizer run tanks.

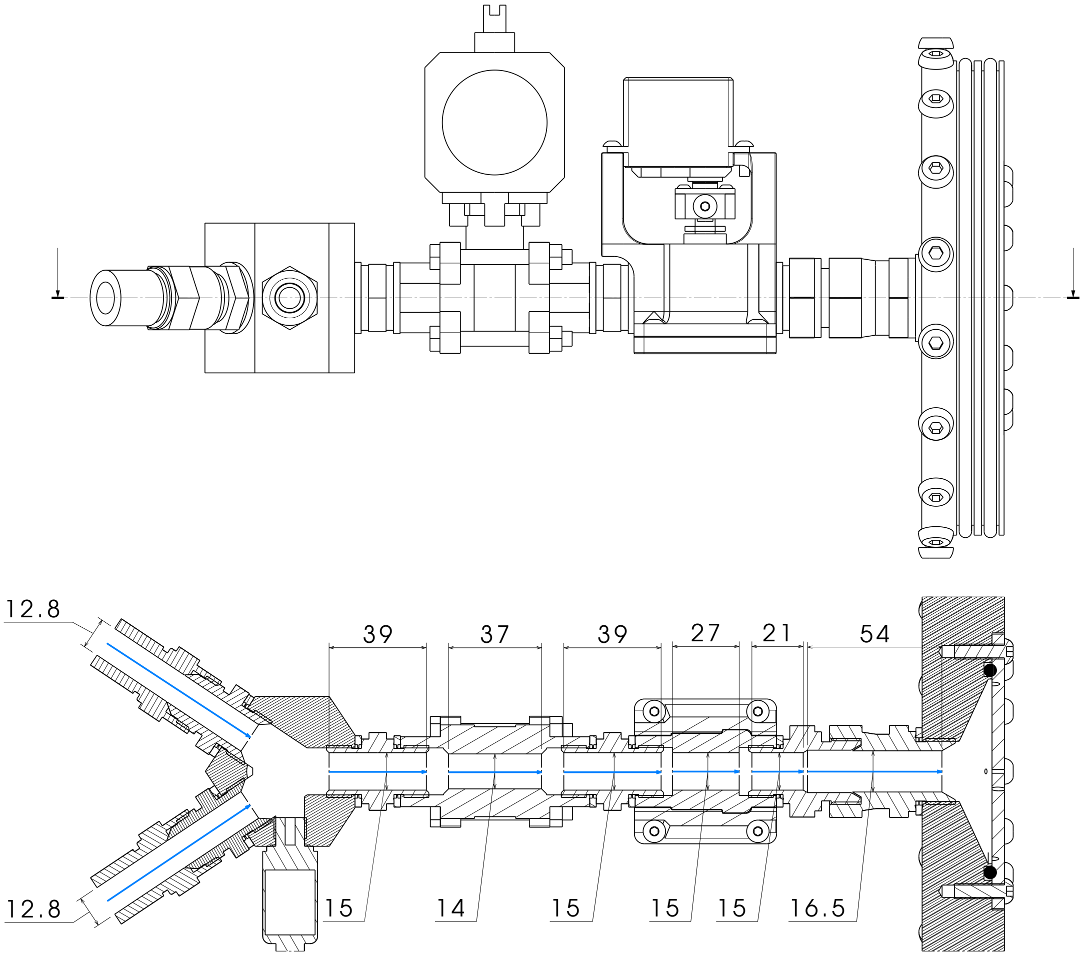


Figure 7-3: (top view) A view of the main feed system section of the DHX-200 Aurora engine. (bottom view) The cross section of the main feed system including dimensions. The sections with the blue arrow show the set of pipe sections which have a constant diameter. All dimensions shown are in millimeters.

7-3 DHX-200 Aurora Test Results

In total, 14 tests were performed with the DHX-200 Aurora engine. Test data was gathered for most of these test. The data set for each test includes:

- Engine mass at the start and end of the burn.

- Tank mass over time.
- Engine thrust force over time.
- Oxidizer run tank pressure over time.
- Pressure inside the feed system junction over time.
- Pressure of the injector manifold over time.
- Pressure of the pre-combustion chamber over time.
- Temperature at various points on the outside of the combustion chamber over time.

For validation of the IPSAT program, only the pressure of the oxidizer tanks, the feed system and the injector manifold are utilized. An overview of all the DHX-200 Aurora tests that were conducted is shown in table 7-3.

Table 7-3: DHX-200 Aurora engine test overview and selected test data.

Test #	**Burntime (s)**	**Engine Failure**	**Selected data range (s)**
1	5.2	No	-
2	9.4	No	-
3	6.1	Yes at 4.7 s	-
4	5.5	Yes at 4.6 s	-
5	N/A	Yes at 0 s	-
6	24.9	No	6.2 - 9.0
7	11.6	Yes at 4.2 s	-
8	11.1	Yes at 4.4 s	-
9	8.7	Yes at 7.6 s	2.8 - 8.0
10	24.5	Yes at 9.1 s	4.0 - 12.0
11	N/A	Yes at 0 s	-
12	10.6	Yes at 4.3 s	-
13	7.9	Yes at 6.4 s	3.0 - 7.0
14	25.2	Yes at 24.1 s	4.0 - 12.0

The data which is considered for the validation of the program is selected according to the following criteria:

- The interval should not contain any quick transients. This includes startup and shutdown transients and any transients that are caused by engine events (e.g. engine failures). The program will be validated in the steady state mode, it will therefore not be able to provide adequate results when modeling quick time dependent events. For the validation analysis performed in this report the data interval should not contain any transient events which can be traced to a specific cause which last for < 0.25 seconds, which is the time interval over which the data is averaged.
- The interval should contain a stable liquid/two-phase flow. Gaseous flow from a relatively small tank volume will have a rapid change of properties over time. This will not be able to be modeled in a steady state mode.
- The interval should be long enough, in time, in order to get a large enough data sample size. For the validation analysis performed in this report a minimum number of 5 data points was defined as being sufficiently large.

Using these selection criteria, test 6, 9, 10, 13 and 14 are selected for the validation of the IPSAT program. The time intervals which have been selected are shown in table 7-3. Each time interval is subdivided into separate equally sized intervals containing 500 data points each. For each of these intervals the mean of all the raw data points within this interval is calculated and that is taken as a data point used for the validation of the program. This is done to reduce the computation time and compensate for instrument noise. It also means that the fast transients that may be present in the pressure data are filtered out.

Table 7-4 shows several parameters for each of the selected tests. Δt_{data} is the total time interval for the selected test data. $\Delta t_{\text{interval}}$ is the time interval between each mean data point. The difference in the length of this time interval between test 10 and test 13 is the increase in sampling rate of the pressure sensors from test 11 onwards (see table 7-2). The data points tab shows the total number of data points with the given time interval for each test. $\dot{m}_{\text{mean}}$ is the mean massflow over the entire selected time interval for each test. The massflow data was obtained by performing a least squares regression analysis on the tank mass data. It was not possible to get accurate mass flow figures for each of the data points within the selected time interval due to the noisy tank mass data. $\Delta p_{\text{f.s., mean}}$ is the mean pressure drop over feed system, defined as the difference between the tank pressure and the manifold pressure, over the entire selected time interval.

Table 7-4: DHX-200 Aurora test data parameters for the selected tests.

Test	Δt_{data}	$\Delta t_{\text{interval}}$	**Data points**	$\dot{m}_{\text{mean}}$	$\Delta p_{\text{f.s., mean}}$
-	(s)	(s)	#	(kg·s^{-1})	(MPa)
6	2.8	0.5	5	3.0610	0.9960
9	5.2	0.5	10	3.2930	0.7095
10	8.0	0.5	16	3.2475	0.7339
13	4.0	0.25	16	2.7866	0.4469
14	8.0	0.25	32	2.8533	0.6131

Figures 7-4 till 7-8 shows the raw pressure sensor data over time for all the selected time intervals. The black dots connected by the black lines show the mean of the surrounding 500 data points for the respective variable. The tank pressure and the injector manifold pressure will serve as an input to the computer program. The feed system pressure will serve as the validation data.

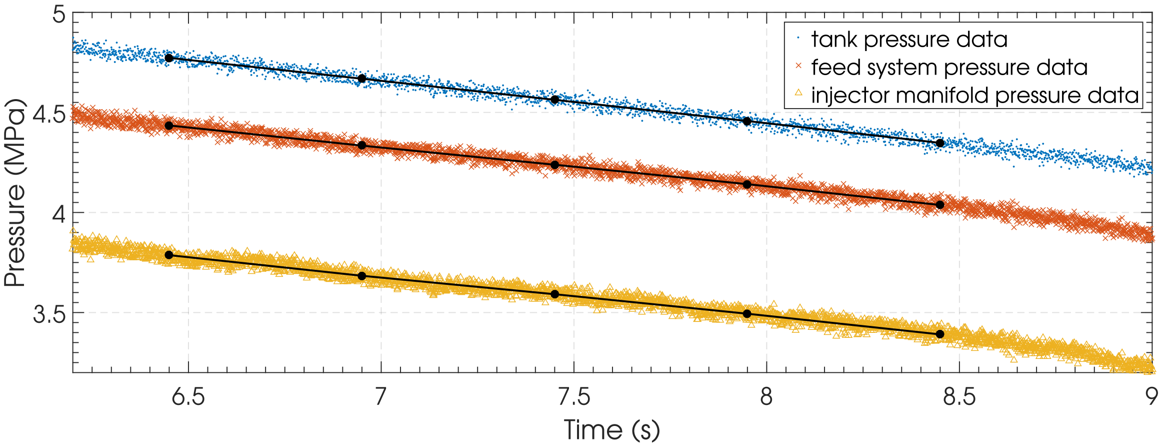


Figure 7-4: Test data from DHX-200 Aurora test 6 showing the data obtained by pressure sensor located in the oxidizer tank, the feed system junction and the injector manifold.

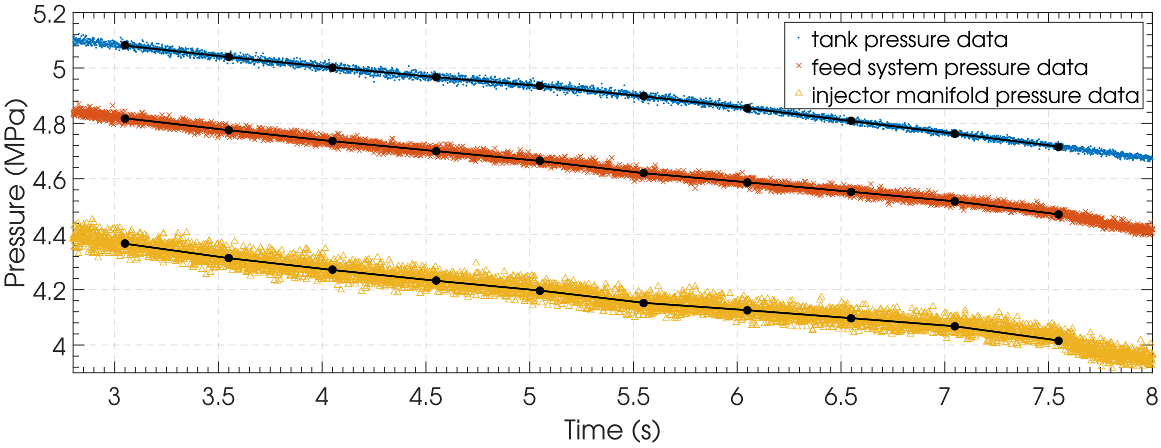


Figure 7-5: Test data from DHX-200 Aurora test 9 showing the data obtained by pressure sensor located in the oxidizer tank, the feed system junction and the injector manifold.

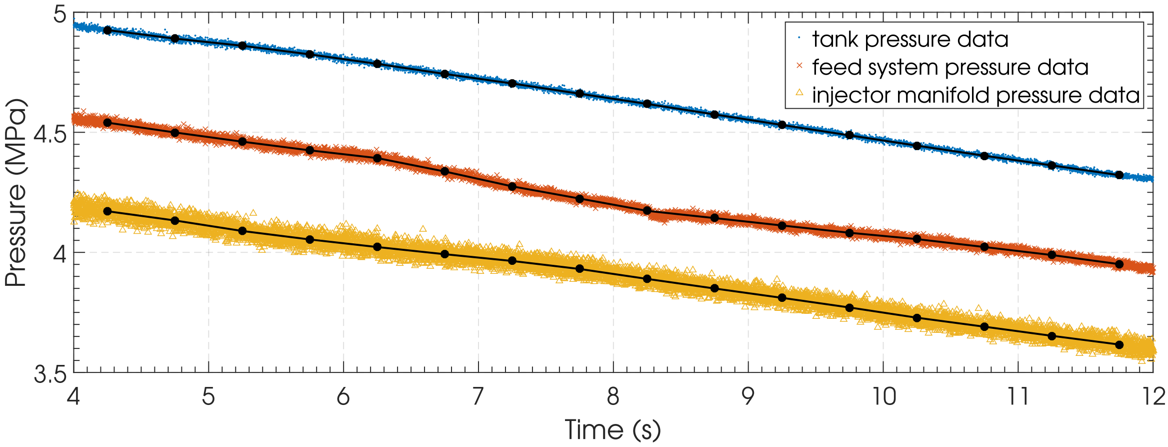


Figure 7-6: Test data from DHX-200 Aurora test 10 showing the data obtained by pressure sensor located in the oxidizer tank, the feed system junction and the injector manifold.

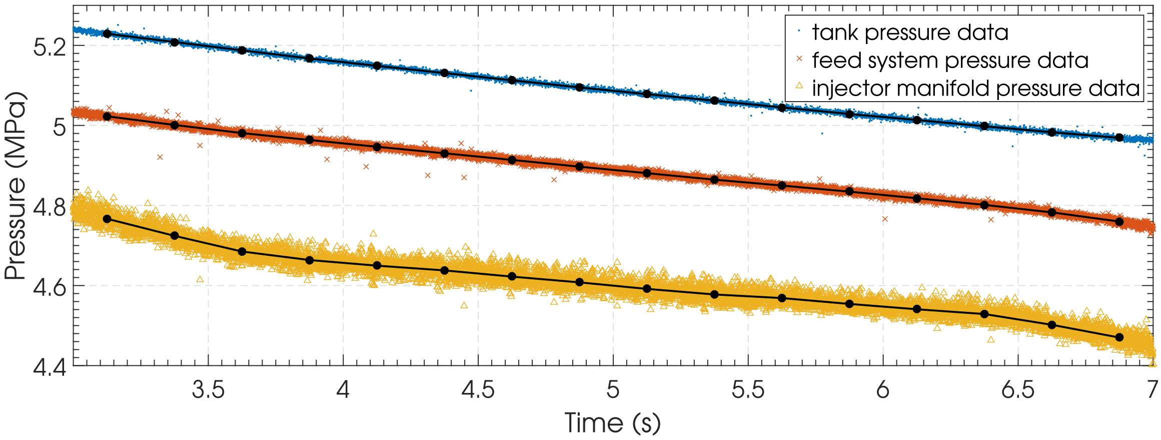


Figure 7-7: Test data from DHX-200 Aurora test 13 showing the data obtained by pressure sensor located in the oxidizer tank, the feed system junction and the injector manifold.

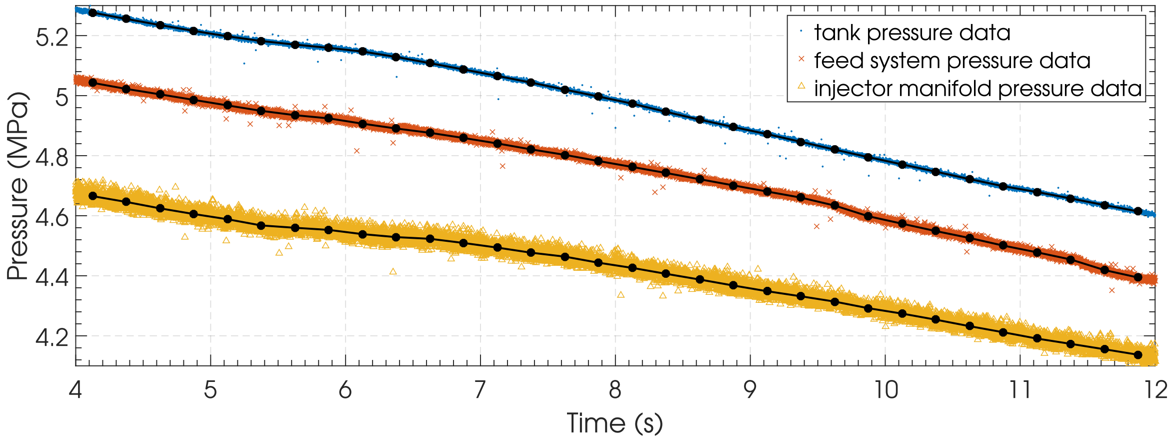


Figure 7-8: Test data from DHX-200 Aurora test 14 showing the data obtained by pressure sensor located in the oxidizer tank, the feed system junction and the injector manifold.

Chapter 8

Computer Program Validation

The final project goal as presented in section 2-3 is that the IPSAT computer program shall be validated against test data. According to the requirement 6.1 in section 2-4 the computer program shall be validated against the DHX-200 Aurora engine test data. This test data has already been presented in chapter 7. The next step is to initiate the validation process. Guidelines for this process have been presented in table 2-6 these steps are:

- 6.1.1 The computer program shall take a systematic diagram of the DHX-200 Aurora static test engine as input.
- 6.1.2 The computer program shall take tank pressure data and combustion pressure data from the DHX-200 Aurora test as input.
- 6.1.3 The computer program shall provide the fluid system response of the DHX-200 Aurora engine as output.
- 6.1.4 The computer program output shall be compared to the original test data.

The first step is to construct a systematic diagram of the part of the DHX-200 Aurora fluid system which is of interest to the validation process. This process is presented in section 8-1. The setup of the IPSAT computer program is presented in section 8-2. The results of the computer program, given the inputs from section 8-2, are presented in section 8-3. Lastly, the the analysis of these results and the comparison with respect to the test data is presented in section 8-4.

8-1 Systematic Diagram of the DHX-200 Aurora Feed System

The DHX-200 Aurora feed system is modeled using a discrete nodes and branches model, the system of nodes and branches is shown in figure 8-1. Each branch represents a pipe with constant diameter in the feed system as shown in figure 7-3.

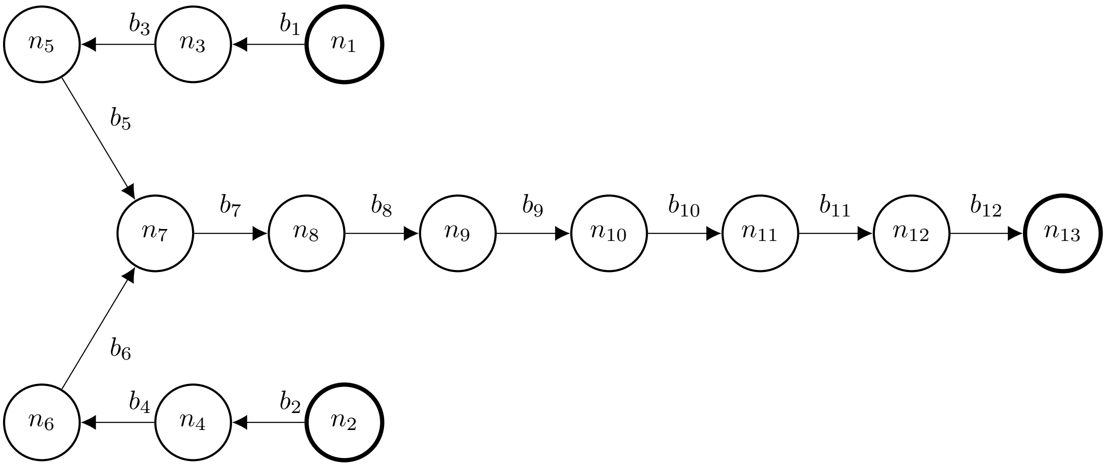


Figure 8-1: DHX-200 Aurora feed system schematic represented by the nodes and branches model. This feed system representation is derived from the cutout presented in figure 7-3.

8-2 Computer Program Setup

This section describes the setup of the model within IPSAT in order to model the DHX-200 Aurora feed system. The program setup includes the definition of the nodes and branches, the settings of the program and the properties of the boundary nodes.

8-2-1 Nodes and Branches settings

The first step in the problem setup is to translate the systematic diagram into the program. This is done by submitting each node and branch separately into the program. Table 8-1 lists all the nodes used in the validation model as shown in figure 8-1. The specie in the boundary nodes is set to nitrous oxide. The pressure in the boundary nodes is set to the measured pressure during the test. The temperature in the boundary nodes is set to the corresponding saturation temperature ± 1 depending on the case which is run.

Table 8-1: The list of nodes, used as an input to IPSAT, derived from the DHX-200 Aurora engine design.

ID	name	type	species	pressure	temperature
n1	Oxidiser_tank	boundary	nitrous_oxide	p_{tank}	$T_s - 1, T_s + 1$
n2	Oxidiser_tank	boundary	nitrous_oxide	p_{tank}	$T_s - 1, T_s + 1$
n3	hose-1-1	internal	-	-	-
n4	hose-2-1	internal	-	-	-
n5	hose-1-2	internal	-	-	-
n6	hose-2-2	internal	-	-	-
n7	Y-piece	internal	-	-	-
n8	MV-2-a	internal	-	-	-
n9	MV-2-b	internal	-	-	-
n10	MV-1-a	internal	-	-	-
n11	MV-1-b	internal	-	-	-
n12	MV-1-c	internal	-	-	-
n13	Manifold	boundary	nitrous_oxide	$p_{manifold}$	$T_s - 1, T_s + 1$

Table 8-2 lists the branches that are submitted into the program. Each branch was modeled as a fluid pipe having a roughness value. The connection tab in table 8-2 shows which nodes are connected by the respective branch, this is done according to the systematic diagram shown in figure 8-1. The length and diameter of each branch is specified according to the engineering drawing of the test setup, as shown in figure 7-3. The surface roughness value was determined using reference roughness values reported by Farshad (85). For the metal parts of the feed system, the roughness value of commercial steel pipes was assumed, which is 46 μm (85). The flexible hoses, denoted by flex_hose_1 and flex_hose_2, are lined by PTFE on the inside. The roughness value for these branches was assumed to be equal to tubes with an internal plastic coating, which is 5 μm (85).

Table 8-2: The list of branches, used as an input to IPSAT, derived from the DHX-200 Aurora engine design.

ID	**name**	**type**	**connection**	**length (mm)**	**diameter (mm)**	**roughness (μm)**
b1	flex_hose_1_fitting	pipe	n1 - n2	70	12.8	46
b2	flex_hose_2_fitting	pipe	n2 - n4	70	12.8	46
b3	flex_hose_1	pipe	n3 - n5	600	15.8	5
b4	flex_hose_2	pipe	n4 - n6	600	15.8	5
b5	flex_hose_1_fitting	pipe	n5 - n7	70	12.8	46
b6	flex_hose_2_fitting	pipe	n6 - n7	70	12.8	46
b7	fitting_1	pipe	n7 - n8	39	15	46
b8	MV-2	pipe	n8 - n9	37	14	46
b9	fitting_2-	pipe	n9 - n10	39	15	46
b10	MV-1	pipe	n10 - n11	27	15	46
b11	fitting_2	pipe	n11 - n12	21	15	46
b12	fitting_3	pipe	n12 - n13	54	16.5	46

8-2-2 Program settings

For each validation data point, the program is setup using the same settings. The program settings used in the validation of IPSAT using the DHX-200 Aurora data is shown in table 8-3.

Table 8-3: Program settings for the validation of IPSAT using the DHX-200 Aurora engine feed system.

Option	**Setting**
Fluid system initialization module settings	
Nodes initialization method	linear interpolation
Branches initialization method	Bernoulli
Solver module settings	
Solver mode	steady state
Iteration min	0
Iteration max	500
Equations	fluid mass, fluid momentum, fluid energy
Variables	pressure, massflow, enthalpy
Conservation of fluid energy	
Solver method	Broyden's method
Relaxation factor	0.1
Convergence limit	10^{-3}
Divergence limit	10^{6}
Equation terms	massflow
Conservation of fluid momentum	
Solver method	Broyden's method
Relaxation factor	0.1
Convergence limit	10^{-3}
Divergence limit	10^{6}
Equation terms	pressure, friction
Conservation of fluid mass	
Solver method	Broyden's method
Relaxation factor	0.1
Convergence limit	10^{-3}
Divergence limit	10^{6}
Equation terms	massflow
Thermophysics module settings	
Variable transform model	Helmholtz MEOS
Equation of state model	Helmholtz MEOS
Melting pressure model	N/A
Vapour pressure model	polylogarithmic formulation
Saturated liquid density model	polylogarithmic formulation
Saturated vapour density model	polylogarithmic formulation
Surface tension model	REFPROP formulation
Viscosity model	saturation polynomial
Thermal conductivity model	N/A

8-2-3 Input Data

The input data for each simulation case are the pressure and temperature of each boundary fluid node. In the DHX-200 Aurora fluid system, as seen in figure 8-1 and table 8-1, these are nodes n1, n2 and n13.

The pressures in those nodes were obtained from the test data, as was presented in figures 7-4 till 7-8 in section 7-3. As was discussed in section 7-3 these pressure values were obtained by averaging the sensor output data over a time period of 1.25 seconds.

Because the temperature of the nitrous oxide was not directly measured in both the tank and the manifold, some assumptions need to be made. The tank is pressurized using the self pressurization properties of nitrous oxide. It is therefore assumed that the nitrous oxide in the tank starts close to the saturation curve and follows that curve. The temperature in the injector manifold is more difficult to determine, however its value has little impact on the rest of the system, because it is located downstream. The temperature in the injector manifold is also set to be close to the saturation temperature. The saturation temperature, for a given pressure, is obtained by inversing equation 5-13 in section 5-1-2.

There is currently no available model which is able to model the viscosity of nitrous oxide for the full range of δ and τ. The model which is used to model the nitrous oxide viscosity are two polynomials which model the saturated vapour viscosity and saturated liquid viscosity separately. Because of this duality, the validation procedure is split into two separate cases. The first case models the fluid as being fully vapour. This is done by moving each data point to the right of the saturation curve by increasing the temperature by 1 degree Kelvin. The second case models the fluid as being fully liquid. This is done by moving each data point to the left of the saturation curve by decreasing the temperature by 1 degree Kelvin. This is done for the properties in both the oxidizer tank and the injector manifold.

Table 8-4 till 8-8 shows the boundary nodes input data for every time interval. The data corresponds to the data points shown in figures 7-4 till 7-8.

Table 8-4: The boundary node input data used to simulate the feed system response of test 6.

#	t	p_{tank}	$T_{\text{s,tank}}$	p_{manifold}	$T_{\text{s,manifold}}$
-	(s)	(MPa)	(K)	(MPa)	(K)
1-1	3.23	4.7751	290.61	3.7891	280.84
1-2	3.48	4.6711	289.66	3.6869	279.72
1-3	3.73	4.5657	288.67	3.5937	278.68
1-4	3.98	4.4593	287.66	3.4965	277.58
1-5	4.23	4.3507	286.61	3.3941	276.39

Table 8-5: The boundary node input data used to simulate the feed system response of test 9.

#	t	p_{tank}	$T_{\text{s,tank}}$	p_{manifold}	$T_{\text{s,manifold}}$
-	(s)	(MPa)	(K)	(MPa)	(K)
2-1	1.52	5.0809	293.33	4.3659	280.84
2-2	1.77	5.0394	292.97	4.3136	279.72
2-3	2.02	5.0018	292.64	4.2710	278.68
2-4	2.27	4.9667	292.33	4.2319	277.58
2-5	2.52	4.9353	291.05	4.1968	276.39
2-6	2.77	4.8979	291.72	4.1522	276.39
2-7	3.02	4.8551	291.34	4.1259	276.39
2-8	3.27	4.8087	290.92	4.0971	276.39
2-9	3.52	4.7625	290.50	4.0678	276.39
2-10	3.77	4.7164	290.08	4.0163	276.39

Table 8-6: The boundary node input data used to simulate the feed system response of test 10.

#	t	p_{tank}	$T_{\text{s,tank}}$	p_{manifold}	$T_{\text{s,manifold}}$
-	(s)	(MPa)	(K)	(MPa)	(K)
3-1	2.12	4.9255	291.97	4.1712	284.83
3-2	2.37	4.8901	291.65	4.1329	284.44
3-3	2.62	4.8604	291.38	4.0891	284.00
3-4	2.87	4.8245	291.06	4.0537	283.63
3-5	3.12	4.7848	290.70	4.0223	283.31
3-6	3.37	4.7431	290.32	3.9924	283.00
3-7	3.62	4.7031	289.95	3.9643	282.70
3-8	3.87	4.6605	289.56	3.9309	282.35
3-9	4.12	4.6180	289.16	3.8895	281.92
3-10	4.37	4.5744	288.75	3.8497	281.49
3-11	4.62	4.5304	288.34	3.8106	281.07
3-12	4.87	4.4873	287.93	3.7705	280.64
3-13	5.12	4.4444	287.52	3.7281	280.17
3-14	5.37	4.4025	287.11	3.6905	279.76
3-15	5.62	4.3622	286.72	3.6526	279.34
3-16	5.87	4.3215	286.32	3.6163	278.94

Table 8-7: The boundary node input data used to simulate the feed system response of test 13.

#	t	$p_{\text{s,tank}}$	$T_{\text{s,tank}}$	p_{manifold}	$T_{\text{s,manifold}}$
-	(s)	(MPa)	(K)	(MPa)	(K)
4-1	3.12	5.2292	294.60	4.7669	290.54
4-2	3.37	5.2084	294.43	4.7242	290.15
4-3	3.62	5.1876	294.25	4.6850	289.79
4-4	3.87	5.1674	294.08	4.6638	289.59
4-5	4.12	5.1488	293.92	4.6497	289.46
4-6	4.37	5.1309	293.76	4.6379	289.35
4-7	4.62	5.1127	293.61	4.6223	289.20
4-8	4.87	5.0950	293.45	4.6080	289.07
4-9	5.12	5.0782	293.31	4.5917	288.92
4-10	5.37	5.0618	293.16	4.5777	288.79
4-11	5.62	5.0451	293.02	4.5687	288.70
4-12	5.87	5.0286	292.87	4.5541	288.56
4-13	6.12	5.0136	292.74	4.5408	288.44
4-14	6.37	4.9979	292.61	4.5289	288.33
4-15	6.62	4.9830	292.47	4.5020	288.07
4-16	6.87	4.9689	292.35	4.4701	287.77

Table 8-8: The boundary node input data used to simulate the feed system response of test 14.

#	t	p_{tank}	$T_{\text{s,tank}}$	p_{manifold}	$T_{\text{s,manifold}}$
-	(s)	(MPa)	(K)	(MPa)	(K)
5-1	4.12	5.2766	295.01	4.6652	289.60
5-2	4.37	5.2555	294.83	4.6460	289.42
5-3	4.62	5.2351	294.65	4.6246	289.23
5-4	4.87	5.2157	294.49	4.6057	289.05
5-5	5.12	5.1976	294.34	4.5883	288.89
5-6	5.37	5.1820	294.20	4.5677	288.69
5-7	5.62	5.1700	294.10	4.5599	288.62
5-8	5.87	5.1592	294.01	4.5528	288.55
5-9	6.12	5.1469	293.90	4.5388	288.42
5-10	6.37	5.1289	293.75	4.5288	288.32
5-11	6.62	5.1082	293.57	4.5232	288.27
5-12	6.87	5.0872	293.39	4.5093	288.14
5-13	7.12	5.0657	293.20	4.4938	287.99
5-14	7.37	5.0435	293.01	4.4769	287.83
5-15	7.62	5.0202	292.80	4.4638	287.70
5-16	7.87	4.9970	292.60	4.4435	287.51
5-17	8.12	4.9733	292.39	4.4263	287.34
5-18	8.37	4.9462	292.15	4.4066	287.15
5-19	8.62	4.9200	291.92	4.3875	286.97
5-20	8.87	4.8955	291.70	4.3675	286.77
5-21	9.12	4.8718	291.49	4.3486	286.59
5-22	9.37	4.8462	291.26	4.3317	286.42
5-23	9.62	4.8201	291.02	4.3140	286.25
5-24	9.87	4.7952	290.79	4.2920	286.03
5-25	10.12	4.7710	290.57	4.2740	285.86
5-26	10.37	4.7468	290.35	4.2538	285.66
5-27	10.62	4.7220	290.13	4.2321	285.44
5-28	10.87	4.6984	289.91	4.2119	285.24
5-29	11.12	4.6789	289.73	4.1908	285.03
5-30	11.37	4.6576	289.53	4.1732	284.85
5-31	11.62	4.6354	289.33	4.1552	284.67
5-32	11.87	4.6143	289.13	4.1369	284.48

8-3 Computer Program Results

The results of IPSAT when using the data as presented in the previous sections are shown in tables 8-9 till 8-13. No issues were reported during the execution of each of simulations covering all the data points. Each simulation case took around 185 iterations to converge

The measured pressure, denoted as p_{meas} is the pressure measured at the junction of the feed system as can be seen in figure 7-3. This point is equivalent to node n7 in the systematic diagram as shown in figure 8-1. The error is defined as the percentage difference between the program results and the measured pressure. The pressure values, $p_{\text{case 1}}$ and $p_{\text{case 2}}$, are defined as the pressure in case of fully vapour flow and the pressure in case of fully liquid flow respectively.

Table 8-9: Results of the IPSAT program for the pressure in n7 compared to the measured pressure from test 6.

#	t	p_{meas}	$p_{\text{case 1}}$	error	$\dot{m}_{\text{case 1}}$	$p_{\text{case 2}}$	error	$\dot{m}_{\text{case 2}}$
-	(s)	(MPa)	(MPa)	(%)	(kg·s^{-1})	(MPa)	(%)	(kg·s^{-1})
1-1	3.23	4.4369	4.4010	-0.8085	3.5386	4.4263	-0.2384	8.1763
1-2	3.48	4.3367	4.2982	-0.8871	3.4790	4.3249	-0.2726	8.1745
1-3	3.73	4.2411	4.1976	-1.0249	3.4031	4.2251	-0.3773	8.1371
1-4	3.98	4.1437	4.0950	-1.1747	3.3320	4.1235	-0.4871	8.1371
1-5	4.23	4.0405	3.9892	-1.2691	3.2650	4.0190	-0.5324	8.0865

Table 8-10: Results of the IPSAT program for the pressure in n7 compared to the measured pressure from test 9.

#	t	p_{meas}	$p_{\text{case 1}}$	error	$\dot{m}_{\text{case 1}}$	$p_{\text{case 2}}$	error	$\dot{m}_{\text{case 2}}$
-	(s)	(MPa)	(MPa)	(%)	(kg·s^{-1})	(MPa)	(%)	(kg·s^{-1})
2-1	1.52	4.8187	4.8016	-0.3545	3.1926	4.8148	-0.0802	7.0520
2-2	1.77	4.7755	4.7568	-0.3924	3.2070	4.7675	-0.1672	7.1814
2-3	2.02	4.7361	4.7174	-0.3941	3.1991	4.7286	-0.1574	7.2074
2-4	2.27	4.7000	4.6810	-0.4048	3.1903	4.6926	-0.1576	7.2286
2-5	2.52	4.6646	4.6483	-0.3491	3.1826	4.6603	-0.0919	7.2478
2-6	2.77	4.6204	4.6084	-0.2595	3.1788	4.6210	0.0126	7.2820
2-7	3.02	4.5875	4.5717	-0.3437	3.1260	4.5840	-0.0769	7.2181
2-8	3.27	4.5532	4.5319	-0.4684	3.0695	4.5438	-0.2074	7.1488
2-9	3.52	4.5187	4.4920	-0.5915	3.0145	4.5035	-0.3361	7.0812
2-10	3.77	4.4718	4.4440	-0.6210	3.0041	4.4561	-0.3503	7.1092

Table 8-11: Results of the IPSAT program for the pressure in n7 compared to the measured pressure from test 10.

#	t	p_{meas}	$p_{\text{case 1}}$	error	$\dot{m}_{\text{case 1}}$	$p_{\text{case 2}}$	error	$\dot{m}_{\text{case 2}}$
-	(s)	(MPa)	(MPa)	(%)	(kg·s^{-1})	(MPa)	(%)	(kg·s^{-1})
3-1	2.12	4.5401	4.6328	2.0420	3.2089	4.6455	2.3217	7.3142
3-2	2.37	4.4993	4.5965	2.1596	3.1974	4.6096	2.4507	7.3300
3-3	2.62	4.4609	4.5618	2.2610	3.2100	4.5757	2.5734	7.3903
3-4	2.87	4.4248	4.5262	2.2907	3.1916	4.5404	2.6121	7.3920
3-5	3.12	4.3962	4.4896	2.2079	3.1567	4.5038	2.5313	7.3624
3-6	3.37	4.3368	4.4523	2.6628	3.1144	4.4663	2.9867	7.3185
3-7	3.62	4.2748	4.4167	3.3197	3.0729	4.4306	3.6441	7.2733
3-8	3.87	4.2241	4.3776	3.6333	3.0354	4.3914	3.9605	7.2391
3-9	4.12	4.1729	4.3356	3.8991	3.0135	4.3497	4.2378	7.2381
3-10	4.37	4.1429	4.2935	3.6357	2.9861	4.3079	3.9817	7.2258
3-11	4.62	4.1120	4.2514	3.3910	2.9566	4.2659	3.7435	7.2087
3-12	4.87	4.0817	4.2096	3.1327	2.9312	4.2243	3.4935	7.1992
3-13	5.12	4.0553	4.1670	2.7542	2.9106	4.1821	3.1262	7.1999
3-14	5.37	4.0223	4.1268	2.5977	2.8835	4.1420	2.9770	7.1845
3-15	5.62	3.9889	4.0875	2.4717	2.8607	4.1030	2.8604	7.1768
3-16	5.87	3.9510	4.0485	2.4683	2.8342	4.0642	2.8647	7.1605

Table 8-12: Results of the IPSAT program for the pressure in n7 compared to the measured pressure from test 13.

#	t	p_{meas}	$p_{\text{case 1}}$	error	$\dot{m}_{\text{case 1}}$	$p_{\text{case 2}}$	error	$\dot{m}_{\text{case 2}}$
-	(s)	(MPa)	(MPa)	(%)	(kg·s^{-1})	(MPa)	(%)	(kg·s^{-1})
4-1	3.12	5.0229	5.0450	0.4398	2.6686	5.0472	0.4842	5.8523
4-2	3.37	5.0012	5.0158	0.2923	2.7198	5.0187	0.3492	5.9815
4-3	3.62	4.9809	4.9880	0.1429	2.7600	4.9914	0.2107	6.0875
4-4	3.87	4.9636	4.9675	0.0778	2.7545	4.9709	0.1476	6.0963
4-5	4.12	4.9467	4.9506	0.0789	2.7353	4.9540	0.1480	6.0747
4-6	4.37	4.9304	4.9351	0.0943	2.7122	4.9384	0.1619	6.0439
4-7	4.62	4.9139	4.9179	0.0807	2.6982	4.9212	0.1484	6.0324
4-8	4.87	4.8970	4.9015	0.0916	2.6823	4.9048	0.1591	6.0163
4-9	5.12	4.8808	4.8849	0.0840	2.6743	4.8882	0.1525	6.0162
4-10	5.37	4.8646	4.8694	0.0994	2.6616	4.8728	0.1680	6.0053
4-11	5.62	4.8498	4.8557	0.1217	2.6349	4.8589	0.1874	5.9643
4-12	5.87	4.8346	4.8399	0.1106	2.6235	4.8431	0.1766	5.9561
4-13	6.12	4.8181	4.8256	0.1560	2.6133	4.8288	0.2223	5.9488
4-14	6.37	4.8016	4.8114	0.2038	2.5972	4.8145	0.2694	5.9294
4-15	6.62	4.7828	4.7919	0.1907	2.6228	4.7954	0.2640	6.0000
4-16	6.87	4.7599	4.7711	0.2343	2.6629	4.7751	0.3183	6.1010

Table 8-13: Results of the IPSAT program for the pressure in n7 compared to the measured pressure from test 14.

#	t	p_{meas}	$p_{\text{case 1}}$	error	$\dot{m}_{\text{case 1}}$	$p_{\text{case 2}}$	error	$\dot{m}_{\text{case 2}}$
-	(s)	(MPa)	(MPa)	(%)	(kg·s^{-1})	(MPa)	(%)	(kg·s^{-1})
5-1	4.12	5.0427	5.0358	-0.1374	3.0672	5.0415	-0.0231	6.6298
5-2	4.37	5.0224	5.0154	-0.1390	3.0532	5.0212	-0.0234	6.6242
5-3	4.62	5.0043	4.9947	-0.1922	3.0464	5.0006	-0.0738	6.6324
5-4	4.87	4.9859	4.9755	-0.2085	3.0365	4.9815	-0.0881	6.6331
5-5	5.12	4.9674	4.9577	-0.1953	3.0268	4.9638	-0.0732	6.6327
5-6	5.37	4.9494	4.9403	-0.1847	3.0314	4.9465	-0.0578	6.6591
5-7	5.62	4.9349	4.9298	-0.1024	3.0163	4.9361	0.0236	6.6409
5-8	5.87	4.9243	4.9205	-0.0781	3.0030	4.9266	0.0470	6.6248
5-9	6.12	4.9067	4.9075	0.0171	3.0014	4.9138	0.1449	6.6349
5-10	6.37	4.8905	4.8926	0.0422	2.9749	4.8987	0.1674	6.5990
5-11	6.62	4.8762	4.8776	0.0280	2.9306	4.8833	0.1466	6.5283
5-12	6.87	4.8590	4.8593	0.0055	2.9048	4.8649	0.1223	6.4963
5-13	7.12	4.8408	4.8401	-0.0153	2.8814	4.8457	0.1003	6.4696
5-14	7.37	4.8210	4.8199	-0.0231	2.8594	4.8254	0.0919	6.4463
5-15	7.62	4.8028	4.8005	-0.0486	2.8254	4.8058	0.0629	6.3980
5-16	7.87	4.7822	4.7784	-0.0796	2.8087	4.7838	0.0328	6.3867
5-17	8.12	4.7625	4.7572	-0.1116	2.7834	4.7625	-0.0006	6.3567
5-18	8.37	4.7424	4.7329	-0.1999	2.7544	4.7381	-0.0905	6.3223
5-19	8.62	4.7216	4.7094	-0.2575	2.7267	4.7145	-0.1496	6.2889
5-20	8.87	4.7011	4.6867	-0.3067	2.7059	4.6917	-0.1990	6.2687
5-21	9.12	4.6806	4.6648	-0.3368	2.6847	4.6699	-0.2296	6.2466
5-22	9.37	4.6604	4.6426	-0.3826	2.6534	4.6474	-0.2781	6.2037
5-23	9.62	4.6339	4.6197	-0.3066	2.6226	4.6244	-0.2045	6.1620
5-24	9.87	4.5983	4.5959	-0.0516	2.6057	4.6007	0.0515	6.1496
5-25	10.12	4.5735	4.5741	0.0135	2.5811	4.5788	0.1153	6.1190
5-26	10.37	4.5489	4.5515	0.0565	2.5619	4.5561	0.1583	6.1002
5-27	10.62	4.5259	4.5279	0.0438	2.5448	4.5325	0.1462	6.0864
5-28	10.87	4.5005	4.5056	0.1135	2.5274	4.5102	0.2163	6.0706
5-29	11.12	4.4781	4.4855	0.1659	2.5239	4.4903	0.2719	6.0820
5-30	11.37	4.4543	4.4657	0.2551	2.5067	4.4704	0.3610	6.0641
5-31	11.62	4.4190	4.4451	0.5904	2.4880	4.4498	0.6963	6.0434
5-32	11.87	4.3941	4.4251	0.7052	2.4732	4.4298	0.8115	6.0303

8-4 Computer Program Results Discussion

The results show that both the fully vapour and fully liquid simulations report lower pressures than was measured for tests 6 and 9 and report higher pressures for tests 10 and 13. Both cases show general good agreement between the simulated fluid system pressure and the test data for all tests except test 10. The average error between the simulated pressure and the actual pressure is <1% for all tests except test 10. Which is within 3 times of the accuracy of the pressure sensors. The program is even able to reproduce the pressure data to < 0.3% for test 14, this is well within the accuracy range of the pressure sensor.

Another observation is that the difference between the fully vapour analysis and the fully liquid analysis is not very significant, in most cases the difference is in the order of 10^3 - 10^4 Pa whilst the pressure is in the order of $4{\cdot}10^6$ Pa. This suggest that the evaluated fluid friction is quite similar in both cases. The model used in this analysis to determine the fluid viscosity of nitrous oxide (63) shows that the viscosity of the saturated liquid and the saturated vapour differ by a factor of 3.5. The flow velocity between the liquid and vapour cases differ by a factor of 3. This means that the Reynolds number of the flow in both cases are almost similar, resulting in a similar friction factor.

The data from test 10 seem to be an out-lier amongst the other results. The difference is about 4 times larger compared to all the other tests and lies well outside the accuracy range of the pressure sensors. The cause of this difference is currently not well understood. Because the difference is relatively consistent during the test and this much of a difference is only seen in this test, it is suggested that the most likely cause of this difference is the sensor. It is likely that the calibration of the pressure sensor or the pressure sensor itself was faulty for this test.

The massflow data between the two cases are very different. This is to be expected because the density of the saturated liquid is about 7 times the density of the saturated vapour around pressures that the engine operates at. Given that the flow velocity of the saturated liquid is around 3 times smaller compared to the saturated vapour, the massflow between the two cases differ by a factor of more than 2. This can be seen in the simulation output data.

Comparing the calculated massflow data to the measured mean massflow, see table 7-4, for each of the tests shows that for every test the case 1 simulation slightly underestimates the massflow and the case 2 heavily overestimates the massflow. This suggests that the density of the fluid during the tests is close to the vapour density. Physically this means that the fluid flowing through the pipes is two-phased. This is in agreement with what is commonly observed in hybrid rocket engine feed systems that employ fluids that are close to the saturation line (86). However, the extend of the vapour fraction that is suggested by these simulation results is doubtful and needs to be separately verified by other tests. The simulated, equivalent, vapour fraction of this two-phase flow could be calculated by combining the measured massflow and the simulated massflows for the two different cases. Unfortunately, there is no information regarding the actual vapour fraction of the fluid at the test, so these values cannot be compared.

The validation analysis shows that the program is capable of accurately modeling the trend of changing pressure in a relatively simple setup. However, it also shows that the dataset is lacking crucial information (e.g. fluid temperature and vapour fraction) in order to confidently conclude that the program is giving the correct thermophsyical data as output. Furthermore, the program is not yet able to make full use of the validation data itself. The prediction of the transients and the combustion part of the data are not yet able to be modeled by the program. It is therefore suggested that future work should focus on improving the program itself, to include more functionality, and validate the program using more and more accurate data sources.

Chapter 9

Conclusions and Recommendations

9-1 Conclusions

The research objective of this thesis report, as stated in chapter 2, is to develop a general purpose fluid system analysis tool for the design of a propulsion system by integrating various modules into one computer program. This section will evaluate to what extend this research objective has been satisfied.

The first step in this thesis project was he definition of the project goals and a number of requirements as presented in chapter 2. DARE was identified as the main customer together with other academic endeavors. In order to gain a historic perspective of the development of similar tools a small literature survey was performed, see chapter 3. The results of this literature survey and the project goals are combined into a top level program design which set the basis of the program, see chapter 4. The functionality of the program is defined by the modules which it contains. Chapter 5 outlined the different modules which currently makes up IPSAT and explained the mathematical foundations of the procedures which are used. The correct implementation of each module was verified by implementing various verification techniques, see chapter 6. The last part of this thesis project was the validation of the program. This was successfully done by comparing the output of the program with the test results of the DHX-200 Aurora rocket engine developed by DARE.

The ultimate goal is to create a modular, flexible and fully transparent propulsion system analysis and design tool which is accessible to both the developer and the designer. In order to verify whether the research objective has been met, a list of six project goals have been defined as stated in chapter 2.

The first project goal states that the program shall be modular. This was achieved to a large extend by the clear definition and implementation of program modules which are explained in further detail in chapter 4. Modules can easily be interchanged using a program settings input file. This was regularly done during the verification and validation phase of the project. The implementation of function libraries in each module allows for an structured and practical way to allow different models and methods to be implemented in the program. Each module can be designed independently as long as the basic module architecture as described in chapter 4 is maintained.

The second project goal states that the program shall provide basic functionality. This was fully achieved by the implementation of five different modules: A system initialization module, a fluid system initialization module, a solver module, a thermophysics module and a fluid friction module. The details of each module are described in chapter 5. Each module is constructed according to the module definitions as stated by the first project goal. Each module has a structured program architecture and contains several different models. The existing modules can be extended by using a straightforward procedure. Future modules can be added by adhering to the module definition as is presented in chapter 4. Both modules and models can be called using the program settings file.

The third project goal states that the program shall be transparent. This was only partially achieved in the current state of the program. The input settings are documented in a structured program settings file. This functions as a basic overview of the models and methods used in the program. However, a full output file summarizing the problem and how it was solved has not yet been implemented. It is expected that this can be done by implementing an output module that takes the program data and converts it into a structured output file. The program architecture has been documented by presenting the top level functionality in chapters 4 and 5. The program code has been organized and largely documented in the individual function scripts. Another goal which was not achieved in the current project is the presentation of the uncertainty and validity of the results obtained by the program. It is expected that this functionality can be added to the proposed output file module.

The fourth project goal states that the program shall be flexible. This was achieved by the implementation of a nodes and branches FVM scheme to model a generic fluid system. This method, presented in section 4, allows the designer to convert a fluid system, as designed in an engineering schematic, into a systematic nodes and branches model which can be analyzed by IPSAT. The designer is free to choose the modules and models/methods which are used to analyze the problem by selecting them in the program settings file.

The fifth project goal states that the program shall be verified. This was achieved by individually verifying the modules and checking whether the models/methods are implemented correctly. The thermophsyics module was verified by checking the output results against source papers and programs using the same models. The fluid friction module was verified by checking the implementation of the models against predefined output values. The solver module was verified by checking the convergence behavior of different methods using a reference problem. The full details of the verification process is presented in chapter 6.

The sixth project goal states that the program shall be validated. This was achieved by comparing the results of the DHX-200 Aurora hybrid engine test against the program output, see chapters 7 and 8. The program is able to model the feed system behavior of the DHX-200 Aurora engine to within the accuracy of the sensors. It is also suggested that this accuracy can be increased with a few additions to the fluid friction module and the thermophysics module. However, the validation process was limited both by the current capabilities of the program and the available validation data set.

It can be concluded that this thesis project has laid the foundation for the creation of a general propulsion system analysis tool with the creation of a basic fluid system solving tool. The most difficult part of creating a general propulsion system design tool have been overcome in this project. The expansion of the functionality of the program is seen as a logical next step in order to meet the ultimate goal of creating a modular, flexible and fully transparent propulsion system analysis and design tool.

9-2 Recommendations

This research project has set the foundation for an extensive fluid system simulation environment. The current version of IPSAT is limited to simple fluid system simulations. It is the intention that the next step in the design of the program is to greatly increase its capabilities. This section will present the recommendations on how to increase the capabilities of the program as envisioned

First the following recommendations can be made with respect to the extension of the modules which are currently implemented in IPSAT, see chapter 5:

- Recommendations regarding the thermophysics module:
  - Adding more species to the specie database will increase the functionality of the program. The following list of species were initially considered for addition but did not make it in the current version of the program:
    - JP-2 / kerosene
    - Hydrogen peroxide
    - Water (for calibration/validation purposes)
  - Currently, the variable transform functions are relatively unstable for data close to the saturation line. Span (42) recommends a variety of procedures in order to increase the stability and accuracy of the obtained solution. It is recommended that these techniques are investigated and implemented.
  - The current evaluation of the multi variable equation of state is relatively slow. This becomes a problem when the analysis involves many fluid nodes and/or many iteration to solve. It is recommended to investigate the optimization of the calculation process in order to speed up the solving process.
  - It is recommended to include simpler equations of state in order to improve the preliminary design capabilities and speed up the design process. Examples include the cubic Soave-Redlich-Kwong (87) or the Peng-Robinson (88) equation of state.
  - In the current thermophysics module there is no two phase flow modeling capabilities. This can be implemented for pure fluids using the procedure described by Span (42).
- Recommendations regarding the solver module:
  - The current implementation of the Newton-Raphson scheme is relatively unstable. It is recommended to implement more sophisticated solving techniques in order to guarentee solver stability over a wider range of problems.
  - In order to increase the capabilities of the program it is recommended to include a solid node and solid branch capable of transferring heat from the fluid by means of conduction and convection.

Next to the recommendations regarding the existing set of modules, there are also several recommended module additions.

- **A chemical reaction module:** A chemical reaction module is a critical element in the design of a propulsion system. This module should be able to calculate the properties of a chemical reaction process similarly to what is done by programs such as CEA (2) and RPA (1). Such a module could be combined with a dedicated combustor library which applies the properties calculated by the chemical reaction subroutine to the properties of a combustor (e.g. liquid rocket engine).

- **A fluid tank module:** A fluid tank module should be able to model the complex dynamics inside a fluid storage tank. One type of tanks which are particularly of interest are the self pressurized tanks that are often used by DARE. The transient behavior of fluid tanks is a well studied phenomenon and a number of methods could be summarized into one fluid tank module.

- **A fluid injection module:** The injector breaks up stream of fluid into small droplets. It is an essential element in any liquid or hybrid rocket engine. There exists a number of empirical methods which can predict the behavior of the injector (see (7) and (9)). These could be implemented in the form of a fluid injection module.

These modules will greatly increase the efficacy of the computer program in future versions of the program. Building these modules can be done as part of future research projects within, or outside of, DARE and the TU Delft.

There are also a number of general recommendations when it comes to the IPSAT computer program. First of all, the current interface with the program is very crude. The designer is currently forced to work with the raw scripting interface of Matlab. It is recommended that a intuitive and effective user interface is developed which can translate the complex inner workings of the program to the designer in a simple format.

Even though the Matlab interface is not particularly difficult to work with, the use of proprietary third party software severely limits the distribution of the computer program. It is recommended that future versions of the program will be written in open source scripting language like Python, or in a programming language like C# such that it can be compiled and run as a standalone program.

Lastly, it is recommended to validate the current version of the program using more data sets from different kinds of tests. The effectiveness of the program is directly linked to the validity of the program results. Each new validation test will increase the knowledge of the capabilities and the limitations of the program. This is essential in keeping the program relevant in the future.

Bibliography

[1] A. Ponomarenko, "RPA - Tool for Rocket Propulsion Analysis," *Space Propulsion Conference*, 2014.

[2] S. Gordon and B. McBride, "Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications I. Analyis," tech. rep., National Aeronautics and Space Administration, 1994.

[3] I. Bell, J. Wronski, S. Quoilin, and V. Lemort, "Pure and Pseudo-pure Fluid Thermophysical Property Evaluation and the Open-Source Thermophysical Property Library CoolProp," *Industrial & Engineering Chemistry Research*, vol. 53, no. 6, pp. 2498–2508, 2014.

[4] E. Lemmon, M. Huber, and M. McLinden, "NIST Reference Fluid Thermodynamic and Transport Properties REFPROP, Version 9.1, User's Guide," tech. rep., National Institute of Standards and Technology, 2013.

[5] R. Sutton, M. Schuman, and W. Chadwick, "Operating Manual for Coaxial Injection Combustion Model," tech. rep., National Aeronautics and Space Administration, 1974.

[6] L. Combs, "Liquid Rocket Combustion Computer Model with Distributed Energy Release, DER Computer Program Documentation and User's Guide Volume I," tech. rep., National Aeronautics and Space Administration, 1974.

[7] J. Muss, T. Nguyen, and C. Johnson, "User's Manual for Rocket Combustor Interactive Design (ROCCID) and Analysis Computer Program Volume I - User's Manual," tech. rep., National Aeronautics and Space Administration, 1991.

[8] J. Portillo, J. Sisco, M. Corless, V. Sankaran, and W. Anderson, "Generalized Combustion Instability Model," *42nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 2006.

[9] D. Hague, R. Reichel, R. Jones, and C. Glatt, "Optimizing a Liquid Propellant Rocket Engine with an Automated Combustor Design Code - AUTOCOM," tech. rep., National Aeronautics and Space Administration, 1971.

[10] A. Majumdar, A. LeClair, R. Moore, and P. Schallhorn, "Generalized Fluid System Simulation Program, Version 6.0," tech. rep., National Aeronautics and Space Administration, 2016.

(11) F. D. Matteo, *Modelling and Simulation of Liquid Rocket Engine Ignition Transients*. PhD thesis, Sapienza University of Rome, 2011.

(12) V. Huijsman, "Stratos II Financial Report," tech. rep., Delft Aerospace Rocket Engineering, 2015.

(13) Anon., "Documentation of Verification, and Accreditation (VV&A) for Models and Simulations MIL-STD-3022," tech. rep., Department of Defense, 2008.

(14) V. Huijsman, "DHX-200 Aurora Engine Configuration IX," tech. rep., Delft Aerospace Rocket Engineering, 2016.

(15) R. Reichel, D. Hague, R. Jones, and C. Glatt, "Program User's Manual for Optimizing the Design of a Liquid Rocket Engine with the Automated combustor Design Code AUTOCOM," tech. rep., National Aeronautics and Space Administration, 1973.

(16) A. Majumdar and K. van Hooser, "A General Fluid System Simulation Program to Model Secondary Flows in Turbomachinery," *31st AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1995.

(17) P. Schallhorn and A. Majumdar, "Numerical Prediction of Pressure Distribution along the Front and Back Face of a Rotating Disc With and Without Blades," *33rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1997.

(18) P. Schallhorn, D. Elrod, D. Goggin, and A. Majumdar, "A Novel Approach for Modeling Long Bearing Squezze Film Damper Performance," *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1998.

(19) P. Schallhorn, A. Majumdar, K. van Hooser, and M. Marsh, "Flow Simulation in Secondary Flow Passages of a Rocket Engine Turbopump," *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1998.

(20) R. Champion and R. Darrow, "X-34 Main Propulsion System Design and Operation," *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1998.

(21) K. Holt, A. Majumdar, T. Steadman, and A. Hedayat, "Numerical Modeling and Test Data Comparison of Propulsion Test Article Helium Pressurization System," *36th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 2000.

(22) A. Majumdar and T. Steadman, "Numerical Modeling of Pressurization of a Propellant Tank," *37th AIAA Aerospace Sciences Meeting Conference and Exhibit*, 1999.

(23) K. van Hooser, J. Bailey, and A. Majumdar, "Numerical Prediction of Transient Axial Thrust and Internal Flows in a Rocket Engine Turbopump," *35th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1999.

(24) A. Hedayat, T. Steadman, T. Brown, K. Knight, C. White, and R. Champion, "Pressurization, Pneumatic, and Vent Subsystems of the X-34 Main Propulsion System," *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhbit*, 1998.

(25) T. Brown, J. McDonald, A. Hedayat, K. Knight, and R. Champion, "Propellant Management and Conditioning WiWith the X-34 Main Propulsion System," *34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 1998.

(26) N. Yamanishi, T. Kimura, M. Takahashi, K. Okita, H. Negishi, and M. Atsumi, "Transient Analysis of the LE-7A Rocket Engine Using the Rocket Engine Dynamic simulator (REDS)," *40th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 2004.

(27) C. Koppel, J. Moral, R. Vara, M. De Rosa, J. Steelant, and P. Omaly, "A Platform Satellite Modelling with EcosimPro: Simulation Results," *45th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, 2009.

(28) A. Isselhorst, "HM7B Simulation with ESPSS Tool on Ariane 5 ESC-A Upper Stage," *46th AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit*, 2010.

(29) V. Huff, S. Gordon, and V. Morrell, "General Method and Thermodynamic Tables for Computation of Equilibrium Composition and Temperature of Chemical Reactions," tech. rep., National Aeronautics and Space Administration, 1951.

(30) S. Gordon, F. Zeleznik, and V. Huff, "A General Method for Automatic Computation of Equilibrium Compositions and Theoretical Rocket Performance of Propellants," tech. rep., National Aeronautics and Space Administration, 1959.

(31) F. Zeleznik and S. Gordon, "An Analytical Investigation of Three General Methods of Calculating Chemical-Equilibrium Compositions," tech. rep., National Aeronautics and Space Administration, 1960.

(32) F. Zeleznik and S. Gordon, "A General IBM 704 or 7090 Computer Program for Computation of Chemical Equilibrium Compositions, Rocket Performance, and Chapman-Jouget Detonations," tech. rep., National Aeronautics and Space Administration, 1962.

(33) R. Svehla and B. McBride, "Fortran IV Computer Program for Calculation of Thermodynamic and Transport Properties of Complex Chemical Systems," tech. rep., National Aeronautics and Space Administration, 1973.

(34) S. Gordon and B. McBride, "Computer Program for Calculation of Complex Chemical Equilibrium Compositions and Applications II. Users Maunual and Program Description," tech. rep., National Aeronautics and Space Administration, 1996.

(35) B. McBride, S. Gordon, and A. Martin, "Thermodynamic Data for Fifity Reference Elements," tech. rep., National Aeronaustics and Space Administration, 1993.

(36) M. Chase, C. Davies, J. Downey, D. Frurip, R. McDonald, and A. Syverud, "JANAF Thermochemical Tables Third Edition," *Analytical Chemistry*, vol. 62, pp. 588A–588A, May 1990.

(37) A. Ponomarenko, "RPA: Tool for Liquid Propellant Rocket Engine Analysis C++ Implementation," tech. rep., RP Software+Engineering UG, 2010.

(38) A. Ponomarenko, "Thermal Analysis of Thrust Chambers," tech. rep., RP Software+Engineering UG, 2012.

(39) A. Ponomarenko, "Assessment of Delivered Performance of Thrust Chamber," tech. rep., RP Software+Engineering UG, 2013.

(40) A. Ponomarenko, "Rocket Propulsion Analysis Version 2.3 User Manual," tech. rep., RP Software+Engineering UG, 2017.

(41) A. Ponomarenko, "Estimation of Engine Mass," tech. rep., RP Software+Engineering UG, 2015.

(42) R. Span, *Multiparameter Equations of State, An Accurate Source of Thermodynamic Property Data*. Springer, 2000.

(43) M. Benedict, G. Webb, and L. Rubin, "An Emperical Equation for Thermodynamic Properties of Light Hydrocarbons and Their Mixtures I. Methane, Ethane, Propane and n Butane," *Journal of Chemical Physics*, vol. 8, pp. 334–345, Apr. 1940.

[44] R. Jacobsen and R. Steward, "Thermodynamic Properties of Nitrogen Including Liquid and Vapor Phases from 63 K to 2000 K with Pressures to 10,000 Bar," *Journal of Physical and Chemical Reference Data*, vol. 2, pp. 757–922, Oct. 1973.

[45] S. Outcalt and M. McLinden, "A Modified Bennedict-Webb-Rubin Equation of State for the Thermodynamic Properties of R152a (1,1-difluoroethane)," *Journal of Physical and Chemical Reference Data*, vol. 25, pp. 605–636, Mar. 1996.

[46] E. Lemmon, R. Jacobsen, S. Penoncello, and D. Friend, "Thermodynamic Properties of Air and Mixtures of Nitrogen, Argon, and Oxygen From 60 to 2000 K at Pressures to 2000 MPa," *Journal of Physical and Chemical Reference Data*, vol. 29, pp. 331–385, May 2000.

[47] E. Lemmon and R. Jacobsen, "Viscosity and Thermal Conductivity Equations for Nitrogen, Oxygen, Argon and Air," *International Journal of Thermophysics*, vol. 25, pp. 21–69, Jan. 2004.

[48] C. Tegeler, R. Span, and W. Wagner, "A New Equation of State for Argon Covering the Fluid Region for Temperature From the Melting Line to 700 K at Pressures up to 1000 MPa," *Journal of Physical and Chemical Reference Data*, vol. 28, pp. 779–850, May 1999.

[49] A. Mulero, I. Cachadina, and M. Parra, "Recommended Correlations for the surface Tension of Common Fluids," *Journal of Physical and Chemical Reference Data.*, vol. 41, p. 043105, Dec. 2012.

[50] D. Bucker and W. Wagner, "A Reference Equation of State for the Thermodynamic Properties of Ethane for Temperatures from the Melting Line to 675 K and Pressures up to 900 MPa," *Journal of Physical and Chemical Reference Data*, vol. 35, pp. 205–266, Mar. 2006.

[51] D. Friend, H. Ingham, and J. Ely, "Thermophysical Properties of Ethane," *Journal of Physical and Chemical Reference Data*, vol. 20, pp. 275–347, Mar. 1991.

[52] J. Schroeder, S. Penoncello, and J. Schoeder, "A Fundamental Equation of State for Ethanol," *Journal of Physical and Chemical Reference Data*, vol. 43, p. 043102, Dec. 2014.

[53] T. Sun, J. Schouten, N. Trappeniers, and S. Biswas, "Accurate Measurement of the Melting Line of Methanol and Ethanol at Pressures up to 270 MPa," *Berichte der Bunsengesellschaft fÃijr physikalische Chemie*, vol. 92, pp. 652–655, May 1988.

[54] M. Assael, E. Sykioti, M. Huber, and R. Perkins, "Reference Correlation of the Thermal Conductivity of Ethanol from the Triple Point to 600 K and up to 245 MPa," *Journal of Physical and Chemical Reference Data*, vol. 42, p. 023102, June 2013.

[55] D. Vega, *A New Wide Range Equation of State for Helium-4.* PhD thesis, Texas A&M University, 2013.

[56] F. Datchi, P. Loubeyre, and R. LeToullec, "Extended and Accurate Determination of the Melting Curves of Argon, Helium, Ice (H2O), and Hydrogen (H2)," *Journal of Physics: Condensed Matter*, vol. 61, pp. 6535 – 6546, Mar. 2000.

[57] J. Leachman, R. Jacobsen, S. Penoncello, and E. Lemmon, "Fundamental Equations of State for Parahydrogen, Normal Hydrogen and Othohydrogen," *Journal of Physical and Chemical Reference Data*, vol. 38, pp. 721–748, Sept. 2009.

(58) M. Assael, J. Assael, M. Huber, R. Perkins, and Y. Takata, "Correlation of the Thermal Conductivity of Normal and Parahydrogen from the Triple Point to 1000 K and up to 100 MPa," *Journal of Physical and Chemical Reference Data*, vol. 40, p. 033101, Sept. 2011.

(59) U. Setzmann and W. Wagner, "A New Eqution of State and Tables of Thermodynamic Properties for Methane Covering the Range from the Melting Line to 625 K at PPressure up tp 1000 MPa," *Journal of Physical and Chemical Reference Data*, vol. 20, pp. 1061–1155, Nov. 1991.

(60) R. Kleinrahm and W. Wagner, "Measurement and Correlation of the Equilibrium Liquid and Vapour Densities and the Vapour Pressure Along the Coexistence Curve of Methane," *The Journal of Chemical Thermodynamics*, vol. 18, pp. 739–760, Aug. 1986.

(61) R. Span, E. Lemmon, R. Jacobsen, W. Wagner, and A. Yokozeki, "A Reference Equation of State for the Thermodynamic Properties of Nitrogen for Temperatures from 63.151 to 1000 K and Pressures to 2200 MPa," *Journal of Physical and Chemical Reference Data*, vol. 29, pp. 1361–1433, Nov. 2000.

(62) E. Lemmon, "Short Fundamental Equations of State for 20 Industrial Fluids," *Journal of Chemical & Engineering Data*, vol. 51, pp. 785–850, Apr. 2006.

(63) C. Beaton, J. Walton, and G. Walter, "Thermophysical Properties of Nitrous Oxide," tech. rep., IHS ESDU, 1991.

(64) R. Steward, R. Jacobsen, and W. Wagner, "Thermodynamic Properties of Oxygen from the Triple Point to 300 K with Pressures to 80 MPa," *Journal of Physical and Chemical Reference Data*, vol. 20, pp. 917–1021, Sept. 1991.

(65) U. Setzmann and W. Wagner, "A New Method for Optimizing the Structures of Thermodynamic Correlation Equations," *International Journal of Thermophysics*, vol. 10, pp. 1103–1126, Apr. 1989.

(66) W. Wagner, "New Vapour Pressure Measurements for Argon and Nitrogen and a New Method for Establishing Rational Vapour Pressure Equations," *Cryogenics, Volume 13, Issue 8*, vol. 13, pp. 470–482, Aug. 1973.

(67) F. Simon and G. Glatzel, "Bemerkungen zur Schmelzdruckkurve," *Zeitschrift fÃijr Anorganische Chemie*, vol. 178, pp. 309–316, Jan. 1928.

(68) V. Kechin, "Thermodynamically Based Melting-Curve Equation," *Journal of Physics: Condensed Matter*, vol. 7, pp. 531–535, Feb. 1995.

(69) V. Vesovic, W. Wakeham, G. Olchowy, J. Sengers, J. Watson, and J. Millat, "The Transport Properties of Carbon Dioxide," *Journal of Physical and Chemical Reference Data*, vol. 19, pp. 763–808, May 1990.

(70) H. Hanley, R. McCarty, and H. Intemann, "The Viscosity and Thermal Conductivity of Dilute Gaseous Hydrogen from 15 to 5000 K," *Journal of Research of the National Bureau of Standards*, vol. 74A, pp. 331–353, May 1970.

(71) G. Olchowy and J. Sengers, "A Simplified Representation for the Thermal Conductivity of Fluids in the Critical Region," *International Journal of Thermophysics*, vol. 10, pp. 417–426, Mar. 1989.

(72) S. Haaland, "Simple and Explicit Formulas for the Friction Factor in Turbulent Pipe Flow," *Journal of Fluids Engineering*, vol. 105, pp. 89–90, Mar. 1983.

(73) F. White, *Viscous Fluid Flow*. McGraw-Hill, 1991.

(74) R. Shah and A. London, "Laminar Flow Forced Convection Heat Transfer and Flow Friction in Straight and Curved Ducts - A Summary of Analytical Solutions," tech. rep., Office of Naval Research, 1971.

(75) C. Broyden, "A Class of Methods for Solving Nonlinear Simultaneous Equations," *Mathematics of Computation*, vol. 19, no. 92, pp. 577–593, 1965.

(76) R. Brent, *Algorithms for Minimization without Derivatives*. Prentice-Hall, 1973.

(77) F. Farshad, H. Rieke, and J. Garber, "New Developments in Surface Roughness Measurements, Characterization, and Modeling Fluid Flow in Pipe," *Journal of Pertoleum Science and Engineering*, vol. 29, pp. 139–150, Apr. 2001.

(78) D. Brkic, "Solution of the Implicit Colebrook Equation for Flow Friction Using Excel," *Spreadsheets in Education (eJSiE)*, vol. 10, Aug. 2017.

(79) V. Huijsman, "DHX-200 Aurora Engine Configuration IX Engine Datasheet," tech. rep., Delft Aerospace Rocket Engineering, 2016.

(80) T. Knop, J. Wink, R. Huijsman, S. Powell, R. Werner, J. Ehlen, K. Samarawickrama, B. Zandbergen, and A. Cervone, "Failure Mode Investigation of a Sorbitol-based Hybrid Rocket Flight Motor for the Stratos II Sounding Rocket," *51st AIAA/SAE/ASEE Joint Propulsion Conference*, 2015.

(81) J. Wink, T. Knop, R. Huijsman, S. Powell, K. Samarawickrama, A. Fraters, R. Werner, C. Becker, F. Lindemann, J. Ehlen, A. Cervone, and B. Zandbergen, "Test Campaign on a 10 kN Class Sorbitol-Based Hybrid Rocket Motor for the Stratos II Sounding Rocket," *Space Propulsion Conference*, 2014.

(82) Anon, "KAS Force Tansducer," tech. rep., Angewandte System Technik Gruppe, 2017.

(83) Anon, "Tedea Huntleigh Load Cell Catalog," tech. rep., Tedea-Huntleigh, 2014.

(84) Anon, "Pressure Tranducers and Transmitters ASIC 'Performer'," tech. rep., Parker, 2007.

(85) F. Farshad and H. Rieke, "Surface-Roughness Design Values for Modern Pipes," *SPE Drilling & Completion*, vol. 21, pp. 212–215, Sept. 2006.

(86) B. Waxman, J. Zimmerman, and B. Cantwell, "Mass Flow Rate and Isolation Characteristics of Injectors for Use with Self-Pressurizing Oxidizers in Hybrid Rockets," *49th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, 2013.

(87) G. Soave, "Equilibrium Constants from a Modified Redlich-Kwong Equation of State," *Chemical Engineering Science*, vol. 27, pp. 1197–1203, June 1972.

(88) D. Peng and D. Robinson, "A New Two-Constant Equation of State," *Industrial & Engineering Chemistry Fundamentals*, vol. 15, pp. 59–64, Feb. 1976.

(89) O. Redlich and J. Kwong, "On the Thermodynamics of Solutions," *Chemical Reviews*, vol. 44, pp. 233–244, Feb. 1949.

Appendix A

Calculation of δ and τ for Two Given Thermodynamic Variables

The Helmholtz multiparameter equation of state is designed to treat the thermodynamic variables z, p, u, h, s, c_p, c_v and w as dependent variables and ρ and T as independent variables. Often, the problem is to find the complete set of thermodynamic variables as a function of 2 thermodynamic variables which are not temperature and/or density. In this case one or two of the equations listed in table 5-5 need to be reversed in order to determine the temperature or density corresponding to the value of the given thermodynamic state property.

The general problem is thus to find the set of τ and δ which corresponds to the values of the two given thermodynamic variables, which are not temperature or pressure or both. This is not a straightforward problem and requires an iterative process, because the equations in table 5-5 cannot be reversed to give δ and τ. The procedure, which is followed by IPSAT, is to turn this problem into a root finding problem, where the problem is to find the root of the following function:

$$F - f(\delta, \tau) = 0 \tag{A-1}$$

Where F is equal to the value of the thermodynamic state which is given and $f(\delta, \tau)$ is the evaluation of the Helmholtz energy derivative corresponding to the given state F (see table 5-5). In the case that either temperature or density is given, equation A-1 is sufficient to provide the other independent variable (density or temperature). In the case that neither temperature nor density is given the problem is more complex and an additional root finding equation, for the second thermodynamic variable, needs to be solved simultaneously.

When trying to find the root of equation A-1, the search may find multiple roots depending on the initial guess of the missing thermodynamic variable. This is illustrated in figure A-1, where for a given pressure and temperate up to 5 different roots of equation A-1 may exist.

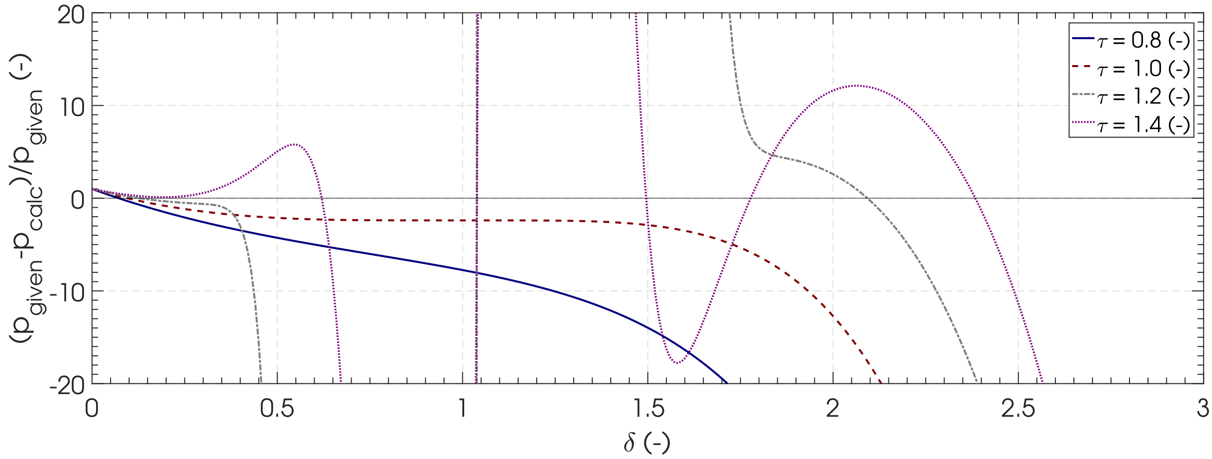


Figure A-1: The function $F - f(\delta, \tau)$ for different values of δ and τ where, F is pressure and $f(\delta, \tau)$ is equal to equation 5-6 in table 5-5. The pressure is set to 10^6 Pa and the substance is nitrogen. The roots of this function represent combinations of δ and τ where equation 5-6 gives 10^6 Pa.

This introduces another problem when trying to find the correct value of δ and τ. A good initial guess of either δ or τ is essential to have the root finding function to converge to the correct root. The remaining part of this appendix will explain the methods used by IPSAT to find a good estimate for δ or τ.

Three different cases will be described in more detail. These three cases are all used by IPSAT for the calculation of a thermodynamic state. The three cases include the transform of pressure and temperature, described in section A-1. The transform of pressure and density, described in section A-2. The transform of pressure and enthalpy, described in section A-3.

A-1 Pressure and Temperature as the Independent Variables

When pressure and temperature are given as the independent variables the root finding problem is specified by:

$$p - f_p(\delta, \tau) = 0 \quad \rightarrow \quad p - \rho RT \left[1 + \delta \left(\frac{\partial \alpha^r}{\partial \delta} \right)_\tau \right] = 0 \tag{A-2}$$

The task is to find a good initial guess of the density which covers the entire fluid range for different fluids. Span (42) recommends to use the soave-redlich-kwong cubic equation of state (87, 89) to get a an approximation of the fluid density for a given temperature. This is also the method used by IPSAT. The soave-redlich-kwong cubic equation of state is given by:

$$p(v, T) = \frac{RT}{v - b} - \frac{a(\tau)}{v(v + b)} \tag{A-3}$$

Where v is the specific molar volume (which is the inverse of the molar density), R is the universal gas constant, T is the fluid temperature and a and b are constants which are given by:

$$a(\tau) = 0.42747\frac{R^2T_c^2}{p_c}\left(1 + m\left(1 - \left(\frac{1}{\tau}\right)^{0.5}\right)\right)^2, \quad \text{and} \quad b = 0.08664\frac{RT_c}{p_c} \tag{A-4}$$

Where T_c is the temperature at the critical point, p_c is the pressure at the critical point, τ is the inverse reduced temperature and m is given by:

$$m = 0.480 + 1.574\omega - 0.176\omega^2 \tag{A-5}$$

Where ω is the acentric factor which is defined to be:

$$\omega = -\log_{10}\left(\frac{p_v(\tau = 1/0.7)}{p_c}\right) - 1 \tag{A-6}$$

Where $p_v(\tau)$ is the vapor pressure for a given τ. In order to get the density out of the soave-redlich-kwong equation of state, we need to define the compressibility factor to be:

$$Z = \frac{p}{\rho RT} \quad \rightarrow \quad v = \frac{1}{\rho} = \frac{ZRT}{p} \tag{A-7}$$

Substituting equation A-7 into equation A-3 will give the following cubic equation:

$$Z^3 - Z^2 + (A - B - B^2)Z - AB = 0 \tag{A-8}$$

Where A and B are defined as:

$$A = \frac{ap}{R^2T^2}, \quad \text{and} \quad B = \frac{bp}{RT} \tag{A-9}$$

In order to find the density, which is captured by Z through equation A-7, the roots of equation A-8 needs to be found. Substituting $Z = Y + 1/3$ into equation A-8 gives the following reduced cubic:

$$Y^3 + rY + q = 0 \tag{A-10}$$

Where r and q are defined as:

$$r = (A - B - B^2) - \frac{1}{3}, \quad \text{and} \quad q = -\frac{2}{27} + \frac{1}{3}(A - B - B^2) - AB \tag{A-11}$$

The discriminant of this cubic equation is given by:

$$D = \left(\frac{r}{3}\right)^3 + \left(\frac{q}{2}\right)^2 \tag{A-12}$$

The number of real roots depend on the value of this discriminant:

- $D > 0$ In this case there is one real root and 2 complex roots.
- $D < 0$ In this case there are three real roots where the largest value of Y corresponds to the vapor phase and the smallest value of Y corresponds to the liquid phase.

- $D = 0$ in this case there is a multiple root and all roots are real.

In the case that $D > 0$ the three roots of the cubic equation can be found by first calculating the value of C, Where C is given by:

$$C = \sqrt[3]{-\frac{q}{2} + \sqrt{D}} \tag{A-13}$$

The three roots can then be found with the following relation:

$$Y_i = \zeta^k C - \frac{r}{\zeta^k C} \quad \text{with} \quad i = 1, 2, 3 \quad \text{and} \quad k = 0, 1, 2 \tag{A-14}$$

Where ζ is the complex cube root of unity:

$$\zeta = -\frac{1}{2} + \frac{\sqrt{3}}{2} i \tag{A-15}$$

Usually the real root corresponds to $k = 0$. However in some cases where $D > 0$ and $q > \sqrt{D}/2$ the first root will be complex. This this case, the second root, where $k = 1$, yields a real value. Due to rounding errors the imaginary part does not disappear when multiplying C with ζ by computer. Therefore, one has to dismiss the imaginary part from the resulting complex number to use this root.

In the case that $D \leq 0$ there are three real roots. These roots can be found using the trigonometric solution method, where:

$$\theta = \sqrt[3]{-\frac{r^3}{27}}, \quad \phi = \cos^{-1}\left(-\frac{q}{2\theta}\right) \tag{A-16}$$

The three roots can be found by the following relation:

$$Y_i = 2\sqrt[3]{\theta} \cos\left(\frac{\phi}{3} + \frac{2k\pi}{3}\right) \quad \text{with} \quad i = 1, 2, 3 \quad \text{and} \quad k = 0, 1, 2 \tag{A-17}$$

In case $D \approx 0$ and $r \approx 0$ equation A-17 becomes undefined. If both values are getting significantly close to zero, it indicates that the root is close to the critical region and Span advises to use the critical density as a guess in that case (42). In all other cases the guess for the density can be obtained by substituting the value of the root into:

$$\rho_{guess} = \frac{p}{RT\left(Y + 1/3\right)} \tag{A-18}$$

A-2 Pressure and Density as the Independent Variables

Similarly to the previous case the root finding problem is specified by:

$$p - f_p(\delta, \tau) = 0 \quad \rightarrow \quad p - \rho RT\left[1 + \delta\left(\frac{\partial \alpha^r}{\partial \delta}\right)_\tau\right] = 0 \tag{A-19}$$

The task is to find a good initial guess of the temperature given the density and pressure to initiate the root finding method. Span recommends to use the original van der Waals equation of state for its simplicity and sufficient accuracy (42). The original van der Waals equation can be written as:

$$\left(p + \frac{a}{v}\right)(v - b) = RT \quad \rightarrow \quad p(v, T) = \frac{RT}{v - b} - \frac{a}{v^2} \tag{A-20}$$

Where v is the specific molar volume (which is the inverse of the molar density), R is the universal gas constant, T is the fluid temperature and a and b are constants which are given by:

$$a = \frac{27}{64}\frac{R^2T^2}{p_c}, \quad \text{and} \quad b = \frac{1}{8}\frac{RT_c}{p_c} \tag{A-21}$$

Inverting equation A-20 gives the following relation for the temperature:

$$T_{guess} = \frac{(p + a\rho^2)(1/\rho - b)}{R} \tag{A-22}$$

The value for T_{guess} can directly be substituted in a Newton-Raphson method, or modified Brent's method (described in section 5-3-1), to solve equation A-19. When using a Regula Falsi, or Brent's method in its original form, a bracketed interval is needed. In this case the two values for T which bracket an interval is given by: $T_{1,2} = T_{guess} * (1 \pm a)$, where an appropriate value for a is equal to 0.05 according to Span (42).

A-3 Pressure and Enthalpy as the Independent Variables

In this case, both δ and τ have to be determined. This means that the root finding method will consist of finding the root of two dependent equations. One option would be to use a two equations Newton-Raphson scheme, as described in section 5-3-1 with the following two equations:

$$p - f_p(\delta, \tau) = 0 \quad \rightarrow \quad p - \rho RT\left[1 + \delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau\right] = 0 \tag{A-23}$$

$$h - f_h(\delta, \tau) = 0 \quad \rightarrow \quad h - \tau RT\left[\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta + \left(\frac{\partial\alpha^r}{\partial\tau}\right)_\delta\right] + \delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau + 1 = 0 \tag{A-24}$$

This would find both roots relatively quickly since both equations are solved simultaneously. However, this method implies that a good initial guess for both δ and τ can be found for the given pressure and enthalpy. A good initial guess can be made when the fluid is far from both the critical point and the saturation line. However, this root finding method can become unstable close to the critical point and the saturation line.

Span recommends to do a two stage root finding method for increased stability and to obtain a better guess for δ and τ (42). This method first acquires an initial guess for τ using the information provided by the given pressure and enthalpy. This estimate of τ is then iterated to a final value of τ which is then used to find the final value of δ using the method described in section A-1. Span recommends the following method to determine the value for τ for the given pressure and enthalpy:

- If $p > p_c$ then calculate τ_0 using the method described in section A-2 where $\delta = \delta_c$. Next calculate the enthalpy h_0 that corresponds to $\tau = \tau_0$ and $\delta = \delta_c$.

  - If $h > h_0$ then τ must be smaller than τ_0. A guess for τ, that is sufficiently close to the root, can be found by steadily decreasing τ_{guess} by using the following relation: $\tau_{guess,i+1} = \tau_i/(1+a)$ where $a = 0.2$. Each value for $\tau_{guess,i}$ is then substituted into the relation described in section A-1 to find for which $\tau_{guess,i}$ the residual changes sign. When this is the case, a root is close and the value of $\tau_{guess,i}$ can be used as a guess.
  - If $h < h_0$ then τ must be larger than τ_0. A guess for τ, that is sufficiently close to the root, can be found by steadily increasing τ_{guess} by using the following relation: $\tau_{guess,i+1} = \tau_i/(1+a)$ where $a = -0.1$. Each value for $\tau_{guess,i}$ is then substituted into the relation described in section A-1 to find for which $\tau_{guess,i}$ the residual changes sign. When this is the case, a root is close and the value of $\tau_{guess,i}$ can be used as a guess.
  - If $h < h_0$ and the enthalpy and $\tau_{guess} > \tau_m$ the given enthalpy and pressure lies outside the valid equation of state region.

- If $p > p_c$ then calculate the saturation temperature and densities, τ_s, $\delta_{s,liq}$ and $\delta_{s,vap}$ corresponding to the given pressure. Next calculate the corresponding saturation enthalpies $h_{s,liq}(\tau_s, \delta_{s,liq})$ and $h_{s,vap}(\tau_s, \delta_{s,vap})$

  - If $h > h_{s,vap}$ then τ must be smaller than τ_s. A guess for τ, that is sufficiently close to the root, can be found by steadily decreasing τ_{guess} by using the following relation: $\tau_{guess,i+1} = \tau_i/(1+a)$ where $a = 0.2$. Each value for $\tau_{guess,i}$ is then substituted into the relation described in section A-1 to find for which $\tau_{guess,i}$ the residual changes sign. When this is the case, a root is close and the value of $\tau_{guess,i}$ can be used as a guess.
  - If $h < h_{s,liq}$ then τ must be larger than τ_s. A guess for τ, that is sufficiently close to the root, can be found by steadily increasing τ_{guess} by using the following relation: $\tau_{guess,i+1} = \tau_i/(1+a)$. Span recommends to use the following value of a to avoid increasing the value of τ by too much: $a = -(\tau_s - \tau_m)/(b\tau_s)$ where $b = 5$. Each value for $\tau_{guess,i}$ is then substituted into the relation described in section A-1 to find for which $\tau_{guess,i}$ the residual changes sign. When this is the case, a root is close and the value of $\tau_{guess,i}$ can be used as a guess.
  - If $h_{s,liq} < h < h_{s,vap}$ then the fluid is in the two phase region. In this case $\tau = \tau_s$ and the value of τ_s has to be determined from the equation of state. Then the value of $h_{s,liq}$ and $h_{s,vap}$ has to be determines from the equation of state. With the values of $h_{s,liq}$ and $h_{s,vap}$ the vapour fraction can be determined from the following equation:

    $$f(\tau, x) = f_{s,liq}(\tau) + x(f_{s,vap}(\tau) - f_{s,liq}(\tau)) \tag{A-25}$$

    Equation A-25 can then be used to evaluate the two phase equilibrium values for all the other thermo physical properties (S, c_v, c_p, etc) where $f_{s,vap}$ and $f_{s,liq}$ can be determined by substituting $\tau = \tau_s$ and $\delta = \delta_{s,vap}$ or $\delta = \delta_{s,liq}$. Where the value of $\delta_{s,vap}$ and $\delta_{s,liq}$ have to be obtained from the equation of state. The density of the two phase mixture can be obtained using the following relation:

    $$\rho(\tau, x) = \left[1/\rho_{s,liq}(\tau) + x\left(1/\rho_{s,vap}(\tau) - 1/\rho_{s,liq}(\tau)\right)\right]^{-1} \tag{A-26}$$

    Care should be taken when the fluid is inside the two phase region, since the properties are non-uniform. This means that the thermophysical properties in this regime have a high level of uncertainty.

Appendix B

IPSAT Data structure summary

Table B-1: Summary of all available variables in the struct (Nodes). (Nodes) contains the information of all the nodes in the system. All customizable field names are listed between square brackets.

Variable name	**Type**	**Unit**
(Nodes).(ID).name	string	N/A
(Nodes).(ID).type	string	N/A
(Nodes).(ID).connectivity	string	N/A
(Nodes).(ID).properties		
(Nodes).(ID).properties.species	string	N/A
(Nodes).(ID).properties.pressure	double	Pa
(Nodes).(ID).properties.temperature	double	K
(Nodes).(ID).properties.density	double	kg·m^{-3}
(Nodes).(ID).properties.melting_pressure	double	Pa
(Nodes).(ID).properties.vapor_pressure	double	Pa
(Nodes).(ID).properties.rho_l_sat	double	kg·m^{-3}
(Nodes).(ID).properties.rho_v_sat	double	kg·m^{-3}
(Nodes).(ID).properties.compressibility	double	-
(Nodes).(ID).properties.internal_energy	double	J·kg^{-1}
(Nodes).(ID).properties.enthalpy	double	J·kg^{-1}
(Nodes).(ID).properties.entropy	double	J·kg^{-1}·K^{-1}
(Nodes).(ID).properties.cv	double	J·kg^{-1}·K^{-1}
(Nodes).(ID).properties.cp	double	J·kg^{-1}·K^{-1}
(Nodes).(ID).properties.gamma	double	-
(Nodes).(ID).properties.speed_of_sound	double	m·s^{-1}
(Nodes).(ID).properties.state	string	N/A
(Nodes).(ID).properties.surface_tension	double	N·m^{-1}
(Nodes).(ID).properties.viscosity	double	Pa·s
(Nodes).(ID).solver		
(Nodes).(ID).solver.equations.fluid_mass.residual	double	kg·s^{-1}
(Nodes).(ID).solver.equations.fluid_mass.massflow.value	double	kg·s^{-1}
(Nodes).(ID).solver.equations.fluid_mass.transient.value	double	kg·s^{-1}
(Nodes).(ID).solver.equations.fluid_energy.residual	double	J·s^{-1}
(Nodes).(ID).solver.equations.fluid_energy.massflow.value	double	J·s^{-1}

Variable name	Type	Unit
(Nodes).(ID).solver.equations.fluid_energy.friction.value	double	J·s^{-1}
(Nodes).(ID).solver.equations.fluid_energy.transient.value	double	J·s^{-1}
(Nodes).system		
(Nodes).system.names.total	cell	N/A
(Nodes).system.names.internal	cell	N/A
(Nodes).system.names.boundary	cell	N/A
(Nodes).system.names.ambient	cell	N/A
(Nodes).system.names.solid	cell	N/A
(Nodes).system.amount.total	integer	N/A
(Nodes).system.amount.internal	integer	N/A
(Nodes).system.amount.boundary	integer	N/A
(Nodes).system.amount.ambient	integer	N/A
(Nodes).system.amount.solid	integer	N/A

Table B-2: Summary of all available variables in the struct (Branches). (Branches) contains the information of all the branches in the system. All customizable field names are listed between square brackets.

Variable name	Type	Unit
(Branches).(ID).name	string	N/A
(Branches).(ID).type	string	N/A
(Branches).(ID).connectivity	cell	N/A
(Branches).(ID).geometry		
(Branches).(ID).geometry.type	string	N/A
(Branches).(ID).geometry.length	double	m
(Branches).(ID).geometry.diameter	double	m
(Branches).(ID).geometry.area	double	m^2
(Branches).(ID).geometry.epsilon	double	m
(Branches).(ID).geometry.flow_coefficient	double	-
(Branches).(ID).geometry.a	double	m
(Branches).(ID).geometry.b	double	m
(Branches).(ID).geometry.D1	double	m
(Branches).(ID).geometry.D2	double	m
(Branches).(ID).geometry.d1	double	m
(Branches).(ID).geometry.d2	double	m
(Branches).(ID).geometry.Lor	double	m
(Branches).(ID).properties		
(Branches).(ID).properties.massflow	double	kg·s^{-1}
(Branches).(ID).properties.reynolds_number	double	-
(Branches).(ID).properties.velocity	double	m·s^{-1}
(Branches).(ID).properties.delta_p	double	Pa
(Branches).(ID).properties.friction_factor	double	kg^{-1}·m^{-1}
(Branches).(ID).solver		
(Branches).(ID).solver.equations.fluid_momentum.residual	double	N
(Branches).(ID).solver.equations.fluid_momentum.pressure.value	double	N
(Branches).(ID).solver.equations.fluid_momentum.friction.value	double	N
(Branches).system		
(Branches).system.names.total	cell	N/A
(Branches).system.names.fluid_branch	cell	N/A

Variable name	Type	Unit
(Branches).system.amount.total	integer	N/A
(Branches).system.amount.fluid_branch	integer	N/A

Table B-3: Summary of all available variables in the struct Settings. Settings contains the information required to solve the fluid system.

Variable name	Type	Unit
Settings.system_initialization.settings		
Settings.system_initialization.settings.file	string	N/A
Settings.system_initialization.modules		
Settings.system_initialization.modules.(module).type	string	N/A
Settings.system_initialization.modules.(module).folder	string	N/A
Settings.system_initialization.modules.(module).function	handle	N/A
Settings.system_initialization.modules.system.total	cell	N/A
Settings.system_initialization.modules.system.root	cell	N/A
Settings.system_initialization.modules.system.presolver	cell	N/A
Settings.system_initialization.modules.system.solver	cell	N/A
Settings.system_initialization.modules.system.in_solver	cell	N/A
Settings.system_initialization.modules.system.postsolver	cell	N/A
Settings.system_initialization.initialization		
Settings.system_initialization.initialization.system	cell	N/A
Settings.system_initialization.initialization.modules	cell	N/A
Settings.fluid_system_initialization		
Settings.fluid_system_initialization.method.nodes	cell	N/A
Settings.fluid_system_initialization.method.branches	cell	N/A
Settings.fluid_system_initialization.function.nodes	handle	N/A
Settings.fluid_system_initialization.function.branches	handle	N/A
Settings.solver.mode		
Settings.solver.mode	string	N/A
Settings.solver.iteration		
Settings.solver.iteration.min	double	-
Settings.solver.iteration.max	double	-
Settings.solver.system		
Settings.solver.system.equations	cell	N/A
Settings.solver.system.variables	cell	N/A
Settings.solver.system.neq	array	-
Settings.solver.system.internal_fluid_node	cell	N/A
Settings.solver.system.fluid_branch	cell	N/A
Settings.solver.equations		
Settings.solver.equations.(equation).method	string	N/A
Settings.solver.equations.(equation).relaxation	double	N/A
Settings.solver.equations.(equation).terms	cell	N/A
Settings.solver.equations.(equation).location	string	N/A
Settings.solver.equations.(equation).convergence	double	-
Settings.solver.equations.(equation).divergence	double	-
Settings.solver.equations.(equation).function	handle	N/A
Settings.solver.equations.(equation).system_type	string	N/A
Settings.solver.equations.(equation).term_functions	cell	N/A
Settings.solver.variables		
Settings.solver.variables.(variable).system_type	string	N/A
Settings.solver.variables.(variable).location	string	N/A

Variable name	**Type**	**Unit**
Settings.solver.methods		
Settings.solver.methods.(method).function	handle	N/A
Settings.solver.methods.(method).equations	cell	N/A
Settings.solver.methods.(method).neq	array	N/A
Settings.solver.methods.(method).system_type	cell	N/A
Settings.solver.methods.(method).variables	cell	N/A
Settings.solver.methods.(method).locations	cell	N/A
Settings.thermophysics.global_constants		
Settings.thermophysics.global_constants.g0	double	m·s^{-2}
Settings.thermophysics.global_constants.R	double	J·mol^{-1}·K^{-1}
Settings.thermophysics.global_constants.k	double	J·K^{-1}
Settings.thermophysics.global_constants.Na	double	mol^{-1}
Settings.thermophysics.global_constants.M	double	kg·mol^{-1}
Settings.thermophysics.global_constants.T_c	double	K
Settings.thermophysics.global_constants.Rho_c	double	mol·dm^{-3}
Settings.thermophysics.global_constants.P_c	double	Pa
Settings.thermophysics.global_constants.T_tp	double	K
Settings.thermophysics.global_constants.P_tp	double	Pa
Settings.thermophysics.global_constants.H_0	double	J·mol^{-1}
Settings.thermophysics.global_constants.S_0	double	J·mol^{-1}·K^{-1}
Settings.thermophysics.global_constants.T_0	double	K
Settings.thermophysics.(specie).models		
Settings.thermophysics.(specie).models.melting_pressure.type	string	N/A
Settings.thermophysics.(specie).models.melting_pressure.constants	matrix	N/A
Settings.thermophysics.(specie).models.melting_pressure.function	handle	N/A
Settings.thermophysics.(specie).models.vapor_pressure.type	string	N/A
Settings.thermophysics.(specie).models.vapor_pressure.constants	matrix	N/A
Settings.thermophysics.(specie).models.vapor_pressure.function	handle	N/A
Settings.thermophysics.(specie).models.saturated_liquid_density.type	string	N/A
Settings.thermophysics.(specie).models.saturated_liquid_density.constants	array	N/A
Settings.thermophysics.(specie).models.saturated_liquid_density.function	handle	N/A
Settings.thermophysics.(specie).models.equation_of_state.type	string	N/A
Settings.thermophysics.(specie).models.equation_of_state.constants.ideal	array	N/A
Settings.thermophysics.(specie).models.equation_of_state.constants.residual	array	N/A
Settings.thermophysics.(specie).models.equation_of_state.function	handle	N/A
Settings.thermophysics.(specie).models.surface_tension.type	string	N/A

Variable name	Type	Unit
Settings.thermophysics.(specie).models .surface_tension.constants	matrix	N/A
Settings.thermophysics.(specie).models .surface_tension.function	handle	N/A
Settings.thermophysics.(specie).models .viscosity.type	string	N/A
Settings.thermophysics.(specie).models .viscosity.constants.dilute	array	N/A
Settings.thermophysics.(specie).models .viscosity.constants.residual	array	N/A
Settings.thermophysics.(specie).models .viscosity.function	handle	N/A
Settings.thermophysics.(specie).models .thermal_conductivity.type	string	N/A
Settings.thermophysics.(specie).models .thermal_conductivity.constants.dilute	array	N/A
Settings.thermophysics.(specie).models .thermal_conductivity.constants.residual	array	N/A
Settings.thermophysics.(specie).models .thermal_conductivity.constants.olchowy	array	N/A
Settings.thermophysics.(specie).models .thermal_conductivity.function	handle	N/A
Settings.thermophysics.variables		
Settings.thermophysics.variables	cell	N/A

Appendix C

Derivatives of the Helmholtz Energy Function

The evaluation of the various thermodynamic state variables, as are shown in table 5-5, requires the calculation of the derivatives of the Helmholtz energy function. Numerical differentiation of the Helmholtz energy function requires multiple function evaluations which takes time. This problem is further increased when trying to iteratively find the inverse of the Helmholtz energy function using the methods described in appendix A. The process can be sped up significantly if a function for the derivative of the Helmholtz energy is available. In this appendix the generalized Helmholtz energy derivatives and the process how to find these derivatives will be described briefly.

C-1 Derivatives of the Residual Component of the Helmholtz Energy Function

The residual part of the Helmholtz energy function is generally given in the following form:

$$\alpha^r = \sum_{i=1}^{n} C_{1,i}\delta^{C_{2,i}}\tau^{C_{3,i}} \exp(-C_{4,i}\delta^{C_{5,i}}) \exp(-C_{6,i}(\delta - C_{7,i})^2 - C_{8,i}(\tau - C_{9,i})^2) \tag{C-1}$$

This can written in a general form as:

$$\alpha^r = \sum_{i=1}^{n} f_i(\delta, \tau) \tag{C-2}$$

Where $f_i(\delta, \tau)$ is equal to the bank of terms displayed in equation C-1. It can be seen that i only changes the values for the various constants used in equation C-1 while the functional form remains the same. What can also be observed is that the function $f_i(\delta, \tau)$ consists of the product of functions which are specified solely in terms of δ or τ (the last exponential function in equation C-1 can be split into a product of two exponential functions). Therefore $f_i(\delta, \tau)$ can be written as:

$$f_i(\delta,\tau) = f_{1,i}(\delta)\cdot f_{2,i}(\tau)\cdot f_{3,i}(\delta)\cdot f_{4,i}(\delta)\cdot f_{5,i}(\tau) \tag{C-3}$$

Where:

$$f_{1,i}(\delta) = C_{1,i}\delta^{C_{2,i}} \tag{C-4}$$
$$f_{2,i}(\tau) = \tau^{C_{3,i}} \tag{C-5}$$
$$f_{3,i}(\delta) = \exp(-C_{4,i}\delta^{C_{5,i}}) \tag{C-6}$$
$$f_{4,i}(\delta) = \exp(-C_{6,i}(\delta - C_{7,i})^2) \tag{C-7}$$
$$f_{5,i}(\tau) = \exp(-C_{8,i}(\tau - C_{9,i})^2) \tag{C-8}$$

For each term in equation C-4 the derivative can be written as a product of a derivative function and the function itself. For example the first term can be written as:

$$\left(\frac{df_{1,i}(\delta)}{d\delta}\right)_\tau = C_{1,i}C_{2,i}\delta^{C_{2,i}-1} = [C_{2,i}\delta^{-1}]\cdot[C_{1,i}\delta^{C_{2,i}}] = g_{1,i}(\delta)\cdot f_{1,i}(\delta) \tag{C-9}$$

Where $g_{1,i}$ is a derivative function which depends on the functional form of $f_{j,i}$. This means that when taking the partial derivative of the complete function f_i with respect to δ the following holds:

$$\left(\frac{\partial f_i(\delta,\tau)}{\partial\delta}\right)_\tau = g_i(\delta)\cdot f_i(\delta,\tau) \tag{C-10}$$

Where $g(\delta)$ is a separate function which is a sum of the derivative functions of the individual terms of $f_i(\delta,\tau)$ which follows from the product rule. The second partial derivative with respect to δ follows from the product rule applied to equation C-10:

$$\left(\frac{\partial^2 f_i(\delta,\tau)}{\partial\delta^2}\right)_\tau = \frac{\partial g_i(\delta)}{\partial\delta}\cdot f_i(\delta,\tau) + g_i(\delta)\cdot\frac{\partial f_i(\delta,\tau)}{\partial\delta} \tag{C-11}$$

Which can be simplified to the following form after substituting equation C-10:

$$\left(\frac{\partial^2 f_i(\delta,\tau)}{\partial\delta^2}\right)_\tau = \left[\frac{\partial g_i(\delta)}{\partial\delta} + g_i(\delta)\cdot g_i(\delta)\right]\cdot f_i(\delta,\tau) \tag{C-12}$$

In the same fashion the derivatives with respect to the inverse reduced temperature, τ can be written as:

$$\left(\frac{\partial f_i(\delta,\tau)}{\partial\tau}\right)_\delta = h_i(\tau)\cdot f_i(\delta,\tau) \tag{C-13}$$

and:

$$\left(\frac{\partial^2 f_i(\delta,\tau)}{\partial\tau^2}\right)_\delta = \left[\frac{\partial h_i(\tau)}{\partial\tau} + h_i(\tau)\cdot h_i(\tau)\right]\cdot f_i(\delta,\tau) \tag{C-14}$$

Finally, the partial derivative with respect to δ and τ can be written as:

$$\left(\frac{\partial^2 f_i(\delta,\tau)}{\partial\delta\partial\tau}\right) = g_i(\delta)\cdot h_i(\tau)\cdot f_i(\delta,\tau) \tag{C-15}$$

Working out the derivatives of each derivative function g_i, h_i, g_i' and h_i' gives the following general forms of the different partial derivatives of the residual part of the helmholtz function:

$$\begin{aligned}\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau =& \sum_{i=1}^{n}\left[C_{2,i}\delta^{-1} - C_{4,i}C_{5,i}\delta^{C_{5,i}-1} - 2C_{6,i}\left(\delta - C_{7,i}\right)\right]\cdot \\ & C_{1,i}\delta^{C_{2,i}}\tau^{C_{3,i}}\exp(-C_{4,i}\delta^{C_{5,i}})\exp(-C_{6,i}(\delta-C_{7,i})^2 - C_{8,i}(\tau-C_{9,i})^2)\end{aligned} \tag{C-16}$$

$$\begin{aligned}\left(\frac{\partial\alpha^r}{\partial\tau}\right)_\delta =& \sum_{i=1}^{n}\left[C_{3,i}\tau^{-1} - 2C_{8,i}(\tau - C_{9,i})\right]\cdot \\ & C_{1,i}\delta^{C_{2,i}}\tau^{C_{3,i}}\exp(-C_{4,i}\delta^{C_{5,i}})\exp(-C_{6,i}(\delta-C_{7,i})^2 - C_{8,i}(\tau-C_{9,i})^2)\end{aligned} \tag{C-17}$$

$$\begin{aligned}\left(\frac{\partial^2\alpha^r}{\partial\delta^2}\right)_\tau =& \sum_{i=1}^{n}\left[-C_{2,i}\delta^{-2} - C_{4,i}C_{5,i}(C_{5,i}-1)\delta^{C_{5,i}-2} - 2C_{6,i}\right] + \\ & \left[C_{2,i}\delta^{-1} - C_{4,i}C_{5,i}\delta^{C_{5,i}-1} - 2C_{6,i}\left(\delta - C_{7,i}\right)\right]^2\cdot \\ & C_{1,i}\delta^{C_{2,i}}\tau^{C_{3,i}}\exp(-C_{4,i}\delta^{C_{5,i}})\exp(-C_{6,i}(\delta-C_{7,i})^2 - C_{8,i}(\tau-C_{9,i})^2)\end{aligned} \tag{C-18}$$

$$\begin{aligned}\left(\frac{\partial^2\alpha^r}{\partial\tau^2}\right)_\delta =& \sum_{i=1}^{n}\left[\left[-C_{3,i}\tau^{-2} - 2C_{8,i}\right] + \left[C_{3,i}\tau^{-1} - 2C_{8,i}(\tau-C_{9,i})\right]^2\right]\cdot \\ & C_{1,i}\delta^{C_{2,i}}\tau^{C_{3,i}}\exp(-C_{4,i}\delta^{C_{5,i}})\exp(-C_{6,i}(\delta-C_{7,i})^2 - C_{8,i}(\tau-C_{9,i})^2)\end{aligned} \tag{C-19}$$

$$\begin{aligned}\left(\frac{\partial^2\alpha^r}{\partial\delta\partial\tau}\right) =& \sum_{i=1}^{n}\left[C_{2,i}\delta^{-1} - C_{4,i}C_{5,i}\delta^{C_{5,i}-1} - 2C_{6,i}\left(\delta - C_{7,i}\right)\right]\cdot\left[C_{3,i}\tau^{-1} - 2C_{8,i}(\tau-C_{9,i})\right]\cdot \\ & C_{1,i}\delta^{C_{2,i}}\tau^{C_{3,i}}\exp(-C_{4,i}\delta^{C_{5,i}})\exp(-C_{6,i}(\delta-C_{7,i})^2 - C_{8,i}(\tau-C_{9,i})^2)\end{aligned} \tag{C-20}$$

C-2 Derivatives of the Ideal Component of the Helmholtz Energy Function

The ideal component of the Helmholtz energy function is generally given in the following form:

$$\alpha^0 = \ln(\delta) + \sum_{i=1}^{n} C_{1,i}\tau^{C_{2,i}}\ln(\tau)^{C_{3,i}}\ln(1-\exp(-C_{4,i}\tau))^{C_{5,i}} \tag{C-21}$$

Similarly to the residual Helmholtz function, the Ideal Helmholtz function is constructed out of a product of terms which can be written solely as function of either δ or τ. This means that the same method could be implemented as was used in the previous section, where the derivative can be written as a product of the derivative function and the function itself. However, in this case some derivatives in the derivative function will contain terms of the form $ln(C)$, where C can equal zero, which makes the result invalid. It is therefore difficult to create a generalized form of the derivatives of the ideal Helmholtz function.

Another procedure is therefore implemented. This procedure makes use of the assumption that each function, i, in equation C-21 will only contain one term. That means that if it is known which terms are switched on or off (where the constants, $C_{2,i}, C_{3,i} and C_{5,i}$ equal 1 or zero), the derivatives can be grouped per term. Taking the partial derivatives for each term gives the following set of partial derivatives:

- If $C_{2,i} = 1$ and $C_{3,i} = 0$ and $C_{5,i} = 0$

$$\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta^k = \sum_{i=1}^{k} C_{1,i} \tag{C-22}$$

$$\left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta^k = \sum_{i=1}^{k} 0 \tag{C-23}$$

- If $C_{2,i} = 0$ and $C_{3,i} = 1$ and $C_{5,i} = 0$

$$\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta^l = \sum_{i=1}^{l} C_{1,i}\tau^{-1} \tag{C-24}$$

$$\left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta^l = \sum_{i=1}^{l} -C_{1,i}\tau^{-2} \tag{C-25}$$

- If $C_{2,i} = 0$ and $C_{3,i} = 0$ and $C_{5,i} = 1$

$$\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta^m = \sum_{i=1}^{m} \frac{C_{1,i}C_{4,i}}{\exp(C_{4,i}\tau) - 1} \tag{C-26}$$

$$\left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta^m = \sum_{i=1}^{m} -\frac{C_{1,i}C_{4,i}^2\exp(C_{4,i}\tau)}{(\exp(C_{4,i}\tau) - 1)^2} \tag{C-27}$$

The final derivatives can thus be written as:

$$\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta = \left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta^k + \left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta^l + \left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta^m \tag{C-28}$$

$$\left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta = \left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta^k + \left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta^l + \left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta^m \tag{C-29}$$