



University of
Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Master in Construction and Material / Energy	Spring semester, 2012 <u>Open</u> / Restricted access
Writer: Eli-Marie W. Sundt (Writer's signature)
Faculty supervisor: Bjørn H. Hjertager External supervisor(s): Jørgen Osenbroch	
Title of thesis: Numerical and Analytical Study of Steady State and Transient Heat Transfer in Liquid Filled Dead legs.	
Credits (ECTS): 30	
Key words: Dead leg, Heat transfer, Steady State/Transient, CFD – simulation, OpenFOAM, Standards.	Pages: 105 (79) + enclosure: 58 + CD Stavanger, 11/06/2012 Date/year

NUMERICAL AND ANALYTICAL STUDY OF STEADY STATE AND TRANSIENT HEAT TRANSFER IN LIQUID FILLED DEADLEGS

Eli-Marie W. Sundt

30 ECTS thesis submitted as part of fulfillment of a
Master degree in Mechanical and Structural Engineering and Material Science

Supervisors

Bjørn H. Hjertager

Jørgen Osenbroch

Faculty of Science and Technology

University of Stavanger

Stavanger, 06 2012

Numerical and Analytical Study of Steady State and Transient Heat Transfer in Liquid Filled Deadlegs

Heat transfer in deadleg

30 ECTS thesis submitted as part of fulfilment of a M.Sc. degree in Mechanical and Structural Engineering and Material Science

Copyright © 2012 Eli-Marie W. Sundt
All rights reserved

Faculty of Science and Technology
University of Stavanger

Ullandhaug
4041, Stavanger
Norway

Bibliographic information:

Eli-Marie W. Sundt, 2012, Numerical and Analytical Study of Steady State and Transient Heat Transfer in Liquid Filled Deadlegs, M.Sc. thesis, Faculty of Science and Technology, University of Stavanger.

Printing: Stavanger
Stavanger, Norway, 06 2012

Abstract

A dead leg is an inactive part of a process pipe, and it has been the cause of serious incidents due to liquid reaching its threshold value (TV) for freezing. The existing industrial approach for the design of dead legs is based on rules of thumb and reference standards. Accordingly, a detailed understanding of the temperature development is not available. An objective of this thesis is therefore to develop a Computational Fluid Dynamic (CFD) model in OpenFOAM that can analyse such cases in a more comprehensive matter.

This study is based on investigations of the existing theory and industrial approach. This is further used to develop a realistic CFD simulation. The results are obtained from two dead leg geometries. One geometry presents the dead leg as a Fin, while the other geometry presents the dead leg connected to a main pipe, creating a T-junction. The dead leg dimensions are based on rules of thumb, namely a diameter of 2" and a length of 0.5m. This gives a length to diameter ratio (L/D) of approximately 9.

The results are obtained from three cases. Where the Fin geometry is related to Case 01 and the T-junction is related to Case 02 and Case 03. Case 01 and Case 03, are purely related to heat transfer, both steady-state and transient. The results of these two cases differs with approximately 10%, indicating that the fin analysis may be sufficient when the ALARP principle is taken into account. In addition, an analysis is performed to investigate the effect of wind, which is observed to have a significant effect on the time for water to reach TV. Case 02 introduces a flow field and turbulence in the T-junction. It is observed that for a range of velocities, $u = 1 - 2.86\text{m/s}$, the circulation in the dead leg reach $L/D_i = 4 - 6$ for the respective velocities. This results in a stagnant fluid in the end of the dead leg as the circulation cease to exist. This effect is reflected in the temperature development, the result implies almost no heat loss until $L/D_i = 4.5 - 6.8$ where the temperature falls rapidly. Consequently, the effect of circulation implies that a design criterion may be established from further studies. Verification of the CFD-models has been performed by the use of grid independence test, existing theory and previous experimental results by Habib et al.

Contents

List of Figures	xi
List of Tables	xv
Nomenclature	xvii
Preface	xxi
1. Introduction	1
1.1. Problem definition	3
1.2. Background and Motivation	5
1.3. Objectives	5
1.4. Thesis Layout	6
2. Heat transfer theory	7
2.1. General heat transfer theory	7
2.2. Heat transfer equation	8
2.3. External Flow region	9
2.4. Internal Flow Region	11
2.5. Pipe and Fluid Properties	12
3. Turbulence modelling	15
3.1. Governing Equation in Turbulent pipe flow	15
3.2. The k - ϵ model equations	16
4. Computational Fluid Dynamics	19
4.1. Finite Volume Method	19
4.2. OpenFOAM	20
4.3. Numerical schemes	22
4.4. Solution algorithms	24
4.4.1. Linear solver control	24
4.4.2. Tolerance and residuals	24
4.4.3. Relaxation Factors	24
4.4.4. Semi-Implicit method for Pressure-Linked Equation (SIMPLE) algorithm	25

4.5. OpenFOAM Solver	25
4.5.1. LaplacianFoam	26
4.5.2. PotentialFoam	27
4.5.3. BuoyantBoussinesqSimpleFoam	27
4.6. Sampling	29
5. Industrial approach	31
5.1. NORSOK P-001 & L-002	31
5.2. BS EN ISO 12241:2008	32
5.2.1. Procedure	32
5.3. Operating procedures for offshore installations	35
6. Analytical Solution	37
6.1. Fin-analysis, mathematical formulation	37
7. Simulation set-up	41
7.1. Development of the geometry	41
7.2. Boundary Conditions	42
7.2.1. Inlet boundary conditions	44
7.2.2. Outlet boundary conditions	45
7.2.3. Wall boundary conditions	45
7.3. External environment	47
7.4. Numerical Accuracy	47
7.5. Case description	53
8. Result and Discussion	55
8.1. Case 01.a – Fin-analysis, wind velocity 9m/s	55
8.1.1. Case 01.b – Fin analysis, wind velocity 3m/s	59
8.2. Effect of wind velocity	61
8.3. Verification and discussion of Case 01	62
8.4. Case 02 – Normal working conditions	64
8.4.1. Case 02.a – Develop a potential flow field	64
8.4.2. Case 02.b - Heat transfer&flow analysis	65
8.5. Verification and discussions of Case 02	68
8.6. Case 03 – Shut down after normal working conditions	70
8.7. Industrial vs. realistic approach	72
9. Conclusion	75
9.1. Further work	76
Bibliography	77
Appendices	83
A. Project Plan	83

B. Initial Conditions	87
C. Governing Equations	91
C.1. Vector notation and Indicjal notation	91
C.2. Equation from the laws of conservation	92
C.3. Turbulence equation	92
D. Flow chart SIMPLE-algorithm	95
E. Flow regime	97
F. Matlab calculations	99
G. Solver Library	113
G.1. LaplacianFoam	113
G.2. PotentialFoam	114
G.3. BuoyantBoussinesqSimpleFoam	117
G.3.1. Velocity equation	119
G.3.2. Temperature equation	119
G.3.3. Pressure equation	120
G.3.4. Transport Properties	121
G.3.5. Turbulence Library	121
H. Procedure to run the case files in OpenFOAM	139
I. Case Files	141

List of Figures

1.1. <i>Process Flow Diagram of a general produced water system.</i>	3
1.2. <i>Sketch of base case.</i>	4
2.1. <i>Cylinder containing hot fluid flow exposed to convective surface conditions [Incropera and DeWitt, 2005]</i>	9
2.2. <i>Boundary layer formation on a cylindrical cross section exposed to cross flow. [Incropera and DeWitt, 2005]</i>	10
2.3. <i>General presentation of the Bernoulli principle of liquid column.</i>	12
3.1. <i>Control volume indicating the first data-point from the wall [Mme, 2010]</i>	18
4.1. <i>Overview of the environments in OpenFOAM [Foundation, 2011]</i>	20
4.2. <i>The file structure in the directories in OpenFOAM [Foundation, 2011]</i>	21
4.3. <i>Probes located in the dead leg.</i>	30
5.1. <i>Temperature distribution in a hollow cylinder [ISO 12241, 2008]</i>	33
6.1. <i>Layout of Fin-geometry</i>	37
7.1. <i>3D caption of dead leg layout</i>	42
7.2. <i>Unstructured triangular mesh - Netgen 1D-2D-3D</i>	50
7.3. <i>Comparison of final temperature profile for mesh 1-4</i>	51

7.4.	<i>Comparison of final velocity distribution for mesh 1-4</i>	51
7.5.	<i>Comparison of final turbulent kinetic energy distribution for mesh 1-4</i>	51
7.6.	<i>Plot of the wall coordinate, y^+</i>	52
7.7.	<i>Residuals of Mesh 3, $N \approx 280\,000$</i>	52
8.1.	<i>Contour of the steady state Fin-analysis</i>	55
8.2.	<i>Initial and final plot of the temperature distribution in Case 01a</i>	56
8.3.	<i>First sign of water reaching TV with wind velocity 9m/s</i>	57
8.4.	<i>Probes measurements for Case 01.a</i>	58
8.5.	<i>Temperature contour when reaching TV at center line at $\approx 4000s$</i>	58
8.6.	<i>Temperature profile with wind velocity 3m/s after 2800 s</i>	59
8.7.	<i>Temperature profile with wind velocity 3m/s after 5000s</i>	60
8.8.	<i>Probe measurements for Case 01.b</i>	61
8.9.	<i>Effect of wind velocity in the center of dead leg after ≈ 1 h</i>	61
8.10.	<i>Effect of wind velocity at the wall of dead leg after ≈ 1 h</i>	62
8.11.	<i>Probe measurements at $5D_i$ for case 01a and 01b</i>	62
8.12.	<i>Comparison of the analytical solution, numerical and the Laplace simulation</i>	63
8.13.	<i>Schematic explanation of the interface boundary conditions</i>	64
8.14.	<i>Contour of the velocity field from potential flow theory</i>	64
8.15.	<i>Velocity profile developed from potential flow theory</i>	65
8.16.	<i>Contour of the temperature distribution for the converged solution</i>	65
8.17.	<i>The temperature distribution at wall and center line of dead leg</i>	66
8.18.	<i>The velocity field in the T-junction for the final converged state</i>	67

8.19. <i>The circulation presented by velocity vectors</i>	68
8.20. <i>Velocity magnitude of circulation in the dead leg</i>	68
8.21. <i>The vertical velocity in the dead leg for each diameter segment</i>	69
8.22. <i>Schematic presentation of the temperature through the dead leg when the internal velocity is 1m/s</i>	70
8.23. <i>Measurement taken at probes located in the main pipe</i>	70
8.24. <i>Measurement taken at probes located in the dead leg</i>	71
8.25. <i>Temperature contour 2800s after shut down</i>	72
8.26. <i>Temperature distribution in the dead leg 2800s after shut down</i>	72
8.27. <i>Measurement of the temperature over time for $5D_i$, $7D_i$ and end</i>	74
A.1. <i>Gant chart of the planned activities and time schedule</i>	86
D.1. <i>Flow chart for the SIMPLE algorithm [Versteeg and Malalasekera, 2007].</i>	96
E.1. <i>Boundary layer formation on a cylindrical cross section exposed to cross flow [Sumer and Fredsøe, 2006].</i>	98

List of Tables

- 2.1. Pipe Schedule 13
- 2.2. Properties of the applicable substances 13

- 4.1. Numerical Schemes in OpenFOAM 22
- 4.2. Equation discretization in OpenFOAM 26
- 4.3. Diameter segments in the vertical deadleg, z-axis 29

- 7.1. Pipe size and wall thickness 41
- 7.2. Defined boundaries on the 3D models 42
- 7.3. Defined boundaries in the Fin-analysis 43
- 7.4. Defined boundaries in the T-junction 43
- 7.5. Defined boundaries for the turbulence variables 44
- 7.6. Environmental conditions 47
- 7.7. Number of cells (N) 48
- 7.8. Simulated Cases 53

- B.1. Initial conditions for velocity 87
- B.2. Initial conditions for pressure, p 87
- B.3. Initial conditions for turbulent kinetic energy, k 88

B.4. Initial conditions for turbulent dissipation, ε	88
B.5. Initial conditions for v_t	89
B.6. Initial conditions for α_t	89
B.7. Initial conditions for Temperature, T	90
B.8. Variables for GroovyBC	90

Nomenclature

Latin letters

T	Temperature [K/ °C]
k	Conductivity [W/mK]
c_p	Specific heat capacity [J/kgK]
R	Linear thermal resistance [mK/W]
h	Local heat coefficient [W/m ² K]
h_{tot}	Total heat transfer coefficient [W/m ² K]
D	Diameter [m]
A	Area [m ²]
L	Length [m]
u	Velocity [m/s]
r	Radius [m]
g	Gravity [m/s ²]
t	Time [s]
x	x-direction

z z-direction

m Mass [kg]

P Perimeter [m]

Greek letters

μ Dynamic Viscosity [Pa·s]

ν Kinematic Viscosity [m²/s]

ρ Density [kg/m³]

α Diffusion [m²/s]

ε Dissipation [W/kg= m²/s³]

k Turbulent kinetic energy [J/kg= m²/s²]

δ The Kronecker delta function

ℓ Turbulent mixing length [m]

Φ Heat flow rate [W]

Subscripts and superscripts

e External

a Ambient

i Internal

M Main pipe

D Dead leg

s Surface
 ∞ Infinite
eff Effective
t Turbulent

Abbreviations and dimensionless numbers

NS Navier–Stokes
MMO Maintenance, Modification and Operation
CFD Computational Fluid Dynamics
ISO Isometric
FOAM Field, Operation And Manipulations
DN Nominal Diameter
SCH Schedule
WT Wall Thickness
FVM Finite Volume Method
RANS Reynolds Average Navier–Stokes
LES Large Eddy Simulations
Pr Prandtl Number
Nu Nusselt number
Re Reynolds number

Ra Rayleigh number

SS Stainless Steel

ALARP As Low As Reasonably Practicable

TV Threshold Value

Preface

This master thesis is the final work in the master program at the University of Stavanger. The student is attending a master program in Construction and Mechanical engineering and Material science, with a specialization in Energy technology. The master thesis is an individual, scientific report. It is expected that the student gain experience in how to plan and work towards a defined problem solution within a time limit. This master thesis is a collaboration with the Aker Solutions, and incorporates both experimental and analytical work. By experimental work it refers to the CFD-simulations.

Working with this thesis has been both interesting and challenging. I have utilized the knowledge obtained during my five years of studies. In addition, new knowledge has been obtained in regards to the CFD programming tool, OpenFOAM.

I would like to use this opportunity to express a great appreciation to Mr. Bjørn H. Hjertager who served as my supervisor at the University. I would like to thank him for all the time he invested in good discussions and guidance, as well as for his patience to answer all my questions. For the CFD part I would like to thank both Mr. Bjørn H. Hjertager and Mr. Knut Erik Giljarhus for sharing their knowledge and experience. I would also thank Mr. Knut Erik Giljarhus for his response to my work. In addition I would like to thank the operation team at Norske Shell AS for contributing to a constructive discussions of their concerns regarding the subject.

I would also like to thank Mr. Jørgen Osenbroch, Aker Solutions, for making this project available, providing the project background and making a previous study available. Also Aker Solutions and the library service deserve a thank for the facilities provided me during this time.

Finally, I would like to thank my family and friends for their support during the thesis, and for the preceding five years of studies.

Stavanger, June 12, 2012

Eli-Marie Sundt

1. Introduction

This thesis is a collaboration with Aker Solutions which is one of the leading engineering companies in the oil and gas industry. The topic concerns the hazards of freezing of liquid in pipe branches, hereby called dead legs. The term dead leg describes an inactive portion of a pipe [Habib et al., 2005c], which is ended by a closed end. The dead leg may be installed as part of by-pass lines, future tie-ins, a demolished part, instrument connection, sampling points and so on. As of today the installation of dead legs is necessary to provide sufficient flexibility of the process. The approach to decide upon a design is done by simple rules of thumb which are developed from previous experience and reference standards.

From an extensive literature search it is found that there are some research reports published which investigating dead legs and closed ducts. Among the published research the focus is mainly on the effect of dead leg geometry, the flow field and circulation, the pressure drop and the subsea dead leg heat transfer. There is not many studies regarding the issue of heat transfer in topside dead legs.

Hong [1977] investigated the natural circulation in horizontal pipes. Hong addressed the problem of natural circulation which occurred when a horizontal pipe had one closed end and one end connected to another pipe where hot fluid was passing. Hong assumed that due to the small tube wall thickness the temperature variation in the radial direction across the wall could be ignored. He also found that there were two bulk temperatures in the tube, one for the hot and one for the cold fluid. From these observations, Hong discovered that the energy equation became the same as the differential equation for the Fin-analysis.

Said, Badr, Hussaini and Habib studied the effect of geometry in vertical dead legs in respect to the flow field and the oil/water separation [Habib et al., 2005c]. They performed a CFD analysis on a vertical dead leg and observed the velocity field for an oil and water mixture of 10% water and 90% oil. Studying different length/diameter (L/D) ratios at a constant velocity of 1m/s. They found that the upper section of the dead leg was characterized by a circulating flow zone up to approximately $3D$ from the inlet of the dead leg. Their study also implied that the middle zone in the dead leg was characterized by several counter-rotating vortices, while the lower part of the dead leg was occupied by stagnant flow. In addition Habib et al. [2005b] conducted a similar study on the characteristic flow field and water concentration in a horizontal dead leg. In this study they performed flow

visualization experiments to verify their calculation procedure. They investigated the flow field for different dead leg geometries and different inlet flow velocities. From the results they concluded that the flow in the horizontal dead legs experienced no circulation beyond $3 - 5D$. The two latter studies were then published in a third study [Habib et al., 2005a] to present the development of a dead leg criterion. The general criterion stated that for the vertical dead leg the circulation ceased to exist for $L > 3D$ while for the horizontal dead leg, the circulation ceased to exist for $L > 3 - 5D$. Also, they discovered that the water concentration increased in the stagnant area of the dead leg, which indicated that a larger L/D increased the water concentration.

Edwards and Catton [1969] studied the effect of closed cylinders heated from below. They investigated vertical pipes which were closed by a hot plate on the bottom and a cold plate on the top. The natural convection was observed as the heated fluid drifted upwards due to the change in density and the cooled fluid fell downwards due to the effect of gravitation. Their study concluded that to discover the true values of Rayleigh number (Ra) for large values of conductivity, k . Extremely careful investigations had to be performed. Their study only investigated the convective part, neglecting the effect of heat conductivity, thus, they implied that this had to be taken into consideration when analysing their results.

Mme [2010] studied how subsea dead legs were affected by cold spots due to lack of possibilities to insulate. The occurrence of convection between the hot and cold fluid was studied, and both experimental and computational studies were performed. The results concluded that the heat transfer from a localized cold spot was more efficient when subjected to a horizontal pipe, and as the pipe was more and more inclined the flow became more unstable and the heat transfer got reduced. He tested both the Reynolds Average Navier–Stokes (RANS) and Large Eddy Simulation (LES) models to compare the results with the experimental measurements. His conclusion stated that the LES-model provided a better agreement than the RANS-model.

Andersen [2007] investigated the formation of hydrates in subsea dead legs, by performing a Computational Fluid Dynamics (CFD) analysis in Fluent, ANSYS. The paper investigated the formation of hydrates in a hydrocarbon production system, and the results are presented for temperature and velocity. Andersen's results implied that the hydrate formation and heat loss was strongly dependent on the heat transfer coefficient, and that a temperature difference induced small fluid velocities in the dead leg.

An investigation report by the Investigation board US [2007] presents the results of an incident which happened due to a dead leg. The investigation report [Investigation board US, 2007] describes the dangers of dead legs. Poor maintenance and lack of freezing protection resulted in a fire which came out of control. The reason for the incident was that the process got redirected to another piping route, creating a dead leg. As a consequence the freezing protection in this area was no longer under the same investigation routine. Therefore, when the winter approached and the liquid started to freeze it resulted in a fracture. The fracture was maintained during the sufficiently cold winter days. However,

1.1. PROBLEM DEFINITION

as the summer came the ice melted and the fraction in the pipe resulted in leakage which under high pressure resulted in an explosion. Kaszniak [2010] investigated the Valero incident report [Investigation board US, 2007] and several other LPG incidents. Kaszniak discusses the results of the report which has been established for the incidents. Further, he propose several recommendations for handling Process Hazards. One is to avoid dead legs when ever possible.

1.1. Problem definition

The problem under consideration is the temperature profile in a dead leg exposed to a harsh external environment. Today the approach is based upon reference documents and previous experience. There is no detailed knowledge on how the temperature distribution is behaving in the dead leg. Therefore, it is desirable to obtain a computational model based on empirical data.

This thesis addresses dead legs in the produced water system on the Draugen field in the North Sea. In Figure 1.1 it can be observed that there are many types of dead legs. Those marked with red are constant dead legs, while those marked with blue may become dead legs. The internal flow cycle of the produced water system goes through hydro cyclones to

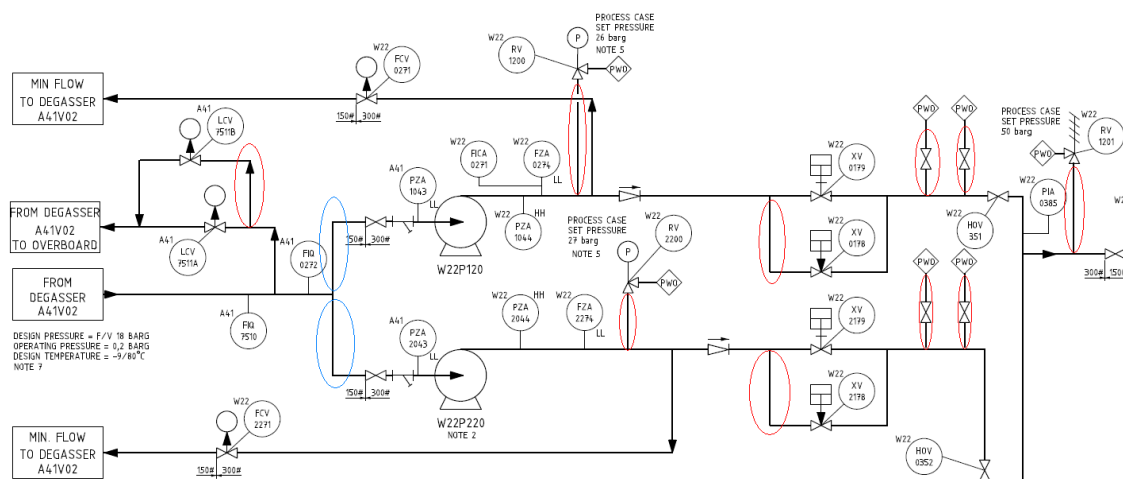


Figure 1.1: Process Flow Diagram of a general produced water system.

degassing drum before the produced water can be used in the water injection system. The produced water system is located on low points which are where dead legs are the most critical. Also, the produced water system is regularly shut down to perform maintenance and water wash of the hydro cyclones and drums. The shut down period depends on the criteria of the operational design. However, from discussion with the operation team at Shell AS [2012], it is implied that the shut down generally is performed once a month for

approximately 24 hours. When the system is shut down, the water temperature will start to decrease.

Under normal operating conditions the main pipe acts as a constant heat source at the inlet of the dead leg and as long as the fluid is continuously flowing over the dead leg inlet there will be a circulation as Habib et al. [2005c] states in his report. Consequently, there will be some exchange of fluid which means that the hazard of freezing will depend on the flow field.

To be able to approach the problem some simplifications and assumptions are made to create a simple but realistic picture of the heat transfer. The industry normally assumes that the dead leg can be regarded as a fin, and then conduct a fin analysis to find the temperature profile. However, as the literature search has brought to light the report by Habib et al. [2005a], which investigate the flow effect in the dead leg, this challenges the existing approach.

Furthermore, the development of this model may ease the investigations of different operating cases. The effect of wall thickness, length scales and different kinds of fluids can be investigated by doing small changes in the model. Therefore, develop a simple model should be a goal in itself. In fact, the idea behind this thesis is to develop a model that may be used by engineers subjected to these kinds of issues in daily life.

Dead legs may occur in many different shapes and sizes as shown in Figure 1.1. The rule of thumb says that a dead leg of $L < 0.5\text{m}$ and $D_e = 2''$ does not need insulation [Osenbroch, 2012]. Hence, this will be used as the base case for this thesis. The T-junction is illustrated in Figure 1.2. It shows how it is expected that the velocity and temperature will evolve in the main pipe. Also the circulation of fluid in the dead leg is presented.

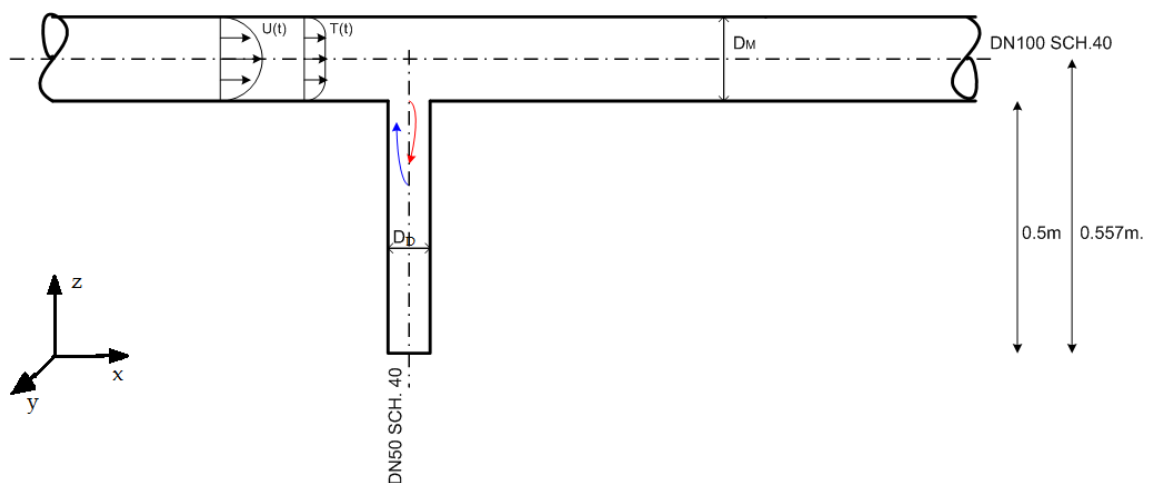


Figure 1.2: Sketch of base case.

1.2. Background and Motivation

The top-side process system is structured from pipes which stretches over the entire platform and is exposed to harsh weather conditions. If any formation of ice-plugs in a produced water line or wax in an oil line should occur, it may lead to production stop, damage of equipment, decrease of the system integrity or personnel injuries [Investigation board US, 2007]. Dead legs are important to enable the possibility to develop the system in the future, to have sampling and measurement points as well as drains.

The issue of formation of ice-plugs and wax occur as a consequence of the temperature going below the liquids threshold value (TV) for freezing and wax formation. For dead legs this is especially critical as the fluid in some parts is stagnant. However, under normal working conditions there will be a constant flow in the main pipe and the dead leg inlet will experience a circulation [Habib et al., 2005c] that may induce some temperature exchange in the fluid.

Today's approach, when designing dead legs and pipes, is to use standard reference documents. For the offshore industry in Norway, standards such as NORSOK, ASME and ISO are used. For the given problem, relevant procedures are NORSOK P-001 [2006], NORSOK L-002 [2009] and ISO 12241 [2008].

The topic of dead legs often creates discussions between the operation personnel and the operating company [Shell AS, 2012]. The operation personnel are concerned when the temperature is around design temperature and the wind velocity is high. A normal procedure is then to either install a wind shield to protect the pipes and equipment that are exposed, or perform a shut down if there is a danger of the fluid reaching TV. A CFD study of the temperature development may result in an increased safety for the personnel and a reduction of cost from shut down. Another issue is the subject of insulation. Insulation is a costly business for the operating companies, and it also decreases the lifetime of the pipe due to corrosion issues. A CFD study may provide a recommendation for how to increase the integrity of the system under different scenarios which may occur, and by incorporate effects such as wind and internal flow, the existing industrial approach can be questioned or justified by comparing the results obtained.

1.3. Objectives

This thesis aims to develop a numerical model which will show how the fluid temperature evolves in the dead leg. The aim is to use the results obtained from the simulation to better understand how to manage the dead leg and the danger of reaching the threshold value. The objectives are,

- Examine existing scientific literature regarding the subject.
- Establish a CFD model in OpenFOAM.
- Investigate today's industrial approach, simulate the same approach in OpenFOAM and compare the results with an existing empirical model developed in Aker Solutions.
- Create a more realistic and complex simulation introducing both flow, turbulence and heat transfer to see how the temperature evolves when the process is working under normal conditions.
- Evaluate the temperature loss that occurs when the system is shut down after running under normal conditions. Give a recommendation on how to operate the system to avoid the temperature going below threshold value (TV).
- Compare the realistic approach with the procedure from reference documents.
- Give a recommendation on the operating procedure to avoid the critical scenario of liquid going below its threshold value.
- Include a simple procedure to run the developed CFD-model.

1.4. Thesis Layout

The thesis is presented in nine chapters. The first chapter introduces the problem and existing literature, motivation and objectives. Chapter two presents the general heat transfer theory and the most important properties to establish the needed coefficients. Chapter three gives an overview of turbulence modelling. Chapter four is a presentation of Computational Fluid Dynamics (CFD) and OpenFOAM. It presents the basics of how the structure of OpenFOAM is. Chapter five presents the industrial approach retrieved from reference documents. Chapter six presents the analytical solution of the fin analysis. Chapter seven includes the simulation set up with details of geometry development, boundary conditions, numerical accuracy and case description. Finally, chapter eight presents the result and discussion and chapter nine presents the conclusion and further work.

2. Heat transfer theory

A heat transfer study may in general be divided into two groups, steady state models and transient models . A steady state model has a solution that only varies with the spatial coordinates but not with time. This makes the equations simpler as the time-derivative terms in the energy equations are equal to zero. Transient models are used to investigate the time dependency of heat transfer problems [LeVeque, 2007].

2.1. General heat transfer theory

In general, the solution of a heat transfer problem depends upon the thermal properties of the system, and any thermal loads which act upon it.

There are two main properties, namely heat conduction and heat convection. A brief explanation and formulation is described in this section and it is mostly retrieved from Cengel et al. [2001] and Incropera and DeWitt [2005].

Heat transfer always involves conduction, which is the transfer of energy from an area of high energy to an area of low energy. It can occur in solids, liquids or gases. The conduction is due to collision of molecules. The rate of heat conduction through a medium depends strongly on the geometry, i.e. thickness, material properties and the temperature difference between the two surfaces.

Steady state heat conduction is the definition of Fourier's law of heat conduction,

$$\dot{Q}_{cond} = -kA \frac{dT}{dx} \quad (2.1)$$

where k is the thermal conductivity, namely a measure of the ability a substance have to conduct heat, A is the cross sectional area of the object and $\frac{dT}{dx}$ is the temperature gradient.

In the offshore industry the most commonly used material are alloys such as carbon steel or stainless steel, which has a thermal conductivity, k , between 10-60 [W/mK]. The conductivity of a substance is found from property tables online or in heat transfer and fluid mechanics books. It is important to note that the conductivity depends on the temperature,

but is often assumed to be constant.

The mechanism of heat conduction in a liquid is more complicated as the molecules in liquids are closer together. As the temperature increase the conductivity of the liquid increases as the internal bonds of the molecules are released. This is called diffusion. The diffusion through a substance is described by another thermal property, namely the heat capacity, $h_f = \rho c_p$, where ρ is density of the liquid and c_p is the specific heat capacity. The heat capacity describes how much heat that can be stored in a substance. The diffusion process, on the other hand, is an indication of how fast the heat diffuses through the material and it only appears in the transient heat conduction, expressed from equation (2.2),

$$\alpha = \frac{k}{\rho c_p} \quad [\text{m}^2/\text{s}] \quad (2.2)$$

As the thermal diffusivity increases, the propagation of heat into the medium becomes more rapid [Cengel et al., 2001]. Water is calculated from equation (2.2) and has a typical thermal diffusivity of,

$$\alpha = 0.15 \times 10^{-6} \text{m}^2/\text{s}$$

The heat convection is the transport of heat due to fluid motion. The faster the fluid motion is, the greater the convection of heat transfer. There are two types of convection, forced- and natural convection. Heat transfer processes that involve change of phase of a fluid are also considered to be convection. Despite the complexity of convection, the rate of heat transfer is observed to be proportional to the temperature difference. This is expressed by Newton's law of cooling,

$$\dot{Q}_{conv} = hA_s(T_s - T_{inf}) \quad [\text{W}] \quad (2.3)$$

where h is the heat transfer coefficient [$\text{W}/\text{m}^2\text{K}$], T_s is the surface temperature [K], $T_{inf} = T_a = T_e$ is the temperature far from the surface [K] and A_s is the surface area of which the flow is exposed to.

2.2. Heat transfer equation

This section is retrieved from Incropera and DeWitt [2005] and it present the general heat transfer equations for a cylinder. As the pipe is exposed to an internal hot fluid flow and a cold cross flow of air on the outside. The heat equation for a steady state heat transfer with no heat generation becomes,

$$\frac{1}{r} \frac{d}{dr} \left(kr \frac{dT}{dr} \right) = 0 \quad (2.4)$$

When integrating equation(2.4) two constants appear, and the ODE is solved by implementing the boundary conditions for temperature at the two surfaces. The thermal resistance is then expressed by combining the temperature distribution with Fourier's law,

2.3. EXTERNAL FLOW REGION

equation (2.1), resulting in equation (2.5),

$$R_{cond} = \frac{\ln(r_2/r_1)}{2\pi Lk} \quad (2.5)$$

where L is the length, k is the thermal conductivity, r_1 and r_2 is the internal and external diameter respectively, Figure 2.1 illustrate the parameters.

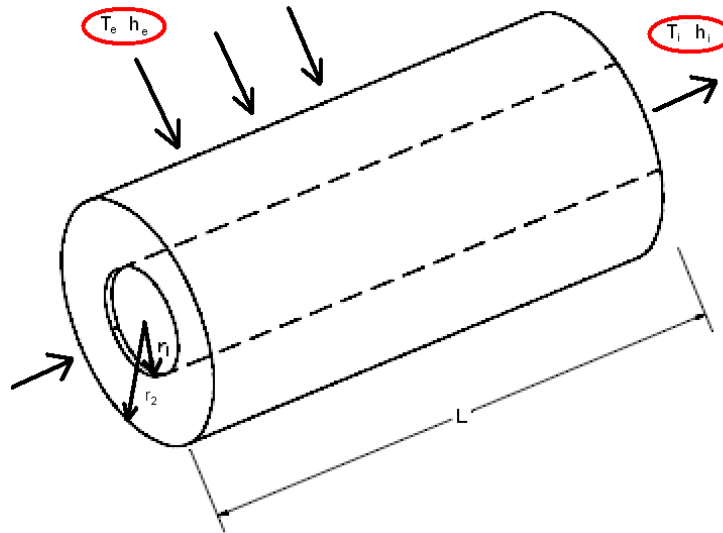


Figure 2.1: Cylinder containing hot fluid flow exposed to convective surface conditions [Incropera and DeWitt, 2005]

For the heat transfer in the fluid and air the law of cooling is expressed, and the equation yield the resistance as,

$$R_{conv} = \frac{\dot{q}}{\Delta T} = 2\pi r_{1,2} L h_{1,2} \quad (2.6)$$

where 1 and 2 implies the internal and external variable respectively. Recalling the rule of series circuits from electronics, the overall heat resistance, R_{tot} , is expressed as,

$$R_{tot} = R_{conv,1} + R_{cond} + R_{conv,2} = \frac{1}{2\pi r_1 L h_1} + \frac{\ln(r_2/r_1)}{2\pi Lk} + \frac{1}{2\pi r_2 L h_2} \quad [\text{K/W}] \quad (2.7)$$

2.3. External Flow region

This section explains external flow, the theory is retrieved from Incropera and DeWitt [2005]. The external flow over a cylindrical pipe is of significant importance in respect to

heat transfer as it is a product of the Reynolds number. Normally the external flow behaves as a cross-flow over the cylinder, involving a fluid motion normal to the axis. The cross flow is characterized by three main stream movements. In Figure 2.2, V is the upstream velocity brought to rest at the forward stagnation point, and then due to a favourable pressure gradient a boundary layer develops. At this point the pressure decrease and a boundary layer develops from the stagnation point with an increase in x , which is the stream line coordinate indicated in Figure 2.2. u_∞ is the free stream velocity and as the air moves toward the rear end the pressure eventually reaches a minimum and the boundary layer develops further in the presence of an adverse pressure gradient. Because of the adverse pressure gradient, the velocity starts to decelerate and when the velocity gradient $\frac{\partial u}{\partial y}|_{y=0}$ eventually reaches zero, flow separation occur and the boundary layer detaches from the surface of the cylinder creating a wake downstream as shown in Figure 2.2.

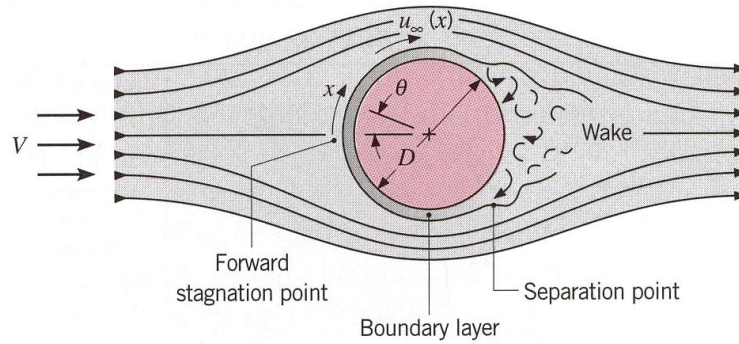


Figure 2.2: Boundary layer formation on a cylindrical cross section exposed to cross flow. [Incropera and DeWitt, 2005]

The angle, θ , indicates the separation point of boundary layer as an angle from stagnation point to separation, this point is called boundary layer transition. This is dependent on the Reynolds number.

$$Re_D = \frac{\text{Inertial force}}{\text{Viscous force}} = \frac{uD_h}{\nu} \quad (2.8)$$

where Re_D refers to the use of a hydraulic diameter, D_h . However, for a cylinder this is the outer diameter.

Different Reynolds regime will change the separation angle. This can be investigated closer in Figure E.1 in Appendix E.

From experiments it is shown that also the Nusselt number is dependent on the separation angle, θ . This can be explained as the Nusselt number increase with an increase in Reynolds number which is due to a decrease in boundary layer thickness. This relation has developed several correlations for the Nusselt number. In general Nusselt number can be said to be a function of,

$$\overline{Nu} = f(Pr, Re)$$

where Prandtl number is the ratio of viscous diffusion and thermal diffusion,

$$Pr = \frac{\text{Viscous diffusion}}{\text{Thermal diffusion}} = \frac{\mu c_p}{k} \quad (2.9)$$

This implies that the Nusselt number is a correlation between convection and conduction. Investigations of the two most common correlation, "Hilpert" approach and "Churchill and Bernstein" approach shows that there is about 3% difference between the two approaches [Incropera and DeWitt, 2005]. As the Churchill correlation, equation (2.10), is valid for all ranges of Reynolds number, as well as a wide range of Prandtl numbers, $Re_D Pr \geq 0.2$. It is assumed that this will be the best approximation.

$$Nu = 0.3 + \frac{0.62 Re_D^{1/2} Pr^{1/3}}{[1 + (0.4/Pr)^{2/3}]^{1/4}} \left[1 + \left(\frac{Re_D}{282,000} \right)^{5/8} \right] = \frac{\bar{h} D_e}{k} \quad (2.10)$$

From equation (2.10) the correlation between convection and conduction may be seen, and as the only unknown variable is the local heat transfer coefficient. This leads to the possibility of defining the overall heat transfer coefficient, h_{tot} ,

$$h_{tot} = \frac{1}{\frac{1}{h_1} + \frac{WT}{k_{steel}} + \frac{1}{h_2}} \quad [\text{W/m}^2\text{K}] \quad (2.11)$$

where WT is the wall thickness and h_1 and h_2 stands for internal and external heat transfer coefficient respectively.

2.4. Internal Flow Region

The internal flow is considered nearly fully developed. This means that there is only a non-zero velocity in the x-direction [Daugherty et al., 1985]. The mean velocity in the main pipe is defined from the static equilibrium of Bernoulli's equation (2.12),

$$P_1 + \frac{1}{2} \rho u_1^2 + \rho g h_1 = P_2 + \frac{1}{2} \rho u_2^2 + \rho g h_2 + \Delta P_{friction} \quad (2.12)$$

Assuming the velocity may be calculated from the principle of liquid column as shown in Figure 2.3. Where $u_1 = u_2 = 0$, $P_2 = P_{atm}$ and friction is neglected. Rearranging the velocity can be expressed as equation (2.13),

$$u_2 = \sqrt{\frac{2}{\rho} (P_1 - P_2 + \rho g h_1)} \quad (2.13)$$

It is assumed that the dead leg is exposed to stagnant flow, except for some circulation which occurs as an effect of the fluid flow over the inlet. Depending on the velocity of the

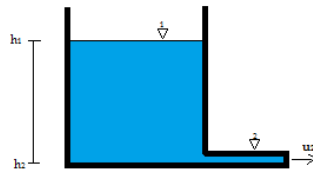


Figure 2.3: General presentation of the Bernoulli principle of liquid column.

flow in the main pipe, the degree of circulation will vary. This is what both Hong [1977] and Habib et al. [2005b] studied in their reports. Hong [1977] concluded from his studies of natural circulation that as the non-dimensional wall parameter became larger than ten, the heat transfer coefficient would become uniform, and therefore the result would lead to the exact analytical solution for a Fin. Habib et al. [2005b] performed experimental and CFD-analysis to see the flow pattern for horizontal and vertical dead legs.

2.5. Pipe and Fluid Properties

The study is performed with standard pipe dimensions. A rule of thumb in regards to dead legs is that if shorter than 0.5m and at least a 2" diameter there is no need for insulation [Osenbroch, 2012].

In addition to pipe size the pipe schedule has to be defined. A pipe schedule refers to the wall thickness and is set so that the pipe wall may resist the internal pressure, with tolerances for corrosion, erosion and mechanical allowances such as treads [Sinnott, 2009]. The schedule number is based on a thin cylinder formula and is defined as,

$$\text{Schedule no.} = \frac{P_s \times 1000}{\sigma_s} \quad (2.14)$$

where P_s is the safe working pressure and σ_s is the safe working stress. For straight low pressure pipes the general approach is to use schedule 40 for carbon steel pipes and 40S for stainless steel if the pipe dimension is below DN200. Table 2.1 shows the general approach on deciding a pipe schedule for straight pipes [Shell AS, 2011].

The internal fluid consists of produced water and it is assumed to have constant fluid properties. Also pipe material and air is assumed to be independent of the temperature change. The thermal properties at the initial temperatures is presented in Table 2.2

2.5. PIPE AND FLUID PROPERTIES

Table 2.1: Pipe Schedule

Pipe size	Pipe SCH
Pipe DN50-DN150	SCH. ≥ 40
Pipe DN200-DN750	SCH. ≥ 30

Table 2.2: Properties of the applicable substances

Properties	Water at 323K	Carbon Steel	Air at 266K
ρ [kg/m ³]	998.5		1.3200
c_p [J/kgK]	4181.2		1006.3
k [W/mK]	0.64	36.9	0.0236
$\alpha \times 10^{-6}$ [m ² /s]	0.154	-	18.01
$\mu \times 10^{-7}$ [mPa·s]	5476	-	167.6
$\nu \times 10^{-6}$ [m ² /s]	0.54	-	12.86

3. Turbulence modelling

The turbulent flow is another important area to investigate as most pipe flows are turbulent. To develop a realistic simulation a turbulence model must be incorporated. For most engineering simulations the available turbulence models are either Reynolds-Averaged Navier-Stokes (RANS) simulations or Large-Eddy Simulations (LES). The LES model demands a significantly better processor capacity. However, it may be more accurate [Mme, 2010]. For this study, the RANS model is considered to be sufficient. The following sections are mostly retrieved from Versteeg and Malalasekera [2007] and Voigt [2001].

3.1. Governing Equation in Turbulent pipe flow

For the Reynolds Average Navier-Stokes Simulation (RANS) the flow is decomposed into an averaged part and a fluctuating part, $u(t) = \overline{u(t)} + u'(t)$. The fluctuations are always behaving in a 3D motion, and particles far apart may be brought together by the eddy motions which occur in the turbulent flow. This results in an effective mixing of fluid particles, which leads to high values of diffusion coefficients for mass, momentum and heat [White, 2006]. In a turbulent flow the governing equations are the continuity and Navier-Stokes equations combined with the transport equation for the two quantities turbulent kinetic energy, k , and dissipation of kinetic energy, ϵ [Voigt, 2001].

The continuity and Navier-Stokes equations for turbulence are then expressed from the governing equation of incompressible flow. Decomposing and performing an ensemble averaging of the turbulent governing equations, the Reynolds-Average Navier Stokes (RANS) equation can be expressed as,

$$\frac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \frac{\partial \overline{u}_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial \overline{p}}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\nu \frac{\partial \overline{u}_i}{\partial x_j} - \overline{u'_i u'_j} \right) \quad (3.1)$$

If the process of decomposing and ensemble is desired to be further investigated, this can be studied in Voigt [2001].

From equation (3.1) the term, $\overline{u'_i u'_j}$, defines the Reynolds stresses, where i and j denotes the direction. The full Reynolds stress tensor is symmetric, and contains 6 additional un-

knowns. Consequently, there are more unknowns than equations, this is known as the closure problem. This is resolved by applying the Boussinesq approximation, which assumes proportionality between the deviatoric part and the Reynolds stress tensor [Voigt, 2001],

$$-\overline{u'_i u'_j} = \nu_t \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} k \quad (3.2)$$

where ν_t defines the eddy viscosity, modelled from equation (3.3),

$$\nu_t = C_\mu \frac{k^2}{\varepsilon} \quad (3.3)$$

where the model constant $C_\mu=0.09$.

Taking the divergence of the Reynolds stress tensor, equation (3.2), it is inserted equation (3.1) and equation (3.4) is obtained,

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial (\bar{p} + \frac{2}{3} \rho k)}{\partial x_i} + \frac{\partial}{\partial x_j} \left((\mathbf{v} + \nu_t) \frac{\partial \bar{u}_i}{\partial x_j} \right) \quad (3.4)$$

where

$$\mathbf{v} + \nu_t = \mathbf{v}_{eff}$$

3.2. The k - ε model equations

The k - ε model is well established and widely used. It is derived for high Reynolds number flow and the coefficients are empirically derived. From experimental evidence Versteeg and Malalasekera [2007] indicate that Boussinesq's proposal in 1877 ensures that the formula gives the correct result when subjected to normal Reynolds stresses. For this model the transport equations are kinetic turbulent energy, k , which is defined as,

$$k = \frac{1}{2} \overline{(u'_i u'_i)}$$

and the dissipation of turbulent kinetic energy, ε , which is defined as,

$$\varepsilon = \nu \overline{\frac{\partial u'_i}{\partial x_j} \frac{\partial u'_i}{\partial x_j}}$$

The final equation for these two transport properties, k and ε , takes the following final form as expressed in equation (3.5) and equation (3.6) respectively. The entire derivation can be investigated further in Voigt [2001].

$$\frac{\partial k}{\partial t} + \bar{u}_j \frac{\partial k}{\partial x_j} = \nu_t \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial}{\partial x_j} \left[\left(\mathbf{v} + \frac{\nu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right] - \varepsilon \quad (3.5)$$

$$\frac{D\varepsilon}{Dt} = C_{\varepsilon_1} \frac{\varepsilon}{k} \left(-\overline{u'_i u'_j} \right) \frac{\partial \overline{u}_i}{\partial x_j} + \frac{\partial}{\partial x_j} \left[\left(\nu + \frac{\nu_t}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_j} \right] - C_{\varepsilon_2} \frac{\varepsilon^2}{k} \quad (3.6)$$

The eddy viscosity is expressed in equation (3.3) and the model constants are

$$C_{\varepsilon_1} = 1.44, \quad C_{\varepsilon_2} = 1.92, \\ \sigma_k = 1.0, \quad \sigma_\varepsilon = 1.3$$

Furthermore, Versteeg and Malalasekera [2007] states that model equations for k and ε requires similar boundary conditions as other elliptic flow equations. As these transport properties are based on measurements and as there rarely is any measurements available, the parameters are often found from the literature. If no literature of similar cases are available, the inlet distributions for k and ε can be obtained from a correlation with turbulence intensity, $T_i = 6\%$, and a characteristic mixing length, $\ell = 0.07L$, where L is the diameter in a cylindrical pipe [Versteeg and Malalasekera, 2007].

$$k = \frac{2}{3} (U_{ref} T_i)^2 \quad (3.7)$$

$$\varepsilon = C_\mu^{\frac{3}{4}} \frac{k^{\frac{3}{2}}}{\ell} \quad (3.8)$$

where U_{ref} is the reference internal velocity.

In the k - ε model the transport equations are not integrated at the walls. Instead the production and dissipation of the kinetic energy is specified in the near wall cell, using the logarithmic law-of-the-wall [Voigt, 2001]. The logarithmic law-of-the-wall is of less constraint than the inner law. A more detailed description of the wall laws is explained below, or it can be found in White [2006].

The wall coordinate, y^+ , is a dimensionless parameter and it is an accurate way of determining the distance from the wall to the nearest data-point in the mesh. As the flow solution is computed on the cell center, this means that the first data-point from the wall will be in the middle of the first cell as Figure 3.1 indicates.

There is three flow regions concerning the wall coordinates,

The viscous sub-layer, $y^+ \leq 5$, this is the region very close to the wall and it is assumed that the boundary layer is linear and that the inner-law of the wall is applicable [White, 2006].

The buffer layer, $5 \leq y^+ \leq 30$, is a region where the profile is neither linear nor logarithmic but a smooth merge between the two [White, 2006].

The log layer, $y^+ \geq 30$, apply the outer law, also called the logarithmic law of the wall [White, 2006].

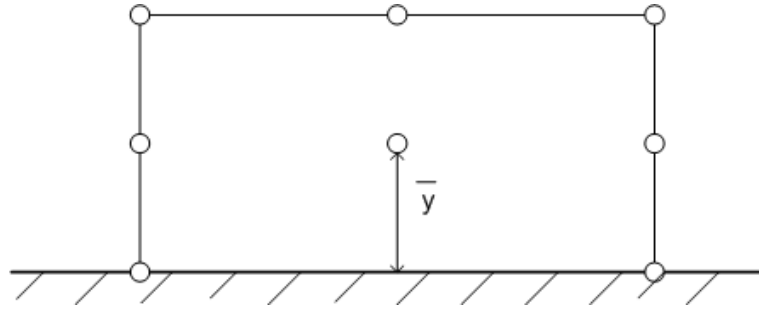


Figure 3.1: Control volume indicating the first data-point from the wall [Mme, 2010]

The y^+ regions serves as an indicator on how fine the mesh has to be to achieve accurate calculations.

The wall coordinates of y^+ is then calculated from equation (3.9),

$$y^+ = \frac{\bar{y}v^*}{\nu} \quad (3.9)$$

where \bar{y} is the distance from the wall to the first data-point, see Figure 3.1, v^* is the wall friction expressed by the kinetic turbulent energy, k and the constant, $C_\mu = 0.09$.

$$v^* = k^{\frac{1}{2}} C_\mu^{\frac{1}{4}}$$

Calculating the wall coordinate will provide the operator the necessary knowledge to decide if the grid model is fine enough for the simulated case.

4. Computational Fluid Dynamics

Computational Fluid Dynamics, CFD, is a collective term for the analysis of systems involving fluid flow, heat transfer and associated phenomena using computer simulations. The CFD codes are structured around the numerical algorithm which can handle the fluid flow problem. In CFD codes there are three main elements [Versteeg and Malalasekera, 2007], namely

Pre-processor, defines the geometry of interest, the grid generation and the physical problem to be solved.

Solver, solves the fluid equations on the given grid using either a finite difference, finite element or finite volume method.

Post-processor, visualize the results in the form of 3D/2D surface contours, graphs and tables.

The CFD code is an interpretation of a complex physical problem. Prior to developing the CFD code an understanding of the physical problem and the algorithms is necessary.

Another complex part when working with CFD is that the validity of the model might look sufficient, however, there might be errors which occur from the complexity of the code. To actually validate the results achieved from the simulation, experimental data and research reports should be used for comparison. However, there is not always extensive experimental work available, so then the verification must be done by the use of analytical solutions of similar simplified problems. Another source to compare the results is by high-quality data from closely related problems found in literature or journals. [Versteeg and Malalasekera, 2007]

4.1. Finite Volume Method

The Finite Volume Method (FVM) is a numerical method which discretizes a volume. There are 3 basic steps which should be understood [Versteeg and Malalasekera, 2007].

Step 1 - Grid generation, the domain is divided into control volumes of which the governing equations can be integrated over. The finer the grid distribution the more accurate solution is obtained. This step will be of great importance for the overall result.

Step 2 - Discretization, is the key of this method. After dividing the structure into smaller control volumes, the governing equations are integrated over each one of them. The process of discretizing is fairly comprehensive. This is the reason why the CFD programs are so attractive.

Step 3 - Solution of equation, the result of the above discretization is a system of equations which must be solved. The typical equations apply to either direct or indirect methods. One example is the TDMA which is a direct method for one dimensional problems, but when applied iteratively it may also be applied for multi-dimensional problems.

4.2. OpenFOAM

Open Source Field Operation and Manipulation (OpenFOAM) is a C++ library which is developed by OpenCFD.ltd. It is released under the GNU general public license. This is an advantage as the program is freely distributed and works as an open source. The user guide is satisfactory, and OpenFOAM is a program with a great range of application. The user can develop his/her own codes and solvers, which creates a range of possibilities.

The environments in OpenFOAM is divided into three parts as explained in the introduction of this chapter, Figure 4.1 indicates this,

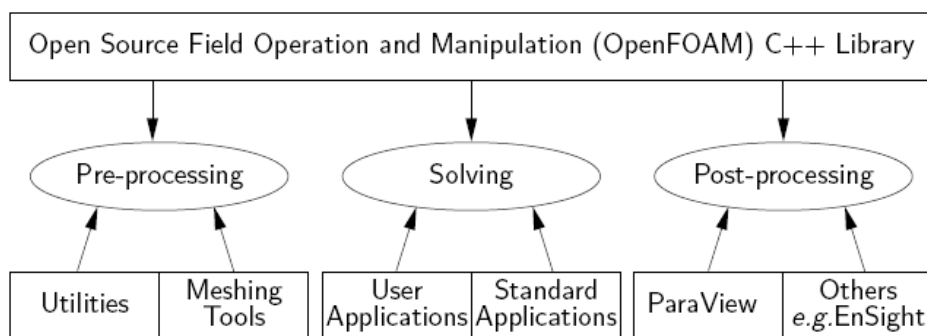


Figure 4.1: Overview of the environments in OpenFOAM [Foundation, 2011]

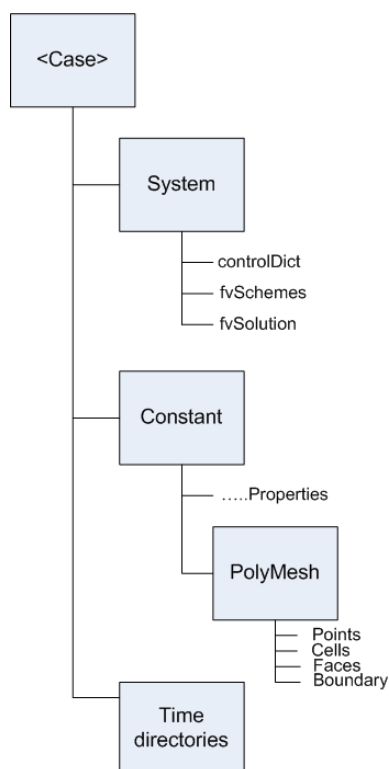


Figure 4.2: The file structure in the directories in OpenFOAM [Foundation, 2011]

The OpenFOAM dictionary files are used to define a specified case. They also define the necessary, physical and numerical, conditions to solve the problem. The file structure is presented in Figure 4.2 [Foundation, 2011].

The initial case file includes three basic directories. The time directories are folders created for each time step the solver is executed. Before the execution only the boundary conditions and initial conditions are specified, with the folder notation 0. For a steady state solvers only the initial and the converged final results are of interest, while for transient solvers animations are created over all time directories.

The constant folder contains the physical properties and grid distribution. The mesh properties are defined in the sub-folder called polyMesh. In this folder, OpenFOAM saves the files which include the mesh properties, *i.e.* boundaries, points, cells and faces. The physical properties which are defined are dependent on the solver. However, most solvers include the transport properties. For other solvers, additional properties such as turbulence and gravity may be defined.

The final folder in the initial solver is the system folder. This folder is associated with the solution procedure. Containing a minimum of 3 files [Foundation, 2011].

fvSchemes, includes the finite volume schemes that are used. The numerical schemes are comprehensive, however, a brief explanation will be given.

fvSolution, includes the solver and equations which are to be solved. Also the tolerances and relaxation factors are stated in this file.

controlDict, contains the time set-up. All parameters which are time dependent are stated here. It is in this file the decision of steady-state or transient study is decided on.

The two following sections will give an introduction to the *fvSchemes* and *fvSolution* files in OpenFOAM. Then the solvers adapted for the present cases will be explained.

4.3. Numerical schemes

The numerical schemes must be set in the *fvSchemes* file. OpenFOAM provides a range of numerical schemes [Foundation, 2011]. To initialize the *fvSchemes* file the operator may decide on the discretization practice of desire. However, the Gaussian finite volume method is the most common. Gauss-method is based on summing the values on cell faces, which is interpolated from cell centres [Foundation, 2011]. Furthermore, the interpolation scheme, divergence scheme, Laplacian scheme and gradient scheme must be chosen. OpenFOAM is again providing a range of possibilities to meet the operator's requirement.

The *fvSchemes* file is divided into sub dictionaries for each numerical scheme. Table 4.1 indicates how they are set for the cases investigated in this thesis.

Table 4.1: Numerical Schemes in OpenFOAM

Case no.	Time scheme $\left(\frac{\partial}{\partial t}\right)$	Grad scheme (∇)	Laplacian scheme (∇^2)	Divergence Scheme ($\nabla \cdot$)	Interpol. scheme
01	Steady State/ Euler	GaussLinear	GaussLinear Corrected	none	linear
02a	Steady State	GaussLinear	GaussLinear Corrected	none	linear
02b	Steady State	GaussLinear	GaussLinear Corrected	GaussUpwind/ GaussLinear	linear
03	Euler	GaussLinear	Gauss Linear Corrected	none	linear

The time scheme in *fvSchemes* defines the method of time dependency. Table 4.1 implies that two time schemes are used. That is steady-state and the Euler method. If desired, OpenFOAM also provide other time schemes. The Euler method is a first order numerical procedure. There are two types of Euler methods, one explicit called forward Euler and one implicit method called backwards Euler. It is the backwards Euler which is incorporated into OpenFOAM. The backward Euler is more CPU costly as a system of equation must be solved at each step. However, it eliminates the stability problems which can occur in the explicit method. [Foundation, 2011]

In OpenFOAM the general expression of the gradient term involves the discretized method, hence Gauss, and the chosen interpolation scheme. In the dictionary it is expressed as below.

$$\text{GradientScheme} \rightarrow \text{Gauss} \langle \text{interpolationScheme} \rangle$$

From Table 4.1 it can be seen that the Gradient scheme is set to Gauss Linear in all cases, where Gauss is the discretizing method and Linear is the interpolationScheme. The interpolation scheme contains terms that are used to interpolate values typically from cell center to face center. This is commonly called the Central Difference Scheme (CDS) and is a second order method. It is expressed as a part in the other schemes, but it may also stand alone [Foundation, 2011]. From Table 4.1 it can be seen that the interpolation scheme is set to *default linear*;

In OpenFOAM the general expression of the Laplacian scheme involves the discretized method, the interpolation scheme and the surface normal gradient scheme.

$$\text{Laplacian Scheme} \rightarrow \text{Gauss} \langle \text{interpolationScheme} \rangle \langle \text{snGradScheme} \rangle$$

The surface normal gradient is required to evaluate the Laplacian term, but it can also be specified on its own right. However, in the cases studied in this thesis it does not stand alone. In Table 4.1 under the Laplacian scheme it may be observed that the normal surface gradient is the corrected scheme which is an explicit non-orthogonal correction unbounded, following a second order and conservative numerical behaviour [Foundation, 2011].

The divergence scheme is defined in OpenFOAM by the discretizing method and the interpolation scheme.

$$\text{Divergence Scheme} \rightarrow \text{Gauss} \langle \text{interpolationScheme} \rangle$$

From Table 4.1 it can be observed that only Case 03 has a defined divergence scheme. In addition it can be observed that there are two types of interpolation schemes. The reason for this is that the most stable upwind scheme does not work for one of the terms in Case 03, the $\nabla v_{eff} dev T \nabla U$. Therefore, an unbounded, second order scheme of Gauss Linear is used. However, trying to use the Gauss Linear scheme on all fields creates divergence and the solver will not be executed.

4.4. Solution algorithms

In regards to solvers there is a range of alternatives. Even if the solvers are specified to suit certain problems, they may still be modified to suit the problem of interest.

As a suitable solver is chosen the *fvSolution* file must be updated to fit the specified problem. The file contains a directory for every variable that requires an equation solver. Furthermore, the global iteration scheme is stated together with relaxation factors.

4.4.1. Linear solver control

The type of solver used depends upon the matrix symmetry. In Foundation [2011] the available linear solvers are presented. However, in the simulations in this thesis, only the preconditioned (bi-) conjugate gradient (PBiCG) is used. The difference between the PCG and PBiCG is that PCG is applicable for symmetric matrices, while PBiCG is applicable for non-symmetric matrices.

4.4.2. Tolerance and residuals

All variables are given its own directory in the *fvSolutions* file, and each one need to have a defined tolerance and relative tolerance. The tolerance given for the specified variable acts as a place holder. Thus, the residuals of the solution needs to be below a given value. Consequently, the initial residuals are evaluated by the current field values before solving the equation, if the residuals go above the field tolerance the solver will stop. Therefore, the tolerance level should be set to a limit which maintains a reasonable accuracy of the solution. The relative tolerance will act as a place-holder for the ratio of the current residuals to the initial residuals. [Foundation, 2011]

4.4.3. Relaxation Factors

The effect of relaxation factors determines how fast the steady solution converges. The steady case with the SIMPLE algorithm requires under-relaxation factors. The relaxation factors imply how much of the old initial and the new iterated value that should be used in the next iteration cycle. For the dependent variable field the relaxation equation will be,

$$X_{new} = (1 - r) \times X_{old} + r \times X_{calculated} \quad (4.1)$$

where X is the applicable variable and r is a given relaxation factor.

The applicable variables for X will be velocity, pressure, temperature and turbulence parameters. A relaxation factor close to 1 indicates that a larger fraction of the new value

is used in the iteration. This can be seen from equation (4.1) as $r = 1$ will make the X_{old} equal to zero. This might lead to divergence [Versteeg and Malalasekera, 2007].

4.4.4. Semi-Implicit method for Pressure-Linked Equation (SIMPLE) algorithm

This algorithm is a steady state algorithm, which follow an iterative procedure for solving the pressure and velocity coupling. In the SIMPLE algorithm the solver only makes 1 correction, indicating that it is a steady state solver algorithm. There is however an additional correction, the non Orthogonal Corrector. This is a corrector which takes into account the non-orthogonality of the mesh. An orthogonal mesh is when the face normal in the mesh is parallel to the vector between the centre of the cells that the face connects to. Depending on the mesh the keyword *nNonOrthogonalityCorrectors* in *fvSolution* is set to 0 for an orthogonal mesh upto max 20 for a non-orthogonal, unstructured, mesh [Foundation, 2011].

In addition to the corrector the SIMPLE algorithm is essentially a guess-and-correct procedure for the calculation of pressure. To initiate it, boundary conditions are set, initializing a guessed pressure field, p^* . The discretized momentum equation is solved so that the intermediate velocity field is computed. Thereafter, the correctors, p' , are defined, so that the correct pressure becomes,

$$p = p^* + p'$$

This procedure is performed in a similar matter for the velocity fields. Then again the corrected pressure field and velocity field is inserted into the discretized momentum equation. Then continue by solving the pressure correction equation and apply the under-relaxation, equation (4.1). Thereafter, all other discretized transport equations may be solved. This cycle continues until convergence is reached and the cycle stops [Versteeg and Malalasekera, 2007]. A flow chart from Versteeg and Malalasekera [2007] is attached in Appendix D.

4.5. OpenFOAM Solver

This thesis will present different cases and to do so more than one solver is adapted. Implementing realistic assumptions to simplify the problem, the solution may be presented as a Fin analysis. This assumes that the water is stagnant and there is no circulation in the dead leg. The fin analysis is easy to validate to existing theory. Subsequently, a realistic simulation will be developed. In the following sections the adapted OpenFOAM solvers are presented, and it is based on information retrieved from Foundation [2011] and WikiFOAM [2012].

To fully understand the solver libraries it is worth noting how OpenFOAM writes the equations. Table 4.2 express the commonly terms which is used in the C++ language when writing equations.

Table 4.2: Equation discretization in OpenFOAM

Term Description	Implicit/ Explicit	Mathematical expression	C++ language (<i>fvm</i> :: / <i>fvc</i> ::)
Laplacian	Implicit/ Explicit	$\nabla \cdot D_t \nabla T$	laplacian(DT,T)
Time derivate	Implicit/ Explicit	$\frac{\partial T}{\partial t}$	ddt(T)
Divergent	Implicit/ Explicit	$\nabla \cdot (\phi T)$	div(phi, T)
Source	Implicit	$T \nabla \phi$	<i>Sp</i> (<i>fvc</i> :: <i>div</i> (<i>phi</i>), <i>T</i>)

For the cases developed in this thesis the following solvers were adapted,

LaplacianFoam, is used to solve steady and transient thermal diffusion in a solid [Foundation, 2011].

PotentialFoam, is a potential flow solver which is used to generate starting fields for full Navier–Stokes (NS) codes [Foundation, 2011].

BuoyantBoussinesqSimpleFoam, is a steady state solver for buoyant, turbulence flow of incompressible fluid [Foundation, 2011].

The solver libraries are listed in Appendix G and the developed case files are included in Appendix I which is a CD-hard drive.

4.5.1. LaplacianFoam

The LaplacianFoam is a steady state and transient solver for pure diffusion. The equations concern only one variable, the temperature. It is expressed in terms of the diffusion rate, D_t . The time derivative shown in equation (4.2) is neglected when executing a steady state simulation.

$$\frac{\partial T}{\partial t} + \nabla(D_t \nabla T) = 0 \quad (4.2)$$

where T is the temperature and D_t is the diffusion. As the source code is executed, the code calls the files one by one. First it sets the correct path, then it creates the time directory and mesh, before it creates the temperature field and sets the thermal diffusivity. When this is completed, the time loop is set, and the calculation start on the first time step.

The LaplacianFoam code uses the non-Orthogonal corrector from SIMPLE. The result is then written out for each time step, and the results are then available for post-processing.

4.5.2. PotentialFoam

The potentialFoam solver use the potential flow theory to develop an internal velocity field. The potential flow model assumes in-viscid fluid and an irrotational velocity field, u .

$$\text{rot}(u) = 0$$

From the irrotational velocity field the velocity potential ϕ is introduced, and the velocity vector becomes,

$$u = \nabla\Phi \quad (4.3)$$

The continuity equation for incompressible flow, $\nabla \cdot u = 0$, creates the Laplace equation for the velocity potential, Φ ,

$$\nabla^2\Phi = 0 \quad (4.4)$$

The potentialFoam solver does not execute any time loop. It rewrites the initial velocity field by integrating the momentum equation to give Bernoulli's equation (4.5),

$$\frac{p}{\rho} + \frac{1}{2}V^2 + gz + \frac{\partial\Phi}{\partial t} = \text{constant through flow field}, \quad V = |u| \quad (4.5)$$

The above correlation is why the potential flow is described by the scalar Laplace equation. In addition, the potential flow solver produces the corresponding stream functions, ψ . The potential flow theory is not realistic. However, it increases the stability of the flow field when adapted to more advanced solver.

4.5.3. BuoyantBoussinesqSimpleFoam

This solver is a more advanced solver which is time independent, this imply that it is a steady-state solver. This solver is for buoyant, turbulent flow of incompressible fluids. For the cases studied in this thesis the process medium is liquid, so no buoyancy is occurring. Inserting β to zero, the buoyancy term is neglected, and the solver solves pressure, velocity and temperature equations respectively. The complete library of buoyantBoussinesqSimpleFoam is attached in Appendix G.3.

The overall implementation is that while the simple loop is running, the code includes the velocity equation, temperature equation and pressure equation. Thereafter, the turbulence variables are corrected, and the solver is executed over the set execution time. Note that the equations are expressed in terms of the mean quantity and turbulence terms. The velocity, temperature and pressure equations are expressed below,

Velocity equation:

As the simple loop solves the momentum equation it first predicts the velocity by implementing the library file called *Ueqn.H*. The velocity is predicted by an implicit method, which means that a set of linear equations are solved in matrix-vector form, $Ax = b$. Accordingly, the implicit left hand side is set up as equation (4.6),

$$\nabla \cdot (\phi U) + \text{turbulence-} > \text{divDevReff}(U) \quad (4.6)$$

where the term "turbulence- >" imply that the term "divDevReff(U)" is retrieved from the turbulence library, Appendix G.3.5. The term is rewritten from C++ and will be expressed as equation (4.7),

$$\text{divDevReff}(\bar{u}) = -\nabla^2 \nu_{eff} \cdot \bar{u} - \nabla \cdot \{ \nu_{eff} \text{dev} [T \nabla \bar{u}] \} \quad (4.7)$$

where $\text{dev} = A - \frac{1}{3} \text{trace}(A)I$. The notation A indicate the matrix to be solved and I is the identity matrix.

Further study of the velocity library, G.3.1, imply that the velocity equation is then under-relaxed before it is solved for the momentum predictor.

Temperature equation:

As the velocity predictor, the temperature predictor is implicit. It is approached by first solving equation (4.8) which is the turbulent diffusivity, κ_t .

$$\kappa_t = \frac{\nu_t}{Pr_t} \quad (4.8)$$

Then the efficient diffusivity is calculated from equation (4.9),

$$\kappa_{eff} = \kappa_t + \frac{\nu}{Pr} \quad (4.9)$$

Thereafter, equation (4.10) is solved, before equation (4.11) solves the boussinesq approximation for the effective kinematic density, ρ_k , based on the new temperature value.

$$\nabla \cdot (\phi T) - T \nabla \cdot \phi - \nabla \cdot (\kappa_{eff} \nabla T) = 0 \quad (4.10)$$

In the equation which is written out the implicit source term, $T \nabla \cdot \phi$, is neglected as it is purely related to numerics [?].

$$\rho_k = 1.0 - \beta (T - T_{Ref}) \quad (4.11)$$

where β is the coefficient of the thermal expansion. As mentioned previously the effect of buoyancy is neglected in all simulations as the process only deals with liquid. When $\beta = 0$ the effective kinematic density becomes,

$$\rho_k = 1.0$$

Pressure equation:

The pressure library is attached in Appendix G.3.3 and it is in the pressure equation library that the corrector step takes place. Equation 4.12 is solved and iterated over the number of *NonOrthogonalCorrectors* defined.

$$\nabla \cdot (rAUf) \nabla p_{rgh} = \nabla \cdot \phi \quad (4.12)$$

where $rAUf$ is a stored variable equal to the inverse A matrix. Further, $A^{-1}|_f$ is multiplied by vector H.

$$U = HA^{-1}|_f$$

this correlation is named U, as this term contributes to the corrected velocity in the corrector loop in the SIMPLE algorithm. The complete derivation can be seen in an article by WikiFOAM [2012]. Note that the article investigates the PISO loop so the full implementation of the SIMPLE algorithm may not be retrieved from here. However, the governing equations will be the same, but following the SIMPLE loop.

4.6. Sampling

The result from the simulations will be presented by contour and plots. The results will be sampled in the length direction of the dead leg. That implies the negative z-direction. In addition, probes are installed at certain areas in the domain. They are set at locations in the dead leg with Δz equal to the diameter distance. This implies that the probe locations are found in the z-direction at 0.5D, D, 1.5D, 2D, 2.5D, 3D until the end of dead leg is reached. The distribution is shown in Table 4.3.

Table 4.3: Diameter segments in the vertical deadleg, z-axis

Item	Segment	0	0.5D	D	1.5D	2D	2.5D	3D
T-junction	-z	0.051	0.0771	0.1031	0.1291	0.1552	0.1812	0.2072
Fin	-z	0.0	0.026	0.052	0.78	0.104	0.128	0.150
Item	Segment	4D	5D	6D	7D	8D	9D	end
T-junction	-z	0.2591	0.3112	0.3631	0.4152	0.4671	0.5192	0.552
Fin	-z	0.202	0.254	0.306	0.358	0.410	0.462	0.5

Figure 4.3 illustrate the probe locations.

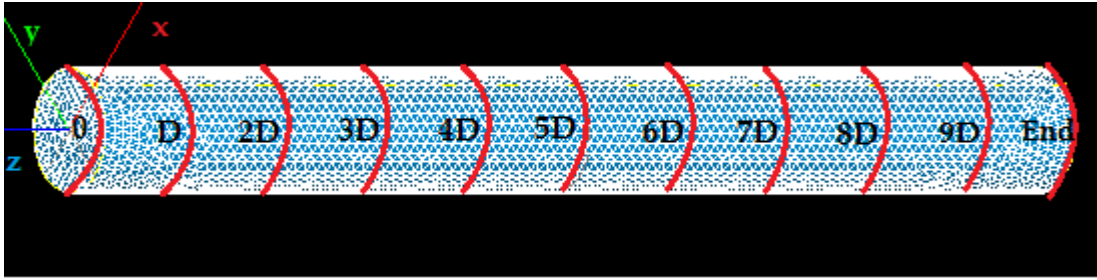


Figure 4.3: Probes located in the dead leg.

5. Industrial approach

The industrial approach is not so different from the typical theoretical cases. The difference is the use of standards and specifications. For a process engineer the first approach to a problem usually goes through an operator company which raises an issue. With known specifications, such as mass flow or pressure, the process engineer uses reference documents to decide on pipe dimensions and class. Every operating company has their own piping classes. As this thesis regards the Draugen field, which is operated by Shell AS, it is their piping classes which are used.

As the process engineer has decided on piping class, dimensions and flow properties, the next approach is for the piping - and mechanical engineers to perform stress calculations and make the layout, as well as defining equipment such as pump and valves from the process data.

A standard is a reference document and it is one of the most important tools when approaching an engineering problem. It provides the engineers a basis of choice in regards to dimensions and parameters which will hold the assigned integrity of the system. For drain points and pipes, standard such as Norsok L-002 and P-001 [L-002, 2009, P-001, 2006], as well as ISO 12241 [2008] provides reference points to deciding on the design criteria. Since this thesis is a collaboration with Aker Solutions the results which are retrieved in the CFD analysis should be compared to the result from the mentioned standards. This will imply whether the procedure provided by these standards is too conservative or if a more detailed investigation should be performed in respect to dead legs. The next sections will present the most relevant reference points and procedures in these standards.

5.1. Norsok P-001 & L-002

These two Norsok standards [L-002, 2009, P-001, 2006] contain general information regarding piping details and design criteria. From Norsok L-002 [2009] the design condition for the ambient temperature is provided,

- Minimum design ambient temperature $-7\text{ }^{\circ}\text{C}$ (266K)

- Maximum design ambient temperature 22°C (295K)

However, in NORSOK P-001 [2006] it is stated that the ambient temperature should be set from historical weather data. This implies that the values given in NORSOK L-002 [2009] only provides a reference point, and should not be used without precautions.

NORSOK P-001 [2006] provides reference limits on line sizing, the pipe roughness, pressure drop and maximum velocity. For liquid filled pipes the maximum velocity is,

- Maximum velocity carbon steel pipe, 6 m/s
- Maximum velocity stainless steel pipe, 7m/s

As Table 2.1 in section 2.5 implies, carbon steel pipes below DN150 are defined by piping schedule 40. The velocity is calculated from equation (2.13) and it may be observed that it is well below the maximum allowed velocity.

$$u = 2.86$$

This will give a Reynolds number of,

$$Re = \frac{D_i u}{\nu} = 534\,190$$

which is far into the turbulent regime.

5.2. BS EN ISO 12241:2008

ISO 12241 [2008] provides the calculation rules of heat transfer related properties of building equipment and industrial installations. It presents the equations and procedure for calculating general heat transfer in steady state. In addition the standard provides equations for calculating the time to reach threshold value (TV) and freezing time. Note that in this international standard the design values for the mean temperature are used. The calculations performed from this procedure will provide results which may be compared to the CFD simulations.

5.2.1. Procedure

The governing equations in this procedure are the same as those expressed in section 2.1. Namely, Fourier's law, equation (2.1), the law of cooling, equation (2.3) and the

thermal resistance, equation (2.5). Figure 5.1 illustrate the different variables used in the equations. Where q is the heat flux, D_e and D_i is the external and internal diameter respectively, and θ_{si} and θ_{se} is the internal and external surface temperature respectively.

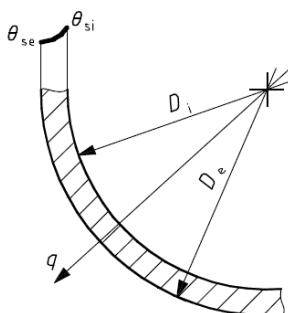


Figure 5.1: Temperature distribution in a hollow cylinder [ISO 12241, 2008]

The law of cooling, equation (2.3) needs a defined convective heat transfer coefficient, h_{cv} . In this standard it has been developed a correlation for this variable. It is defined for both turbulent and laminar flow. As it is assumed that the internal surface coefficient is equal to the medium inside the pipe, it may be neglected. Accordingly, only the external surface coefficient is found from the following correlation,

$$h_{cv,L} = \frac{8.1 \times 10^{-3}}{D_e} + \pi \times \sqrt{\frac{u_a}{D_e}} \quad [\text{W/m}^2\text{K}] \quad (5.1)$$

$$h_{cv,T} = 8.9 \times \frac{u_a^{0.9}}{D_e^{0.1}} \quad [\text{W/m}^2\text{K}] \quad (5.2)$$

where D_e is the external insulation diameter [m], u_a is the air velocity [m/s], cv stands for convection and T and L is turbulent and laminar respectively. If the flow condition is laminar or turbulent, equation (5.1) or equation (5.2) is calculated respectively. The defined turbulence criteria are

$$h_{cv,L} : u_a \times D_e \leq 8.55 \times 10^{-3}$$

$$h_{cv,T} : u_a \times D_e \geq 8.55 \times 10^{-3}$$

Furthermore, the reciprocal of the external surface coefficient, h_{cv} , gives the linear surface resistance, R_{ls} ,

$$R_{ls} = \frac{1}{h_{cv} \pi D_e} \quad [\text{mK/W}] \quad (5.3)$$

From the above procedure the linear thermal transmittance, U_l , for pipes is calculated,

$$U_l = \frac{1}{R_{T,l}} \quad [\text{W/mK}] \quad (5.4)$$

where

$$R_{T,l} = R_{s,i} + R + R_{s,e} = \frac{1}{h_i \pi D_i} + R + \frac{1}{h_e \pi D_i} \quad [\text{mK/W}]$$

Using the thermal transmittance, the total heat flow rate, Φ , of the pipe can be found from equation (5.5),

$$\Phi_T = U_l L (\theta_i - \theta_a) \quad [\text{W}] \quad (5.5)$$

where U_l is the linear thermal transmittance, L is the length in meters and $\theta_{i,a}$ is the internal medium and ambient temperature respectively expressed in degree Celsius.

As the temperature change which actually is tolerated in a pipe is relatively small the change in temperature can be found approximately from the following equation,

$$\Delta\theta = \frac{\Phi_T \times 3.6}{\dot{m} c_p} \quad (5.6)$$

where $\Delta\theta$ is the longitude temperature change, \dot{m} is the mass-flow and c_p is the heat capacity of the medium.

If a more accurate value is desired, the following equation can be used,

$$\Delta\theta = |\theta_i - \theta_a| e^{-\alpha L} \quad (5.7)$$

where $\alpha = \frac{U_l \times 3.6}{\dot{m} c_p}$.

Another section of this international standard investigates the cooling and freezing time of liquids, both for flowing liquid and for stagnant liquid. The cooling time for flowing liquid is calculated from equation (5.8),

$$t_f = \frac{(\theta_{im} - \theta_a) m_w c_{pw} \ln \left(\frac{\theta_{im} - \theta_a}{\theta_{fm} - \theta_a} \right)}{\Phi_T \times 3.6} \quad [\text{h}] \quad (5.8)$$

When subjected to stagnated liquid it is almost impossible to keep the liquid from freezing over time unless actions are taken. The process of cooling will start as soon as there is no longer an efficient circulation of the fluid. The time until freezing starts is expressed from equation(5.9),

$$t_f = \frac{(\theta_i - \theta_a) (m_w c_{pw} + m_p c_{pp}) \ln \left(\frac{\theta_{im} - \theta_a}{\theta_{fm} - \theta_a} \right)}{\Phi_T \times 3.6} \quad [\text{h}] \quad (5.9)$$

where

Φ_T is the total heat flow rate [W] from equation (5.5).

θ_a is the ambient temperature [$^{\circ}\text{C}$].

θ_{im} is the initial medium temperature [deg $^{\circ}\text{C}$].

θ_{fm} is the final medium temperature [$^{\circ}\text{C}$].

m_p and m_w is the mass of the pipe and water respectively [kg].

c_{pw} and c_{pp} is the specific heat capacity for water and pipe respectively [kJ/kgK]

Note that the total heat flow rate, Φ_T , will be different if the pipe is uninsulated. This is because the external surface coefficient must be taken into consideration. Then Φ_T becomes,

$$\Phi_T = h_{se}(\theta_{im} - \theta_a) \pi D_e L \quad [W] \quad (5.10)$$

After the freezing is started it is also of interest to see how fast the whole pipe freeze. The freezing time depends on the density of heat flow and the diameter of the pipe. It is found from equation (5.11),

$$t_{fr} = \frac{f}{100} \times \frac{\rho_{ice} \pi D_i^2 \Delta h_{fr}}{\Phi_{T,fr} \times 3.6 \times 4} \quad [h] \quad (5.11)$$

where

f is mass fraction of frozen water in percent

Δh_{fr} is the latent heat of ice formation, 334 [kJ/kg]

ρ_{ice} is the density of ice, 920 [kg/m³]

and the heat flow rate of freezing is found from,

$$\Phi_{T,fr} = \frac{\pi(-\theta_a)}{\frac{1}{2\lambda} \ln\left(\frac{D_e}{D_i}\right)} \ell \quad [W] \quad (5.12)$$

As this section imply, ISO 12241 [2008] provides a stringent procedure on how to find the important factors regarding heat loss in pipes. However, it does not say anything on how the cooling behaves, or where the first sign of cooled water will be located. A numerical scheme has been implemented in an Aker Solutions [2011] summer intern project . A finite difference method was defined so that the steady state temperature through the dead leg could be observed. The approach is based on the analytical fin-analysis. Hence the calculations only show the averaged temperature loss in the pipe. Investigations of the code imply that the numerical set up is valid and it will be a good comparison to the results obtained from a steady heat transfer simulation in OpenFOAM, assuming that the dead leg can be regarded as a Fin.

5.3. Operating procedures for offshore installations

This section is developed from discussions with the operation team at Shell AS [2012] and from an operating procedure for operating Nyhamna field in cold weather [Shell AS, 2010]. As one aspects in this thesis is to investigate the existing approach and methods on how to handle situations which could decrease the system integrity. A discussion with the

operation team in Shell AS was arranged. This served as a purpose of understanding the many opinions regarding the topic. One special issue which created disagreement was the effect of wind. In the existing operating procedures for cold weather it is stated that the chill effect from the wind is not relevant for equipment. However, the operation team has experienced differently. Consequently, the operation team indicate that when the wind velocity is strong, measures such as wind shielding is performed.

In addition to wind, the design temperature is also important for the operation personnel. As mentioned before, Norsok L-002 [2009] will provide a design criterion, but in addition Norsok P-001 [2006] state that historical data should be investigated before determining environmental design criteria. From investigations of several oil fields which Shell AS operates this seems to be the overall approach. At Draugen the ambient design temperature is set to -9°C and on Nyhamna the ambient design temperature is as low as -14°C , while the standard Norsok says that -7°C is sufficient. It is important that the design temperature is conservative. If the ambient temperature goes below the design temperature the operating procedure states that the process system must be shut down. Should this occur, it would be a costly affair and it may decrease the integrity severely as the danger of cooling increases with a stagnant process.

6. Analytical Solution

To verify the obtained results in OpenFOAM they should be compared with existing theory, theoretic numerical models or similar experiments. The summer intern project in Aker Solution is not validated. Therefore, to validate the calculation sheet it is assumed that the dead leg may be regarded as a Fin and simulate the Fin-analysis in OpenFOAM. Another way to validate the result is to compare it to the analytical Fin-analysis. The mathematical approach given in Cengel [2006] for the Fin-analysis is for convenience presented here.

6.1. Fin-analysis, mathematical formulation

This section is cited from Cengel [2006]. The fin analysis is a steady one-dimensional problem, and it is usually presented similar to Figure 6.1, divided into several control volumes.

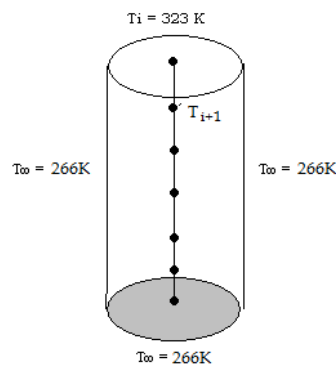


Figure 6.1: Layout of Fin-geometry

The energy balance can be expressed as,

$$\dot{Q}_{cond,x} = \dot{Q}_{cond,x+dx} + \dot{Q}_{conv} \quad (6.1)$$

where

$$\dot{Q}_{cond} = -kA \frac{dT}{dx} \quad (6.2)$$

$$\dot{Q}_{conv} = hP\Delta x(T - T_{inf}) \quad (6.3)$$

Inserting equation (6.3) into equation (6.1) and divide by Δx , the equation yield,

$$\frac{\dot{Q}_{cond,x+\Delta x} - \dot{Q}_{cond,x}}{\Delta x} + hP(T - T_{inf}) = 0 \quad (6.4)$$

Taking the limit $\Delta x \rightarrow 0$ gives,

$$\frac{d\dot{Q}_{cond}}{dx} + hP(T - T_{inf}) = 0$$

and inserting Fourier's law, equation (6.2), the equation becomes,

$$\frac{d}{dx}\left(kA \frac{dT}{dx}\right) - hP(T - T_{inf}) = 0 \quad (6.5)$$

Equation (6.5) is the final differential equation for the Fin. As the dead leg is of uniform perimeter and area, equation (6.5) may be rewritten to equation (6.6),

$$\frac{d^2\theta}{dx^2} - m^2\theta = 0 \quad (6.6)$$

where

$$m^2 = \frac{hP}{kA}$$

and

$$\theta = T - T_{inf}$$

The general solution is then derived from equation (6.6) resulting in equation (6.7),

$$\theta(x) = C_1 e^{mx} + C_2 e^{-mx} \quad (6.7)$$

where C_1 and C_2 are arbitrary constants determined from the boundary conditions. As the end of the dead leg usually is connected to a valve, it is assumed that it can be perceived as insulated. Accordingly the boundary condition at the end is expressed as,

$$\left. \frac{d\theta}{dx} \right|_{x=L} = 0$$

The connection between the boundary conditions at the tip and the base may then be used to determine the relation for the temperature distribution. The analytical solution is then found from equation (6.8),

$$\frac{T(x) - T_{inf}}{T_b - T_{inf}} = \frac{\cosh(m(L-x))}{\cosh(mL)} \quad (6.8)$$

where $T(x)$ relates to the temperature at location x , T_∞ is the ambient temperature, T_b is the bulk temperature in the medium, m is the square root of $\frac{hP}{kA}$ and L is the length of the fin.

If the fin tip is not assumed isolated the approach is to change the parameter L in equation(6.8) with a corrected fin length which is expressed as,

$$L_c = L + \frac{D}{4}$$

The summer intern project in Aker Solutions developed a mathematical script for solving this Fin problem numerically [Aker Solutions, 2011]. The solution strategy was to discretize equation (6.6) and create a tri-diagonal system so that the Thomas algorithm (TDMA) could be applied. After investigating there code, their result seems to be valid. The results also coincide with the exact solution retrieved from equation (6.8).

7. Simulation set-up

7.1. Development of the geometry

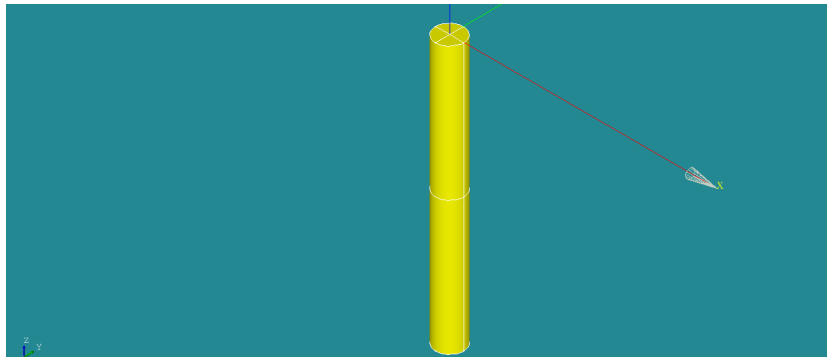
In OpenFOAM there is integrated a separate program called Salome. Salome is a 3D CAD program where both structure and mesh can be created. As described in the introduction, the first part of this thesis is to set up a simulation similar to the fin-analysis. Subsequently, a second model based on the rule of thumb is to be created. The industry addresses dead legs on whether they should be insulated or not. The rule of thumb implies that if the dead leg is at least 2" in diameter and no longer than 0.5 m, then no insulation is needed [Osenbroch, 2012]. The base geometry is therefore made in schedule (SCH.) 40, with a 2" dead leg and 4" main pipe, see table 7.1.

Table 7.1: Pipe size and wall thickness

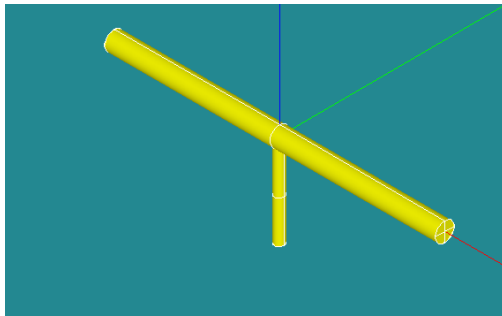
Pipe	Nominal size [in]	D_e [mm]	D_i [mm]	WT [mm]	SCH.
Main pipe (DN200)	4	114.3	102.26	6.02	40
Deadleg (DN100)	2	60.33	52.51	3.91	40

The geometry is shown in Figure 7.1, the length of the dead leg is set to 0.5m, which is about $9D_i$. To create the geometry in Figure 7.1(a), two vertexes and a vector line is created so that a solid cylinder could be developed. The geometry in Figure 7.1(b) and Figure 7.1(c) are created in the same manner. However, instead of one cylinder, two cylinders had to be created and then fused together. The geometry is then divided into groups which will define the patches, such as inlet, wall, outlet, dead leg wall and dead end.

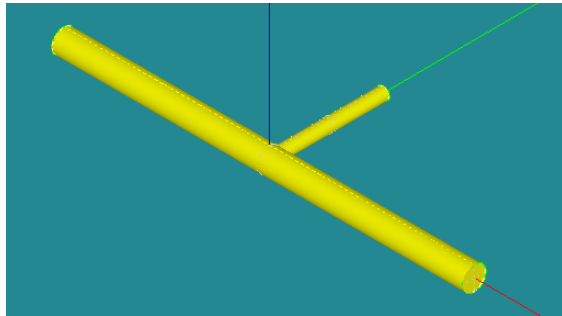
In addition to enable the geometry development, Salome provides several mesh possibilities. In this case a mesh of type, Netgen 1D-2D-3D is used and the cell distribution is determined from a grid independence test. The Netgen 1D-2D-3D is an unstructured triangular mesh. Therefore, the numerical stability must be maintained by the non-orthogonal corrector in OpenFOAM [Foundation, 2011]. Further the mesh is divided into groups which separates the boundary patches, see Table 7.2.



(a) Fin Layout



(b) T-junction vertical



(c) T-junction horizontal

Figure 7.1: Layout of the dead leg geometries are presented in (a)-(c)

Table 7.2: Defined boundaries on the 3D models

ITEM	DEFINED BOUNDARY PATCHES					
Part 1: FIN	Internal field	Inlet	Wall			Deadend
Part 2: T-junction	Internal field	Inlet	WallsM	WallsD	Outlet	Deadend

7.2. Boundary Conditions

The most important feature to run a CFD simulation is the initial boundary conditions. It is important to understand how they affect the whole simulation. In OpenFOAM the boundary conditions are separated into patches of different types [Foundation, 2011].

Base type, defines the type of patch in terms of the geometry.

Primitive type, defines a numerical condition for the assigned field variable.

Derived type, defines a complex patch condition, derived from the primitive type, and assigned to a field variable.

7.2. BOUNDARY CONDITIONS

Two base type boundaries are used in this thesis; patch and wall. The patch type contains no geometric or topological information about the mesh, e.g. the inlet and outlet. The wall base is defined to account for the wall boundaries and is used by turbulence models to compute normal wall distance [Hjertager, 2009]. The primitive types of boundary specifications are set for different variables and the derived boundary type is set on the wall and derived from the primitive boundary type [Foundation, 2011].

In the initial boundary condition file there is defined one internal field in addition to the defined boundary patches in Table 7.2. For convenience the two simulations are referred to as part 1, Fin-analysis, and part 2, the T-junction. For every boundary there is defined a boundary condition of primitive or derived type. In OpenFOAM there is a range of boundary conditions [Foundation, 2011]. Table 7.3, 7.4 and 7.5 summarize the boundary conditions for the Fin, the T-junction and the turbulence variables respectively.

Table 7.3: Defined boundaries in the Fin-analysis

Boundary	Boundary type, T	Explanation
Internal field	uniform value	Sets a uniform value in the internal field
Inlet	fixedValue	A specific value is given
Wall	GroovyBC	Based on mixed BC, and an algorithm is set to solve for the surface temperature.
Outlet	zeroGradient	Sets the normal gradient to zero, assuming that the end is insulated.

Table 7.4: Defined boundaries in the T-junction

Boundary	Boundary type				
	T	U	p	p_{rgh}	κ_t
Inlet	fixedValue	fixedValue	zeroGradient	buoyantPressure	fixedValue
wallsM	groovyBC	fixedValue	zeroGradient	buoyantPressure	kappaJaya WallFunction
Outlet	inletOutlet	inletOutlet	fixedValue	fixedValue	inletOutlet
wallsD	groovyBC	fixedValue	zeroGradient	buoyantPressure	kappaJaya WallFunction
Deadend	zeroGradient	fixedValue	zeroGradient	buoyantPressure	kappaJaya WallFunction

Part 1, Fin-analysis is purely related to the temperature. Hence only temperature boundaries are defined. The boundary types are presented shortly in Table 7.3. The initial conditions are presented in Table B.7 in Appendix B. In part 2, T-junction the temperature conditions will be the same as the latter case. However, additional boundary patches will be introduced. Additionally, Part 2 introduces flow and turbulence into the process.

Table 7.5: Defined boundaries for the turbulence variables

Boundary	Boundaries for the Turbulence Variables		
	k	ϵ	ν_t
Inlet	turbulentIntensity-KineticEnergyInlet	turbulentMixingLength-DissipationRateInlet	nutkWallFunction
Outlet	inletOutlet	inletOutlet	inletOutlet
wallsM	kqRWallFunction	epsilonWallFunction	nutkWallFunction
WallsD	kqRWallFunction	epsilonWallFunction	nutkWallFunction
Deadend	kqRWallFunction	epsilonWallFunction	nutkWallFunction

Hence the amount of variables is increased as implied in Table 7.4 and Table 7.5. All inlet conditions are defined in Appendix B and they are calculated from the attached Matlab script in Appendix F. In the next sections the defined boundary conditions will be explained in more detail for each boundary.

7.2.1. Inlet boundary conditions

For part 1 Table 7.3 indicates that the inlet boundary condition is defined as a fixed value. Assuming that the fluid in the dead leg is stagnant, holding a constant inlet temperature. This is due to an assumption that the inlet is connected to a main pipe holding a constant medium temperature.

$$T_{\text{inlet}} = 50^{\circ}\text{C}$$

Part 2 is influenced by several other variables as Table 7.4 imply. The inlet is now located in the main pipe not in the dead leg inlet. However, the temperature at the inlet is set similar to part 1.

The velocity is calculated from equation (2.13), assuming a liquid column of 0.418 m. As the velocity is assumed constant the boundary condition is set to fixedValue.

$$U = \text{uniform}(2.86 \ 0 \ 0)$$

The way the velocity is presented indicates that there is no velocity gradient in y- and z-direction as it is assumed a nearly fully developed flow.

Hjertager [2009] imply that when the velocity is fixed at the inlet the boundary condition for pressure may be set to *zeroGradient*, which allows the actual value on the boundary to float.

Furthermore, the inlet boundary conditions for turbulence are set as in Table 7.5. To obtain the most accurate simulation the values of turbulent energy, k , dissipation rate, ϵ , and eddy viscosity, ν_t , should be applied from measured data. As there exist no such

7.2. BOUNDARY CONDITIONS

measurements, the values are obtained from approximated correlations, such as equation (3.7), (3.8) and (3.3).

$$k = 0.0197 \quad \varepsilon = 0.0634 \quad \nu_t = 5.5 \times 10^{-4}$$

The boundary conditions assigned to these variables are; *turbulentIntensityKineticEnergyInlet* for k which needs a defined turbulent intensity, T_i , in addition to the initial value,

$$T_i = 0.06 = 6\%$$

turbulentMixingLengthDissipationRateInlet for ε which needs a defined mixing length in addition to the initial value,

$$\ell = 0.07L = 0.00714 \text{ -- where } L \text{ is the internal pipe diameter}$$

and the turbulence viscosity, ν_t , is defined by the *calculated* boundary condition which means that the boundary value will be calculated from other fields [Hjertager, 2009].

In addition to these variables, there are two more. That is the dynamic pressure, p_{rgh} , and the heat transfer diffusivity, κ_{eff} , which is defined by boundary condition *buoyantPressure* and *fixedValue* respectively. The buoyant pressure boundary condition sets a fixed gradient pressure based on the atmospheric pressure gradient [Hjertager, 2009], which is similar to the *zeroGradient* boundary condition. However, it needs a defined density field, which is the effective density from equation (4.11).

$$\rho_k = 1.0$$

7.2.2. Outlet boundary conditions

For part 1 the temperature condition at the outlet is only set to *zeroGradient*, as this is an insulated end with no heat loss.

In part 2, nearly all outlet boundary conditions are set to *inletOutlet*. The *inletOutlet* boundary condition behave as a *fixedValue* if subjected to an inwards flow and similar to *zeroGradient* when subjected to an outwards flow [Hjertager, 2009]. The only variables which are not defined by this boundary condition are the pressure and dynamic pressure. Both of these are given the *fixedValue* boundary condition at a value of uniform zero.

7.2.3. Wall boundary conditions

The wall boundaries are the most common boundary, and it appear as a fixed boundary, initializing the fluid from the environment. For temperature on the walls the boundary of

part 1 and 2 is defined by a boundary condition called GroovyBC. This boundary condition develops a mixed-BC condition. The fields are defined by value, variables, valueExpression and fractionExpression. It is the fractionExpression that specify the boundary expression to be solved. It can be used to set non-uniform boundary-conditions without programming [Gschaider, 2012]. The approach for expressing the heat transfer came from the CFD discussion forum [Hjertager et al., 2012].

```
wall
{
  type          groovyBC
  variables     "h_tot = 48.39; T_inf = 266; rho = 998.5; c_p = 4185.0; k = DT * rho * c_p;"
  valueExpression "T_inf"
  value        uniform323
  fractionExpression "1.0/(1.0 + k/(mag(Delta()) * h_tot))"
}
```

Note that for part 2, DT is exchanged by κ_{eff} .

The theory states that heat transfer from a cylinder is expressed by conduction and convection. Therefore, the overall heat transfer coefficient, h_{tot} , is calculated in accordance to equation (2.11) for the main pipe (M) and the dead leg (D) respectively,

$$h_{tot,M} = 25.43\text{W/m}^2\text{K} \quad h_{tot,D} = 48.39\text{W/m}^2\text{K} \rightarrow u_a = 9\text{m/s}$$

$$h_{tot,M} = 13.36\text{W/m}^2\text{K} \quad h_{tot,D} = 25.43\text{W/m}^2\text{K} \rightarrow u_a = 3\text{m/s}$$

The fraction expression is actually solving the face temperature of the cell, developed from the standard heat balance equation (7.1) in accordance to the CFD discussion forum [Hjertager et al., 2012],

$$k \frac{dT}{dn} + h\Delta T = 0 \quad (7.1)$$

When expressing the wall boundary for temperature in the latter method an assumption is that the effect of wall conduction is relatively small compared to the heat loss in the medium, which also is stated by [Hong, 1977].

For the other variables such as velocity and pressure, more standard boundary conditions such as *fixedValue* and *zeroGradient* are used respectively. In accordance to the given rule when combining velocity and pressure boundaries, it is stated that as the velocity is fixed, the pressure should be *zeroGradient* [Hjertager, 2009]. It is assumed no slip on the walls, hence the velocity is fixed to zero.

For the turbulence parameters such a k , ε and ν_t specified wall functions are more suitable. The turbulent dissipation rate, ε is assigned the *epsilonWallFunction*, the turbulent kinetic energy, k , is assigned the *kqRWallFunction* and the eddy viscosity, ν_t , is assigned the *nutkWallFunction* as implied in Table 7.5 [Hjertager, 2009]. The boundary condition for turbulent kinetic energy solves equation (7.2) for each iteration step,

$$k \equiv \frac{1}{2} \overline{u'_i \cdot u'_i} = \frac{1}{2} (U_x'^2 + U_y'^2 + U_z'^2) \quad (7.2)$$

where $\overline{u'_i} = U'_i$ is the fluctuating velocity in the x, y and z direction respectively. The defined boundary condition for ε solves equation 7.3 for each iteration step,

$$\varepsilon = \frac{C_\mu^{0.75} k^{1.5}}{\ell} \quad (7.3)$$

while the turbulent viscosity, ν_t , is derived from equation (3.3). The wall functions are derived in Voigt [2001] if a further investigation is desired.

All initial conditions are calculated from the Matlab script in Appendix F and the values are presented in Table B.1-B.7 in the appendix B.

7.3. External environment

From NORSOK standard L-002 [2009] the external design temperature of pipes is given as -7°C (266K). The external temperature influence the heat transfer in the pipe, however, the wind is not mentioned in this standard. Therefore, in addition to the temperature a wind velocity is set and as this influences the local heat transfer coefficient on the outside it is of interest to see how much it influences the cooling rate.

The following outside environment conditions are set,

Table 7.6: Environmental conditions

Ambient temperature [K]	Wind velocity [m/s]
266	9
266	3

7.4. Numerical Accuracy

The numerical accuracy in a CFD simulation is the basis to obtain high quality results. In general it should always be performed a grid independence test when using a discrete

method, the closer to zero the cell size becomes, the better accuracy will be achieved. Furthermore, residuals may be investigated to make sure that the target quantity is independent on the convergence criterion and mesh [Versteeg and Malalasekera, 2007]. OpenFOAM provides the tools to create the grid distribution and it enables the possibility to easily change the grid density. The grid independence test is performed with several grid sizes, from coarse to very fine as may be seen in Table 7.7.

Table 7.7: Number of cells (N)

Grid no.	1	2	3	4
N	170 000	240 000	280 000	310 000
Number of iterations	1306	1417	1610	1503

The results from the grid independence test should follow the same tendency. From the converged result, the coarsest grid distribution which follows the result tendency should be chosen. This is because of the high cost of computer space and the time it takes to obtain convergence in very fine grid distributions [Versteeg and Malalasekera, 2007].

In Figure 7.2 the grid distribution is presented for the main pipe inlet and along the x -plane of the geometry. As can be seen it is unstructured and much finer at the dead leg than the larger main pipe.

From the four mesh refinement the converged temperature, velocity and turbulent kinetic energy are plotted in Figure 7.3, 7.4 and 7.5 from the center of the main pipe through the end of the dead leg in the z -axis.

The results show that the three variables follow the same tendency for the three finest grid distributions. As the temperature is the key quantity, the result here is the most important. Figure 7.3 implies that there is some changes in the end of the plot. However, the two finest grid distributions seems to follow the same tendency at a sufficient manner. Hence it is assumed that a grid with $N \approx 280\,000$ cell is sufficient to hold the accuracy of the solution.

As the simulation is introduced to turbulence, a wall refinement test is performed due to the dependencies between the wall coordinates and the results obtained from a turbulence model. The wall coordinate, y^+ , has been explained in section 3.2. Figure 7.6 show that the wall coordinates is well below the limit of 50 which is the constraint of the k - ϵ model. In addition the values are within the buffer-layer which is sufficient in regards to the logarithmic law of the wall and the inner law.

The sufficient mesh distribution should then be tested individually to see that the residuals of the key quantity converge within the given tolerance. Residuals are explained in section 4.4.2. For the temperature and velocity the tolerance is set to 1×10^{-5} with a given relative tolerance of 0.1. Plotting the residuals in *xmgrace* for the key quantity, temperature, the result imply that the residuals converge just slightly below the given tolerance as shown in Figure 7.7.

As mentioned previously there is an additional corrector which must be set. That is the non-orthogonal corrector in the *fvSolution* file in OpenFOAM. As the grid is highly unstructured this corrector is set to a maximum value of 20 [Foundation, 2011]. The effect of this corrector has been investigated in the potential flow case, and the result showed that the stability of the velocity became much better after increasing the corrector to 20. This contributes to increase the overall accuracy of the solution.

If a transient solver of flow is executed there is another important parameter which enables stability. Namely the Courant number, expressed in equation (7.4). The Courant number should always be kept below 1. In the beginning of a transient simulation it may be efficient to hold the Courant number even lower to ensure stability.

$$C_o = \frac{U\Delta t}{\Delta x} < 1 \quad (7.4)$$

where U is the velocity magnitude, Δt is the time step and Δx is the size of the cell. As the Courant number is greatly influenced by the time step, OpenFOAM provides a function of adjustable time step. So if a transient analysis is desired this makes it easy to obtain the stability needed.

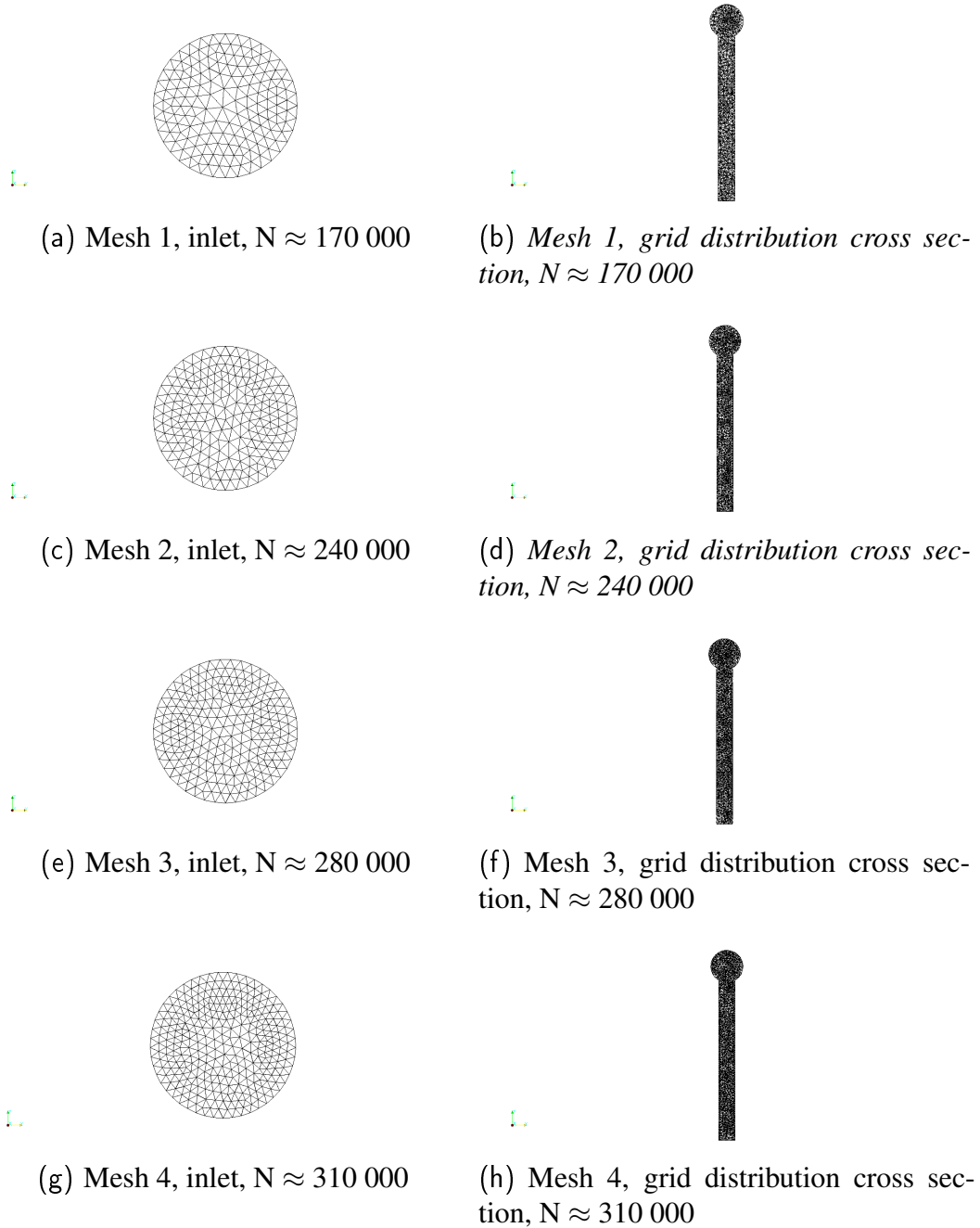


Figure 7.2: Unstructured triangular mesh - Netgen 1D-2D-3D presented for main pipe inlet and along the x-axis for the four mesh distributions

7.4. NUMERICAL ACCURACY

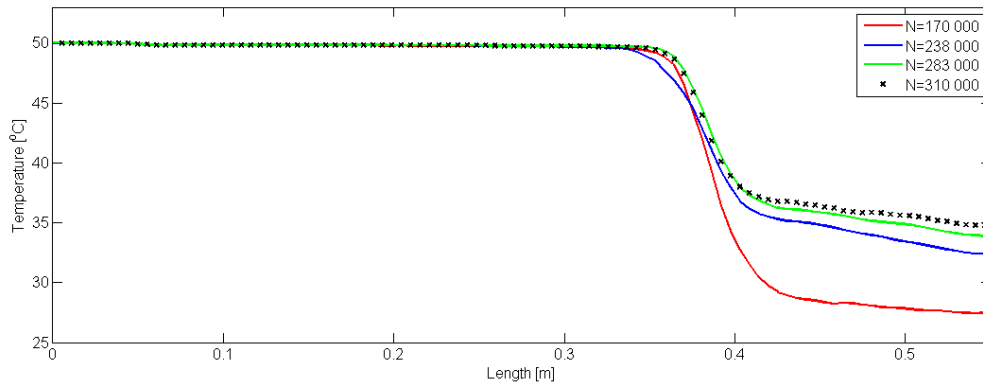


Figure 7.3: Comparison of final temperature profile for mesh 1-4

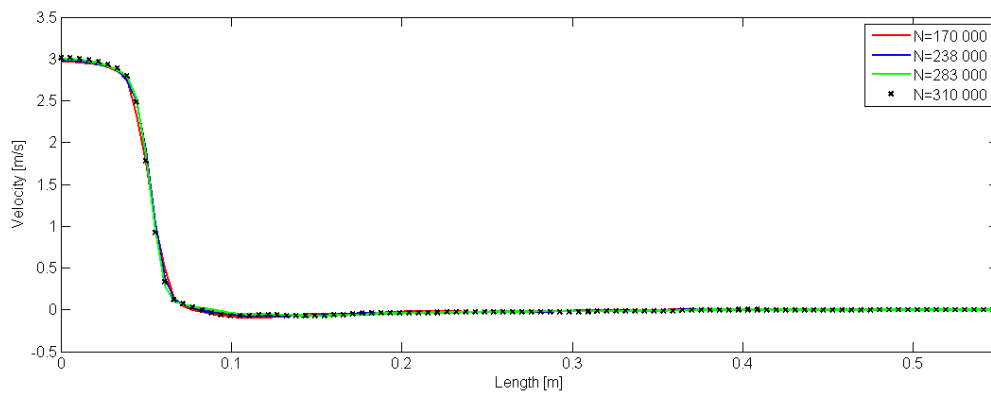


Figure 7.4: Comparison of final velocity distribution for mesh 1-4

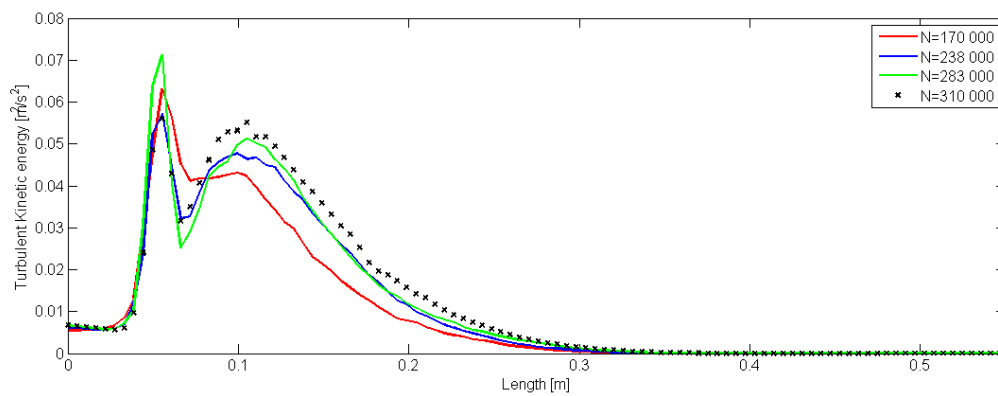


Figure 7.5: Comparison of final turbulent kinetic energy distribution for mesh 1-4

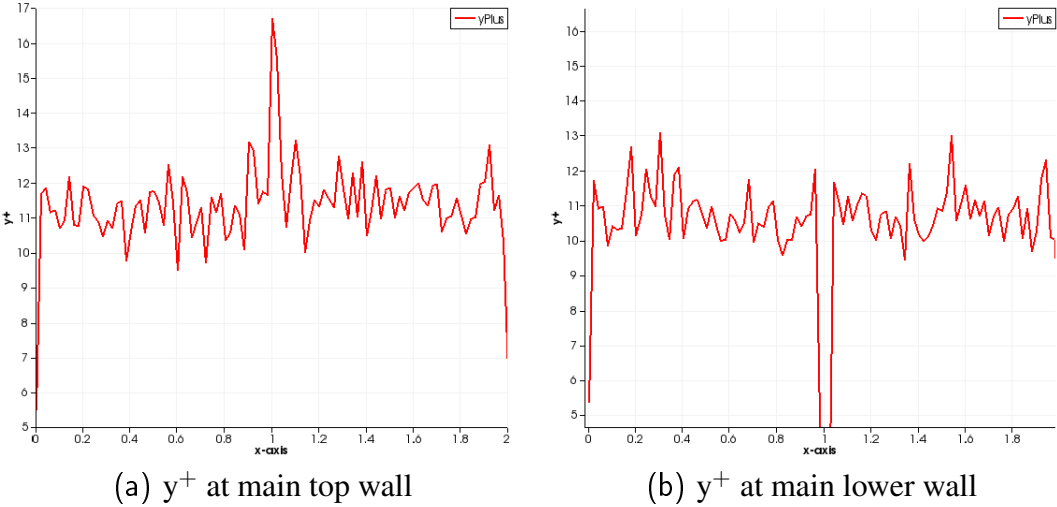


Figure 7.6: Plot of the wall coordinate, y^+ , for the main pipe

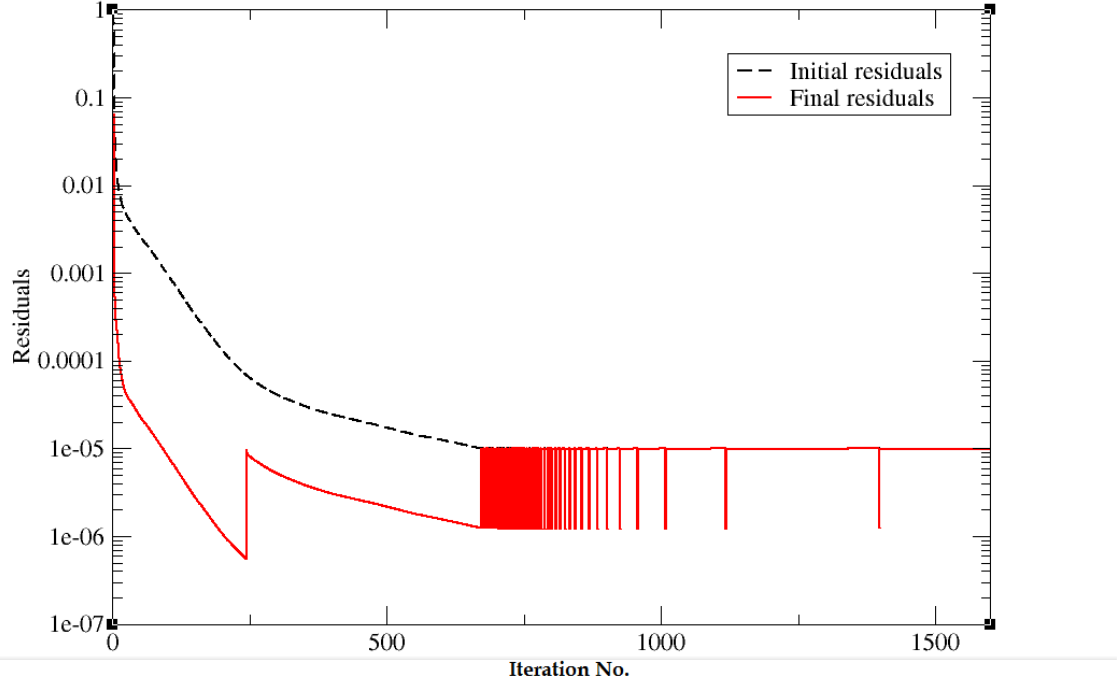


Figure 7.7: Residuals of Mesh 3, $N \approx 280\ 000$

7.5. Case description

Several separated cases of the problem are investigated. The first case adapt the industrial Fin-analysis, the second case introduces the flow field in the heat transfer study and obtain a converged temperature field under normal working conditions and the third case simulate a shut down from the converged temperature field obtained from normal working conditions.

The cases which are investigated in this report are presented in Table 7.8 and a more detailed description is given in the following text.

Table 7.8: Simulated Cases

Case no.	Case description	Solver	Time Scheme
01	Fin-analysis	LaplacianFoam	Steady&Transient
a)	Wind 9m/s		
b)	Wind 3m/s		
02	Normal working conditions		Steady
a)	Develop a potential flow field	PotentialFoam	
b)	Heat transfer&flow analysis	BuoyantBoussinesq-SimpleFoam	
03	Shut down after normal working conditions	LaplacianFoam	Steady&Transient

Case 01 – Fin analysis, use the base geometry of the Fin, Figure 7.1(a), and adapt a solver called LaplacianFoam to solve the diffusion problem for stagnant water. The initial boundary conditions for temperature are implemented such as in Table B.7. To adapt the Fin-analysis several assumptions are implemented.

- The initial temperature is 50°C for the whole inlet area and held constant.
- Natural convection inside the pipe is negligible.
- The liquid in the dead leg is stagnated.
- The pipe is exposed to air temperature at -7°C with a wind speed of 9m/s and 3m/s, resulting in a turbulent airflow around the cylinder.

- The cylinder is exposed to mixed boundary conditions.

The total heat transfer coefficient is dependent on the wind velocity. Thus two cases of wind velocities, case 01 a) and b), are simulated to see how much the wind influences the temperature profile. Both steady and transient simulations are executed.

Case 02 – Normal working conditions, introduces flow and turbulence into the main pipe during the heat transfer study. It is the base model in Figure 7.1(b) that is used. However, to create a stable model the execution is set in two steps. As Table 7.8 implies the first part is Case 02a. This is a potential flow solver which creates the potential flow field over the domain. The flow field which is developed is almost fully developed, and this increase the stability of the more advanced solver which is set in case 02 b). Transferring the velocity field from a) to the initial velocity file in b) and incorporate turbulence, the buoyantBoussinesqSimpleFoam solver may be executed. This simulation will eventually converge at the steady solution, which means that there will be no change in time after this point. Hence the results implies how the heat transfer will behave under normal working conditions.

Case 03 - Shut down after normal working conditions, simulate a shut down by inserting the developed temperature field from Case 02 b). This case adapt the same solutions strategy as Case 01, but the geometry is the same as in Case 02, Figure 7.1(b). This simulation creates a more realistic picture of the dead leg connection and how it behaves when it is exposed to a shut down after working under normal conditions.

8. Result and Discussion

The main objective of this thesis is to investigate the method used in the industry today, and to then simulate a "real life" situation for a process system. The model which is developed is easy to adapt for different fluids on topside dead legs. Even different geometries other than base model is easy to implement.

The case set up is presented in Table 7.8 and all the case files are included in Appendix I. A procedure on how to run the simulation cases are attached in Appendix H.

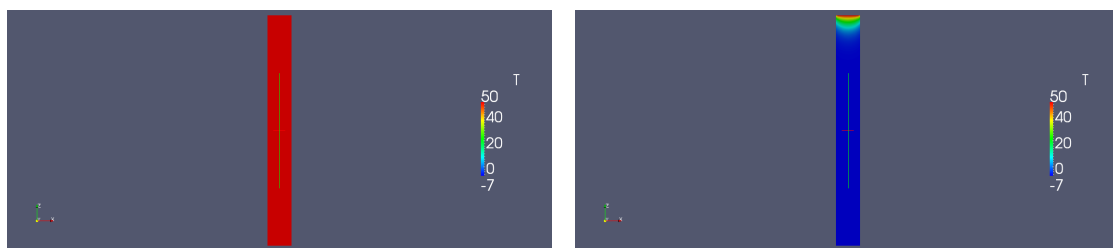
In this chapter the result of the different cases will be presented. The result will further be compared to existing theory and experimental research to verify the results obtained from the simulation. In addition, discussions regarding the results will be performed along the way.

8.1. Case 01.a – Fin-analysis, wind velocity 9m/s

Contour of the temperature profile is presented in Figure 8.1(a) and Figure 8.1(b). The heat transfer coefficient in this case is,

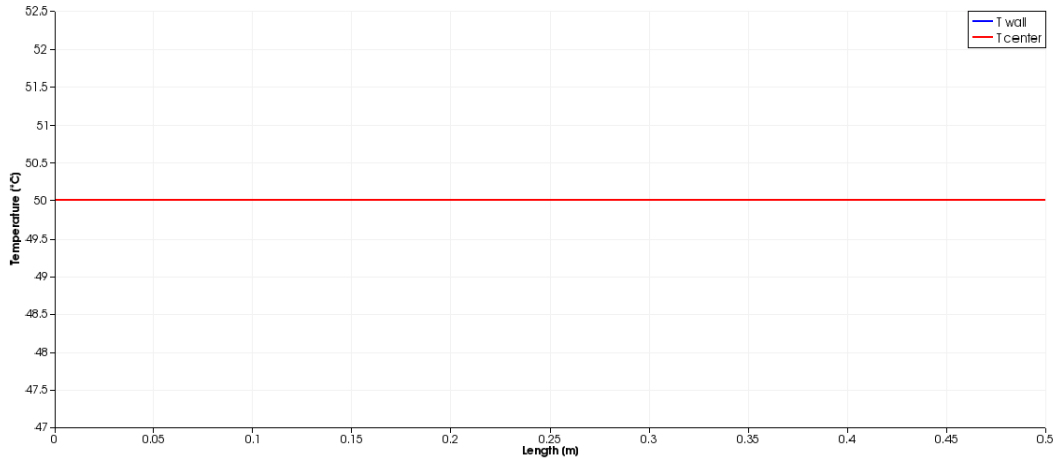
$$h_{tot} = 48.39 \quad [\text{W}/\text{m}^2\text{K}]$$

To better see the temperature distribution, a center and wall line, in z-direction, through

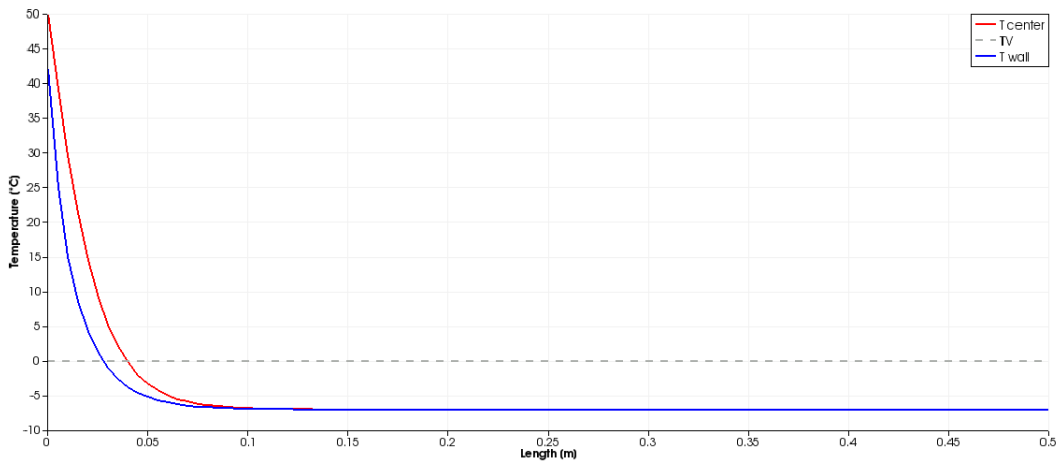


(a) Contour of the initial temperature distribution (b) Contour of the Final temperature distribution

Figure 8.1: Contour of the steady state fin analysis



(a) Schematic illustration of the initial temperature distribution



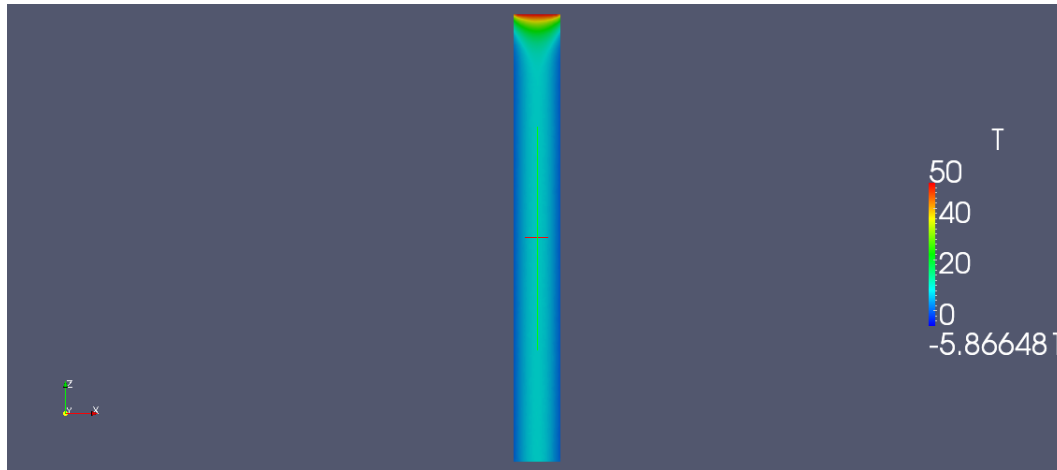
(b) Schematic illustration of the Final temperature distribution

Figure 8.2: Initial and final plot of the temperature distribution in Case 01a

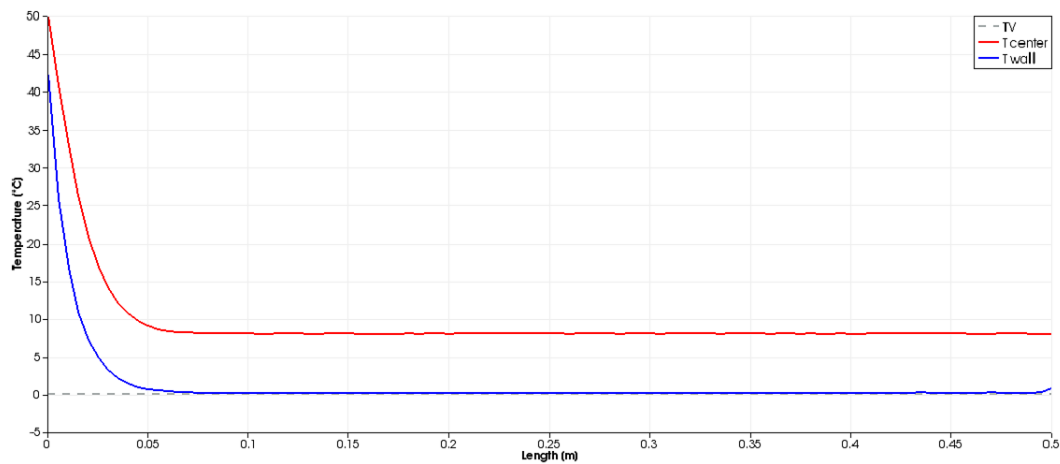
the dead leg is plotted in Figure 8.2. The initial temperature is uniform at 50°C . However, as the cooling process proceeds, the temperature reaches a converged state.

The steady-state results in Figure 8.2(b) shows the converged state of the temperature, for both wall and center. It may be seen that the threshold value (TV) is reached as close as 0.03m and 0.04m from the inlet, for the wall and center line respectively. A steady state only shows the final state if left unattended. Therefore, the transient analysis is important to understand when the TV occurs. In OpenFOAM, the transient simulation will not only tell how long it takes for the liquid to go below threshold value (TV), it also gives the temperature contour of the cooled liquid which is observed. This may be seen in Figure 8.3.

The first sign of water below TV is observed after $2800\text{s} \approx 46\text{min } 39\text{s}$. The contour and



(a) First sign of water reaching TV at 2800s.



(b) Schematic illustration of water reaching TV at 2800s.

Figure 8.3: First sign of water below zero degrees with wind velocity 9m/s

plot over dead leg wall and center line is presented in Figure 8.3. The contour in Figure 8.3(a) show that water is cooled evenly at the wall.

From the standard cooling time calculations given in ISO 12241 [2008] and equation (5.9) the pipe should reach TV after

$$t \approx 45\text{min}42\text{s}$$

This is in accordance with the results observed in OpenFOAM. However, from the plot in Figure 8.3(b) it is observed that only the wall is below the TV. The temperature change in the centreline is consequently obtained from probe installations as mentioned in section 4.6. The simulation shows the temperature with respect to time for different locations, the results are plotted in Figure 8.4.

The measurements show that the temperature reaches the TV after $4120\text{s} \approx 1\text{h } 40\text{min}$, at $1.5D_i \approx 0.078\text{m}$ from the dead leg inlet. This is consistent to the contour in Figure 8.5 as it can be seen that the temperature is uniformly decreasing on the wall towards the centreline. Note that the temperature scale has been reduced to account for the differences in the low temperatures.

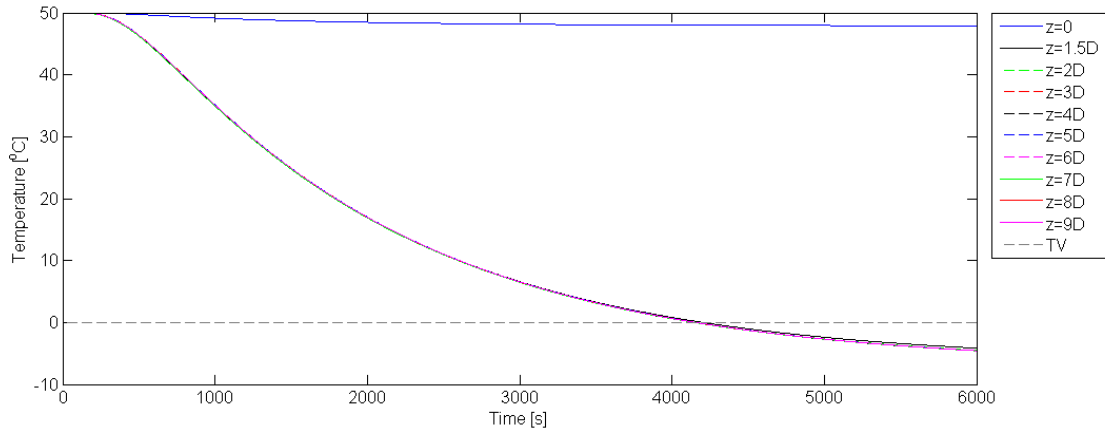
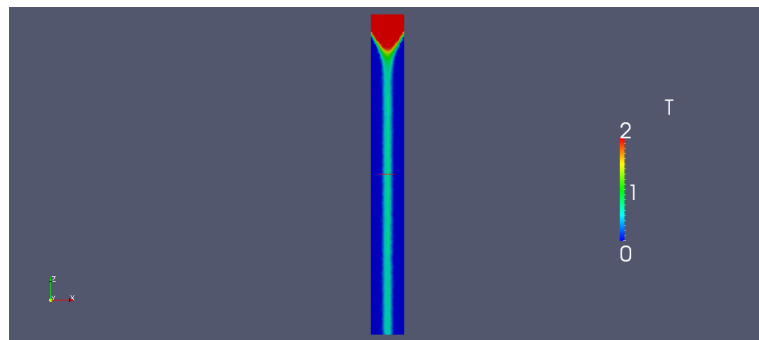
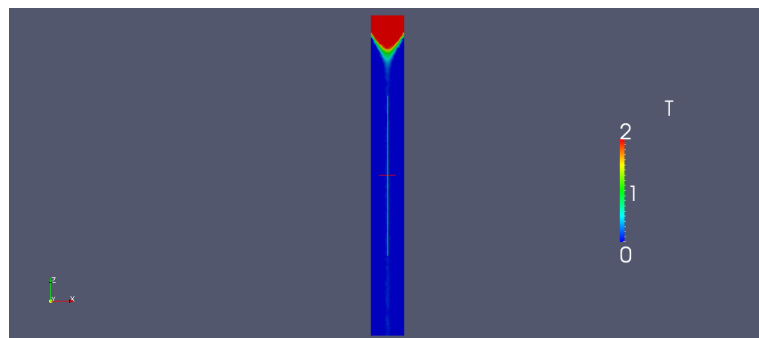


Figure 8.4: Probes measurements for Case 01.a



(a) Temperature contour after 4000 s



(b) Temperature contour after 4120 s

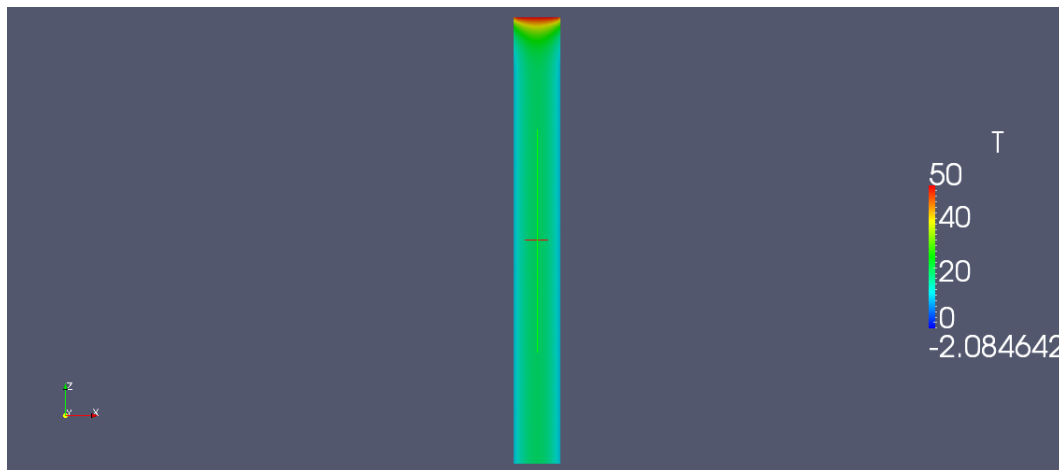
Figure 8.5: Temperature contour when reaching TV at center line at $\approx 4000\text{s}$

8.1.1. Case 01.b – Fin analysis, wind velocity 3m/s

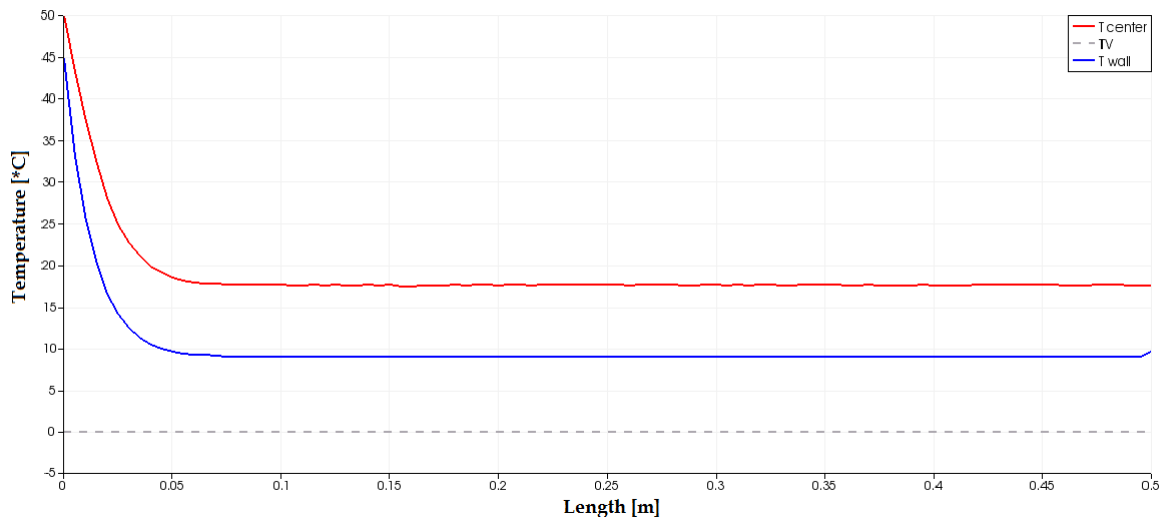
In this case the heat transfer coefficient is calculated to be,

$$h_{tot} = 25.44 \quad [\text{W}/\text{m}^2\text{K}]$$

which is lower than for case 01a. This is expected due to the lower wind velocity. The contour of the temperature and the schematic plot along the center-line and wall are given in Figure 8.6 and Figure 8.7 at time 2800s and 5000s respectively.



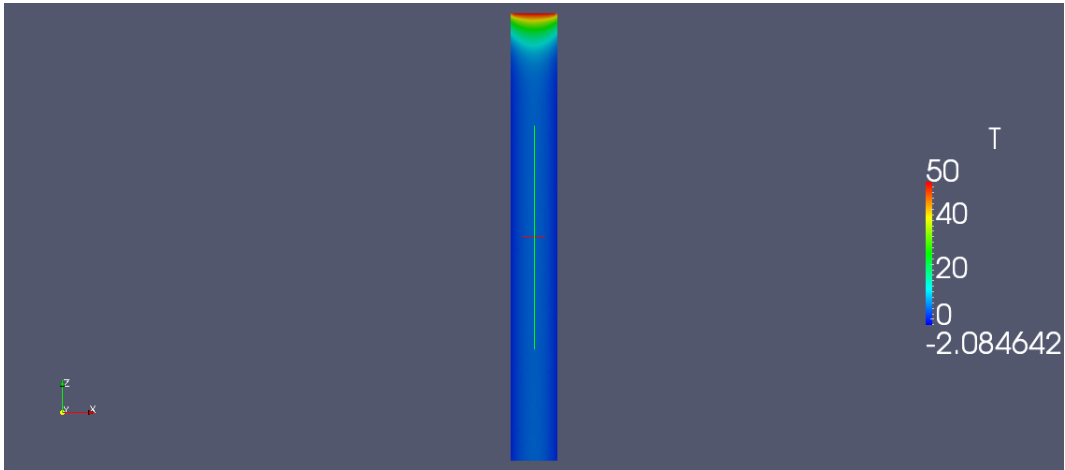
(a) Temperature contour after 2800s.



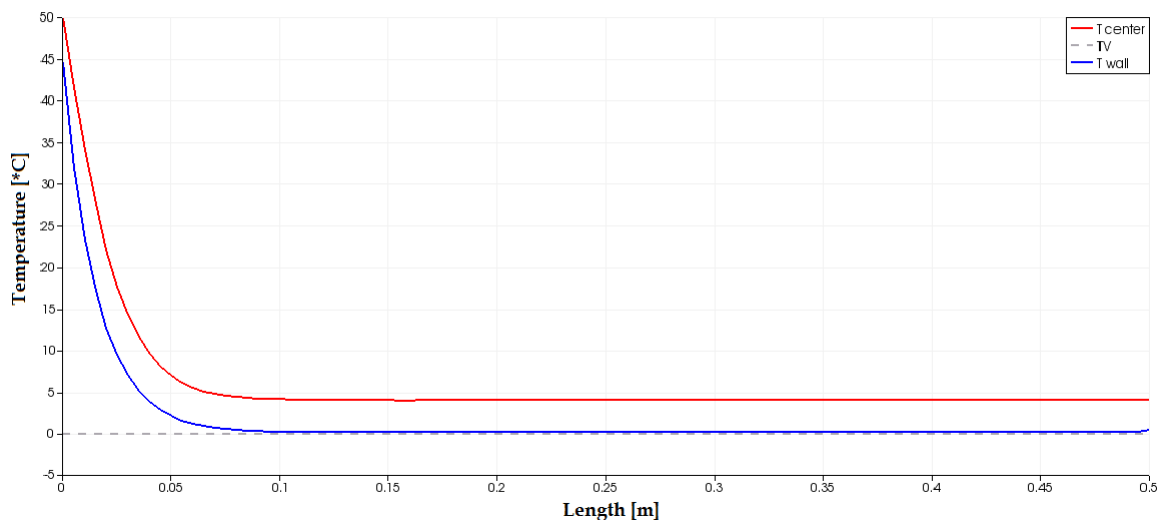
(b) Schematic illustration of the temperature distribution after 2800s.

Figure 8.6: Temperature profile with wind velocity 3m/s after 2800 s

Figure 8.6 shows that there is no signs of freezing within the 2800s as opposed to case with wind velocity at 9 m/s. The temperature stays above TV until it reaches around 5000s as Figure 8.7, and it is only the walls that have reached TV during this time. The center line of the pipe is not cooled below the TV during the execution time when the wind velocity is 3m/s. This is illustrated by the probe measurements in Figure 8.8.



(a) Temperature contour after 5000s



(b) Schematic illustration of the first sign of TV after 5000s.

Figure 8.7: Temperature profile with wind velocity 3m/s after 5000s.

8.2. EFFECT OF WIND VELOCITY

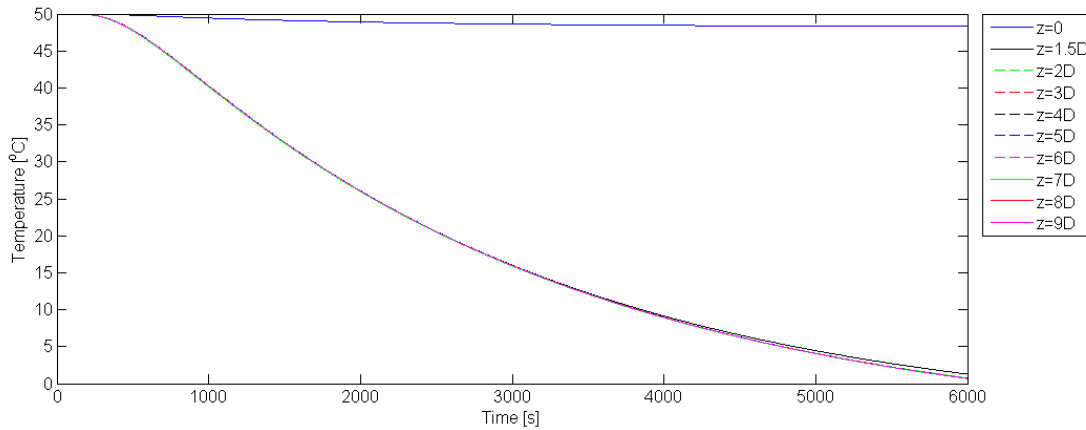


Figure 8.8: Probe measurements for Case 01.b

8.2. Effect of wind velocity

The effect of the wind velocity is observed by comparing the temperature profiles through the dead leg after 3600s for Case 01.a and 01.b. Figure 8.9 and Figure 8.10 shows the temperature profiles at the centre and wall, respectively. Additionally, the probe measurements in Figure 8.11 show that the time to reach TV is increased by a decrease in wind velocity. More specifically, when the wind velocity is reduced by a factor of 3, the time for the water to reach the TV is increased by approximately 34min. Hence, this result implies that the temperature development is dependent on the heat transfer coefficient. Note that since the temperature loss is constant from the probes at $2D_i$ to $9D_i$ only one probe location needs to be presented. Hence probe 5D.

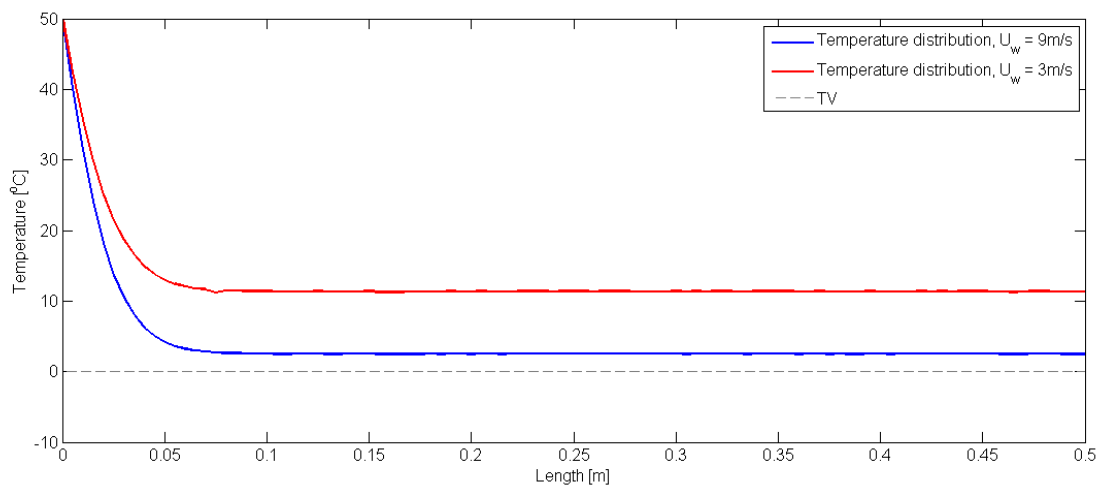


Figure 8.9: Effect of wind velocity in the center of dead leg after ≈ 1 h

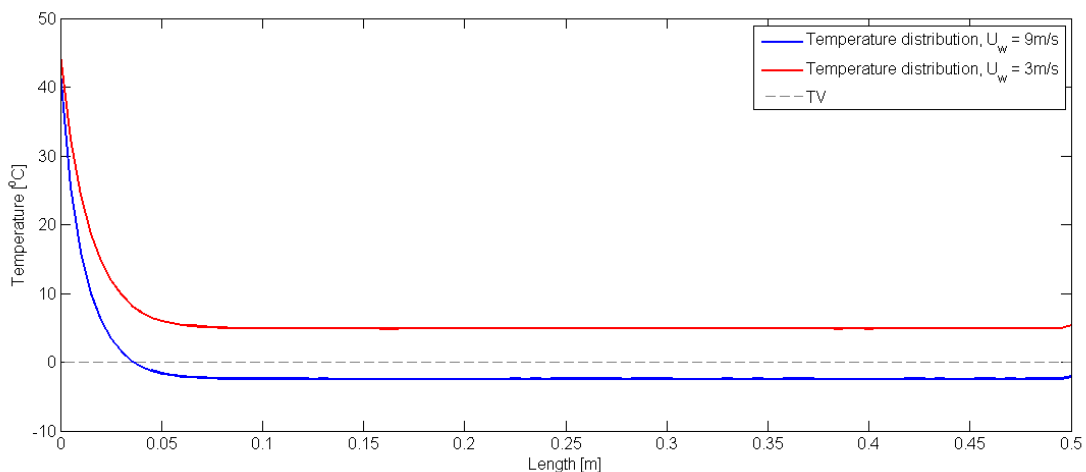


Figure 8.10: Effect of wind velocity at the wall of dead leg after ≈ 1 h

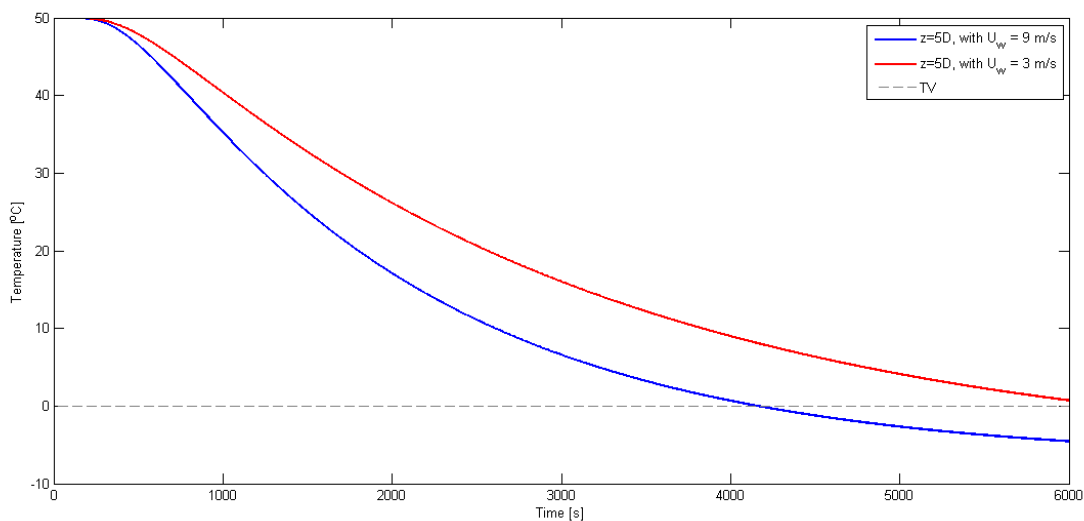
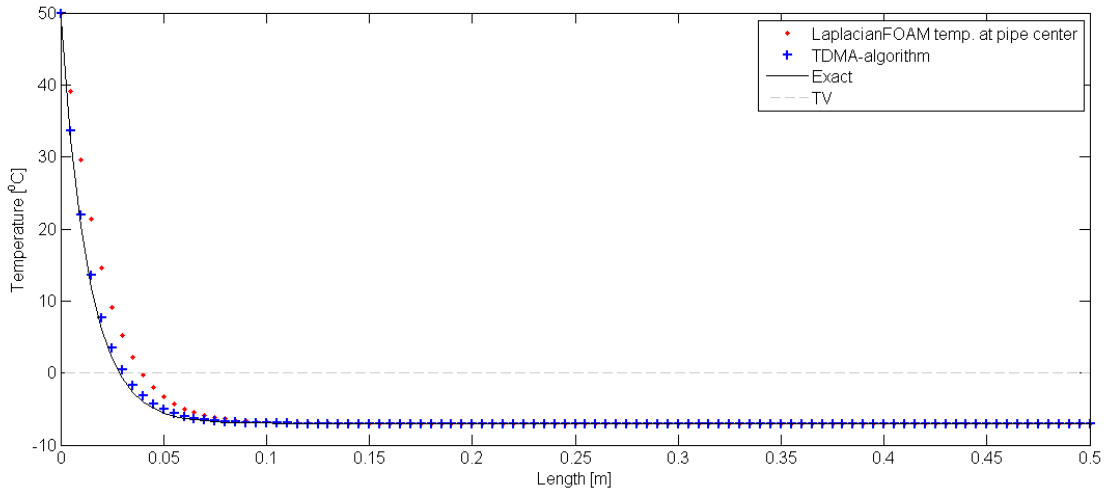


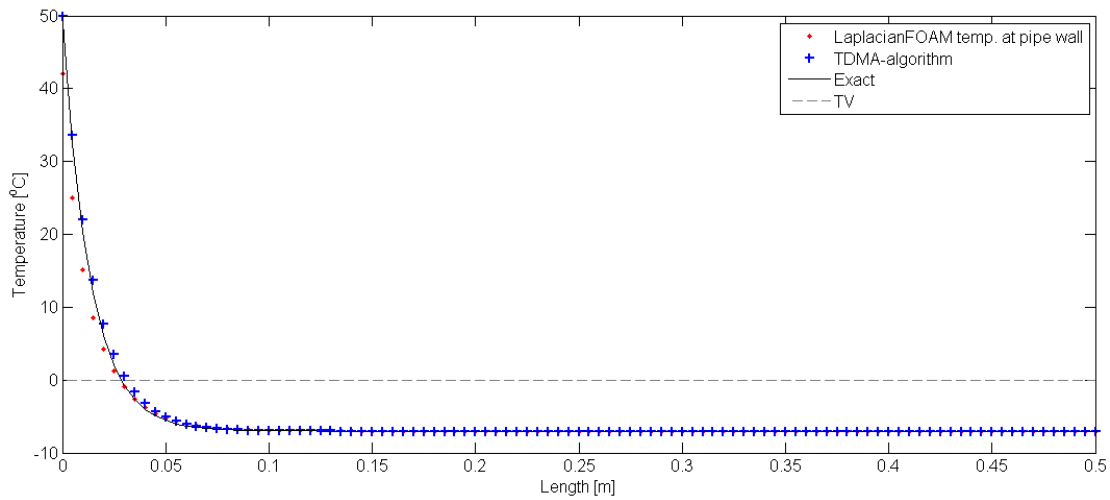
Figure 8.11: Probe measurements at $5D_i$ for case 01a and 01b

8.3. Verification and discussion of Case 01

To verify the latter case method, the result is compared to both the exact analytical solution for a Fin with neglected end [Cengel, 2006], and the numerical solution developed with the TDMA [Aker Solutions, 2011]. Figure 8.12(a) and Figure 8.12(b) present the center- and wall-line with the analytical solution respectively. The results are consistent.



(a) Comparison of the analytical solution, numerical and the center line of the Laplace simulation (Case 01)



(b) Comparison of the analytical solution, numerical and at the wall of the Laplace simulation (Case 01)

Figure 8.12: Comparison of the analytical solution, numerical and the Laplace simulation

However, in Figure 8.12(a) it seems like the temperature does not reach the TV at the same length as the analytical solution, this indicates the 3D effect of the CFD simulation. In addition, Figure 8.12(b) shows that the temperature at the wall starts at 45 °C and not 50°C. This may be due to the interface between the wall and the inlet. Subsequently, the wall boundary conditions initiate a temperature loss also at the interface. Figure 8.13 is a schematic explanation of this phenomenon.

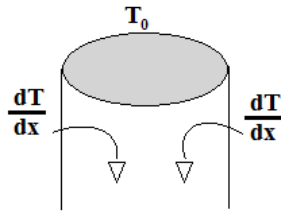


Figure 8.13: Schematic explanation of the interface boundary conditions

8.4. Case 02 – Normal working conditions

When creating a realistic model it is important to make it as stable as possible to ensure reasonable results. Therefore, the case is divided into two parts, 02a and 02b. The first part will create a fully developed velocity field to increase the quality of the initial velocity conditions. The second part will execute the case file and investigate the circulation and heat transfer in the dead leg.

8.4.1. Case 02.a – Develop a potential flow field

This 1st part uses the potential flow solver to create a fully developed flow field. The simulation was executed for several values of the *NonOrthogonalCorrector*. The result showed that the flow did not get fully developed until the corrector was set to the maximum number of 20. The velocity field is presented in Figure 8.14 and Figure 8.15. The result shows a fully developed flow.

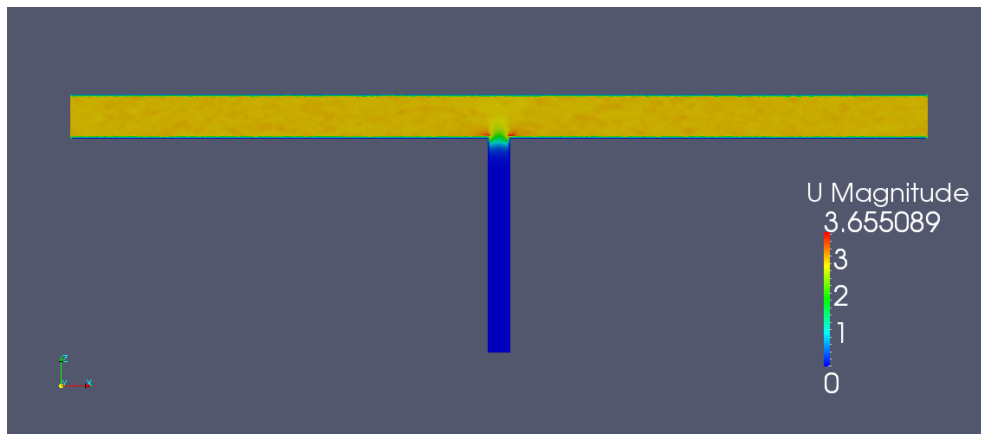


Figure 8.14: Contour of the velocity field from potential flow theory

8.4. CASE 02 – NORMAL WORKING CONDITIONS

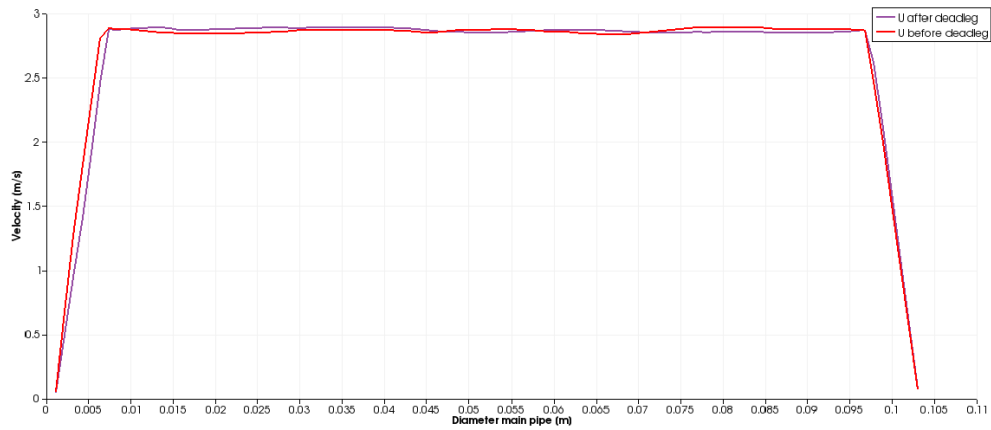


Figure 8.15: Velocity profile developed from potential flow theory

8.4.2. Case 02.b - Heat transfer&flow analysis

The 2nd part of this case executes the buoyantBoussinesqSimpleFoam solver. The results show the temperature in the pipe structure, in addition to the flow through the main pipe and the circulation of fluid in the dead leg. As the solver is a steady-state solver it will converge to a final solution. This implies that the converged state will be stable as long as normal working conditions are preserved. Figure 8.16 shows the contour of the final temperature field.

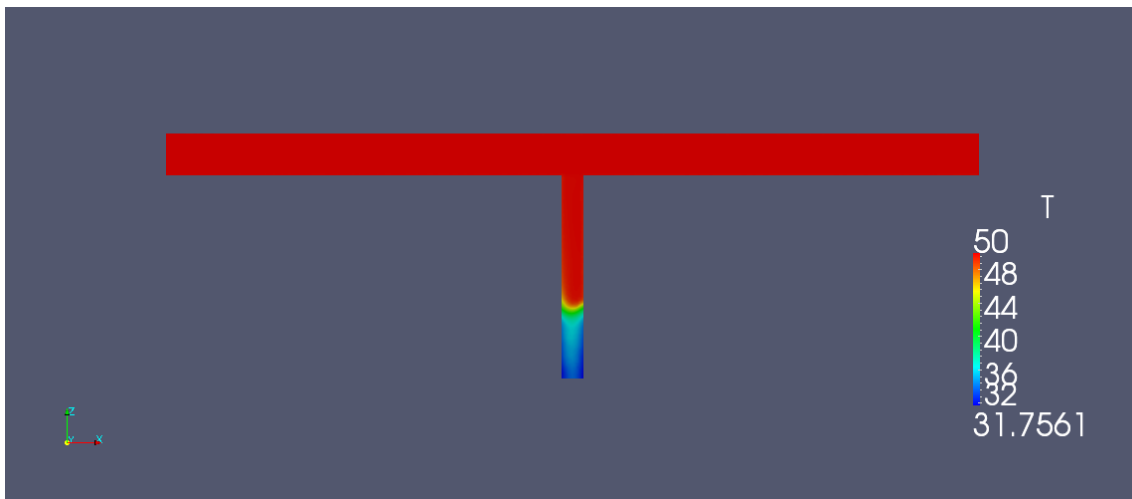


Figure 8.16: Contour of the temperature distribution for the converged solution

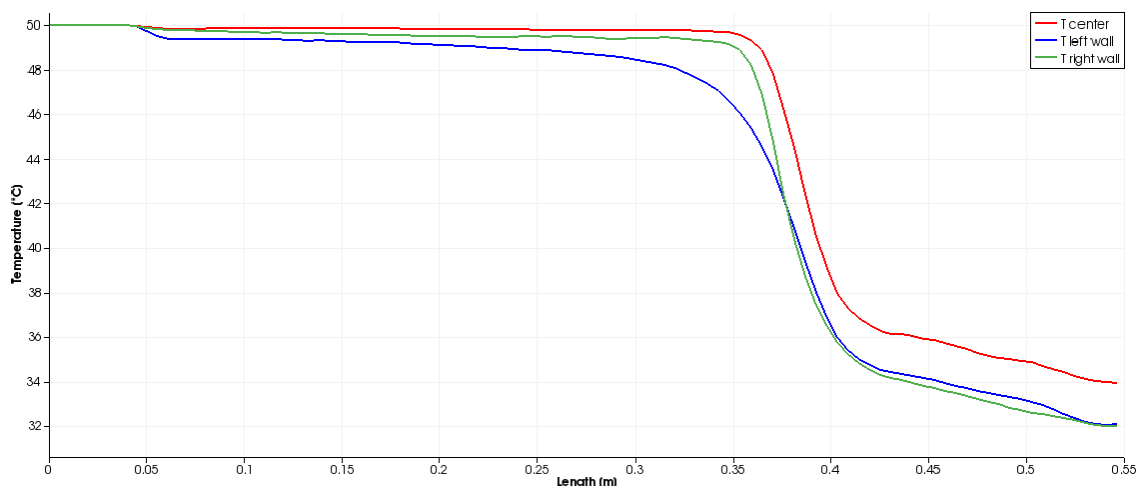
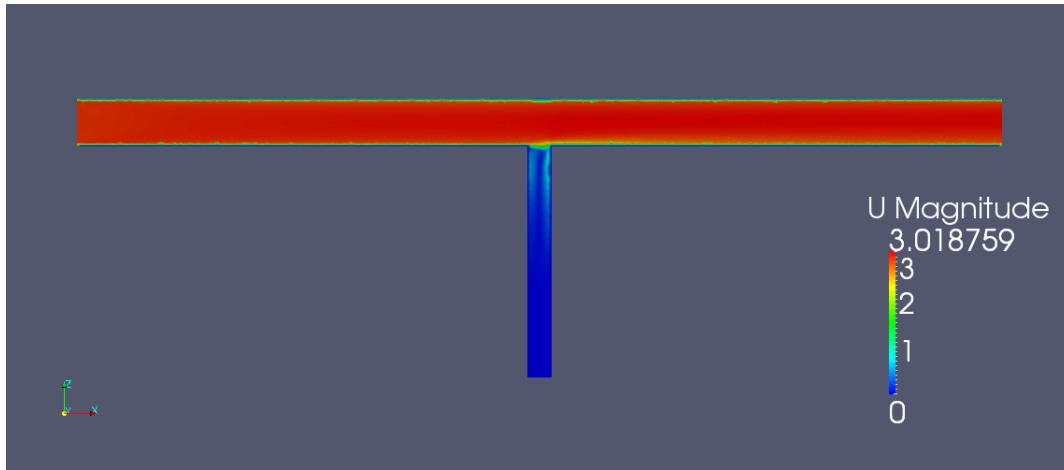


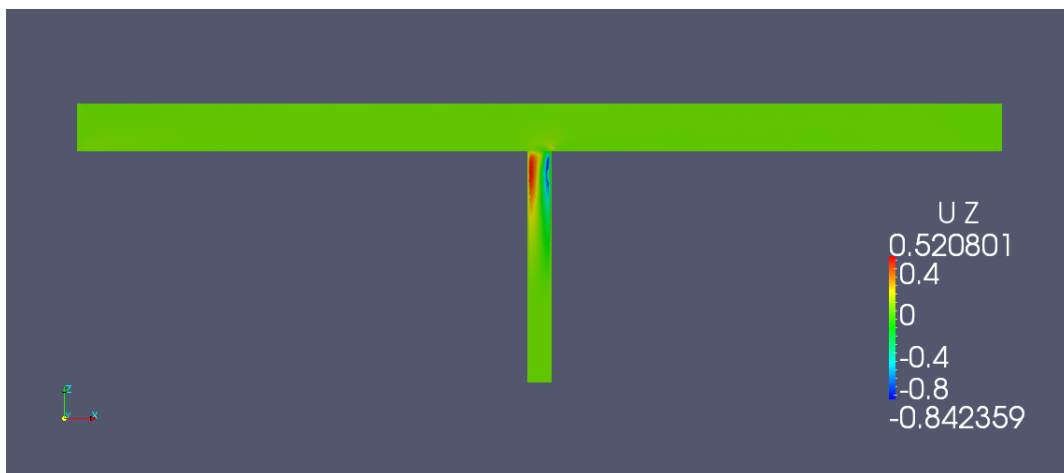
Figure 8.17: The temperature distribution at wall and center line of dead leg

Figure 8.17 illustrates a schematic temperature curve through the dead leg, in the z -direction. It shows an interesting trend which may be related to the circulation effect. It is observed that the temperature profile in the dead leg is stable with minor temperature loss until it reach a distance of $0.36\text{m} \approx 6.8D_i$ from the inlet. At this instance the temperature drops rapidly before it again stabilises at around 35°C . The results imply that there is no danger of temperature reaching TV when working under normal conditions.

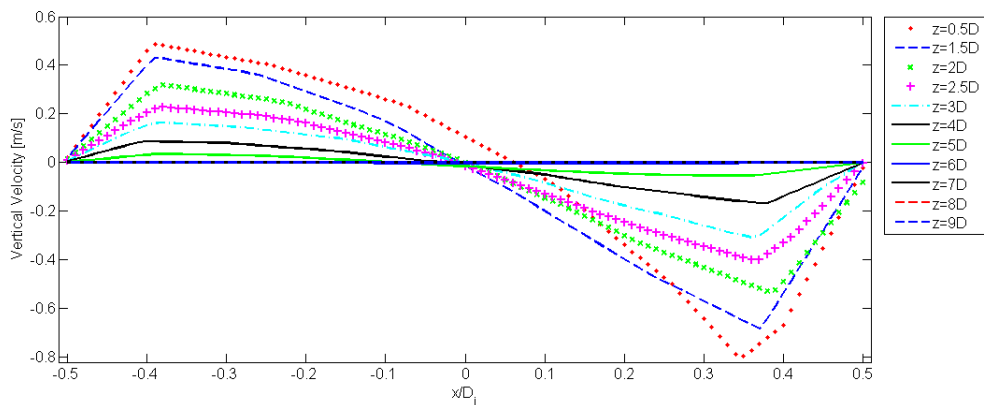
The velocity field in Figure 8.18 shows the trend of the circulation. Figure 8.18(b) shows the vertical velocity and describes the magnitude of the circulation. Where the blue color indicate the downward flow in the dead leg, having a maximum speed of $|0.84|$ m/s while the red color indication show the upward velocity of the circulation which reach a maximum velocity of $|0.52|$ m/s. The circulation magnitude shown in the contours do not seem to go as far as $6.8D_i$. Consequently, a closer investigation of the vertical velocity is shown in Figure 8.18(c) and Figure 8.19. The vertical velocity for each diameter segment is plotted through the dead leg and the circulation is indicated with velocity vectors in the two figures respectively. Figure 8.18(c) shows that the vertical velocity decrease for each segment until it reach $6D_i$ where the water is almost stagnant. Thus investigating Figure 8.19 it is observed some circulation lines all the way up to $6.8 D_i$. This is consistent to Figure 8.17 and it implies that the temperature is maintained due to the circulation, even if it is just small vortex motions.



(a) The magnitude of the velocity field



(b) The velocity field developed in z-direction



(c) The vertical velocity plotted for diameter segments

Figure 8.18: The velocity field in the T-junction for the final converged state

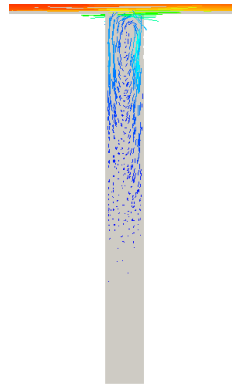


Figure 8.19: The circulation presented by velocity vectors

8.5. Verification and discussions of Case 02

As described briefly in the introduction Habib et al. [2005a] studied the circulation effect in the dead leg for several length/diameter (L/D) ratios for an oil/water mixture at an inlet velocity of 1 m/s. They obtained their results from both simulations in Fluent and experiments in a flow loop. This is the closest to a realistic case available today. Therefore, a simulation is performed in OpenFOAM with increased similarity to the experiment done in Habib et al. [2005a]. The fluid properties for viscosity should be changed and the velocity decreased. However, as Habib et al. [2005a] do not state any value for their viscosity it is chosen to perform a qualitative investigation. This is done by changing the inlet velocity to 1 m/s in Case 02. The results which are obtained are compared with the velocity profiles in Habib et al. [2005c].

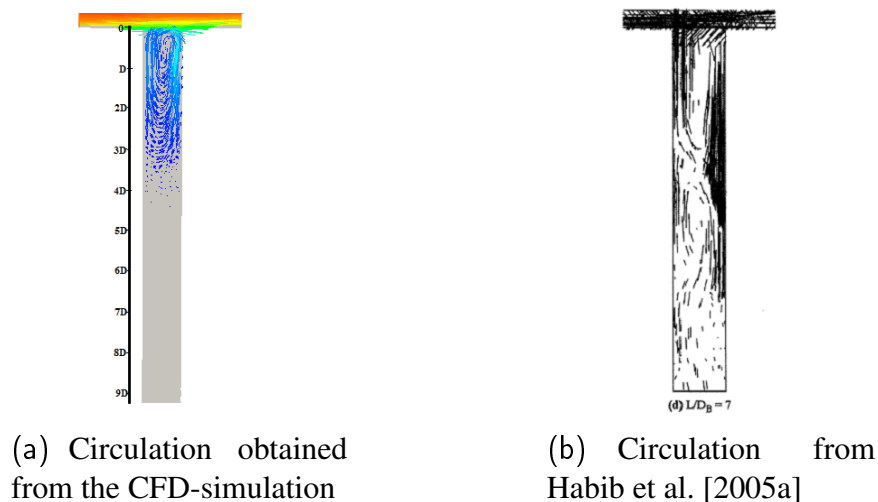
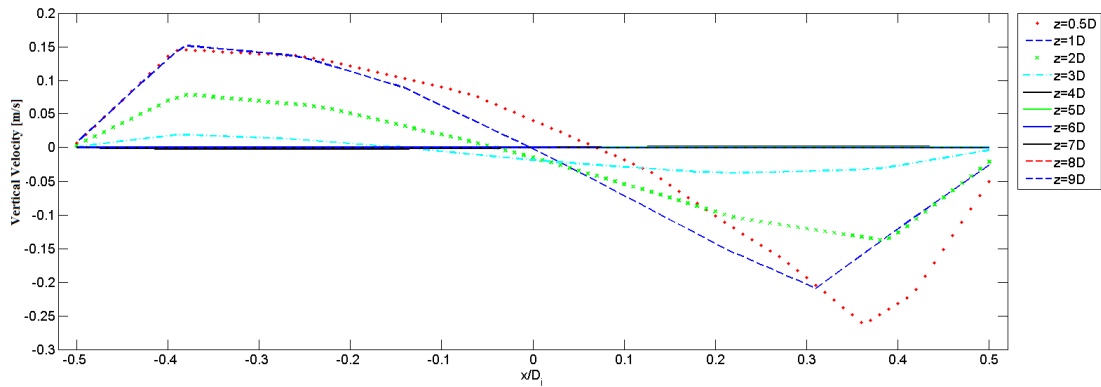
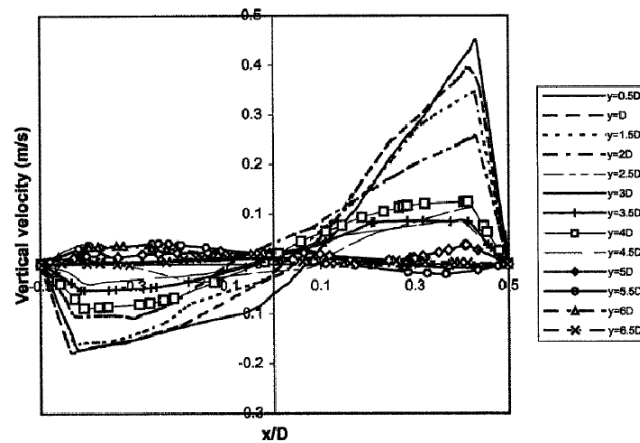


Figure 8.20: Velocity magnitude of circulation in the dead leg



(a) Vertical velocity plotted from result retrieved in OpenFOAM



(b) Vertical velocity from Habib et al. [2005c] report on effect of geometry in vertical dead legs

Figure 8.21: The vertical velocity in the dead leg for each diameter segment

Figure 8.20 shows a comparison of the circulating water in the dead leg. Habib et al. concluded that the circulation would cease to exist as the distance from the dead leg inlet increases above $\approx 3D_i$. This is also seen in the results obtained in OpenFOAM, Figure 8.20(a). In addition, Habib et al. states that up to $5D_i$ they observed vortex motion, this vortex effect is not as clear from the result obtained in OpenFOAM. This may be due to the difference in viscosity and the two phase mixture in Habib et al.’s experiment.

Figure 8.21 also implies a good qualitative agreement between the results in OpenFOAM and the results by Habib et al. Note that the velocity profiles are mirrored due to the orientation of the coordinate axis. The results coincide with Habib et al.’s conclusion that after $3D_i$ the vertical velocity is approximately zero. As the velocity is decreased from Case 02 it is observed that the circulation field is decreased. The effect this has on the temperature is illustrated in Figure 8.22, comparing the result with Figure 8.17 it is observed that the temperature falls rapidly much closer to the dead leg inlet.

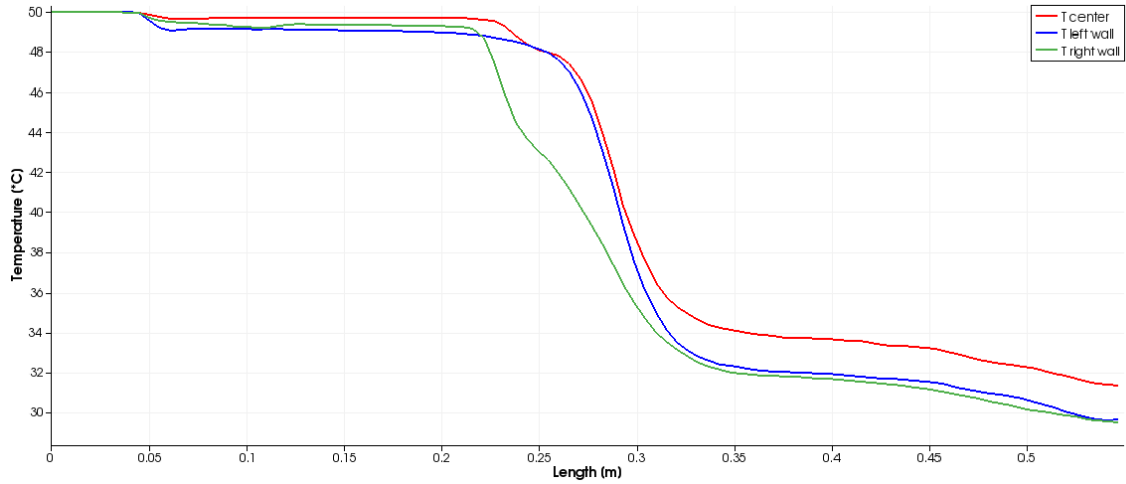


Figure 8.22: Schematic presentation of the temperature through the dead leg when the internal velocity is 1m/s

8.6. Case 03 – Shut down after normal working conditions

This case simulates a process shut down. The initial temperature field is implemented from Case 02 and it is the developed steady state temperature which is obtained under normal working conditions. The results are found from probe measurements in both main pipe and dead leg. These illustrate the temperature loss over the executed time interval. Figure 8.23 shows how the temperature develops in the main pipe. The probes are located in the center and at the wall of the main pipe. The results show that the temperature starts to fall at the wall shortly after execution. During the execution time the temperature in the center experiences a temperature loss of 26°C.

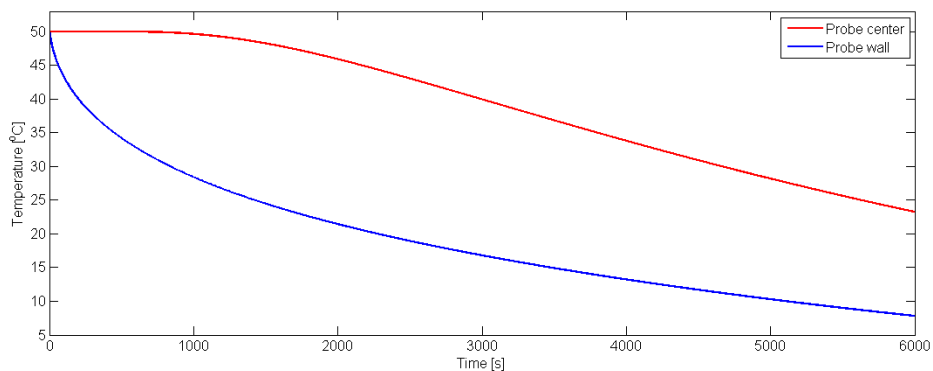


Figure 8.23: Measurement taken at probes located in the main pipe

8.6. CASE 03 – SHUT DOWN AFTER NORMAL WORKING CONDITIONS

Figure 8.24 shows the probe measurement of temperature in the center of the dead leg. The first half and the second half of the probes is presented in the top and bottom sub figure respectively. The result implies that all probes located more than $1D_i$ from the inlet eventually goes below TV. The probes located in the lower most part of the dead leg, from $\approx 7D_i$, goes below TV after approximately 3500 s. The probes located between $1D_i$ and $6D_i$ goes below TV simultaneously after approximately 4100 s. At this time only the inlet is above TV. Figure 8.25 shows the temperature contour 2800s after shut down was proceeded. The contour shows that the temperature has reached the TV in the dead leg, and the main pipe has lost some of its initial temperature. The plot in Figure 8.26 shows that the temperature has reached the threshold value close to the wall. It may also be observed that the part of the dead leg where $L/D_i > 6$ the temperature is approximately 3°C lower that the rest of the deadleg wall. This is due to the effect of circulation which was observed in Case 02, and then transferred to this case through the initial temperature field.

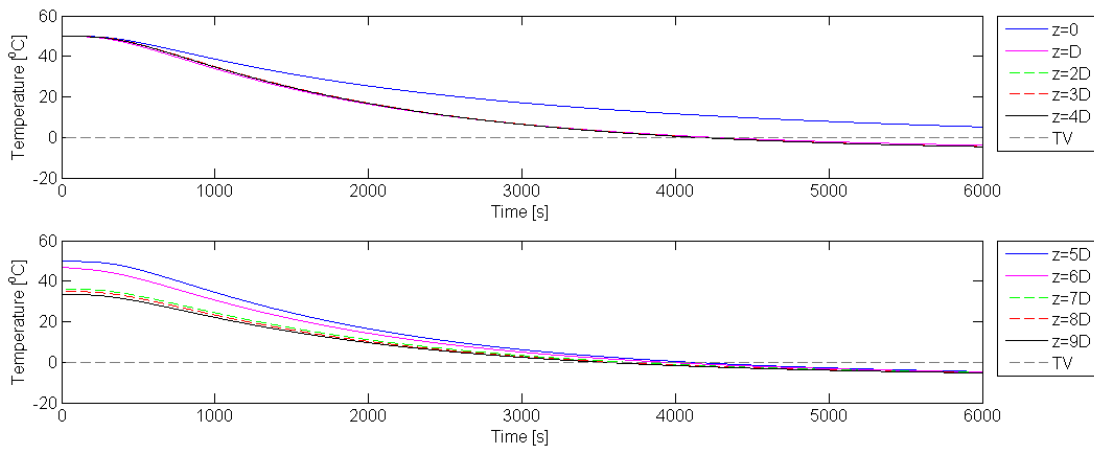


Figure 8.24: Measurement taken at probes located in the dead leg

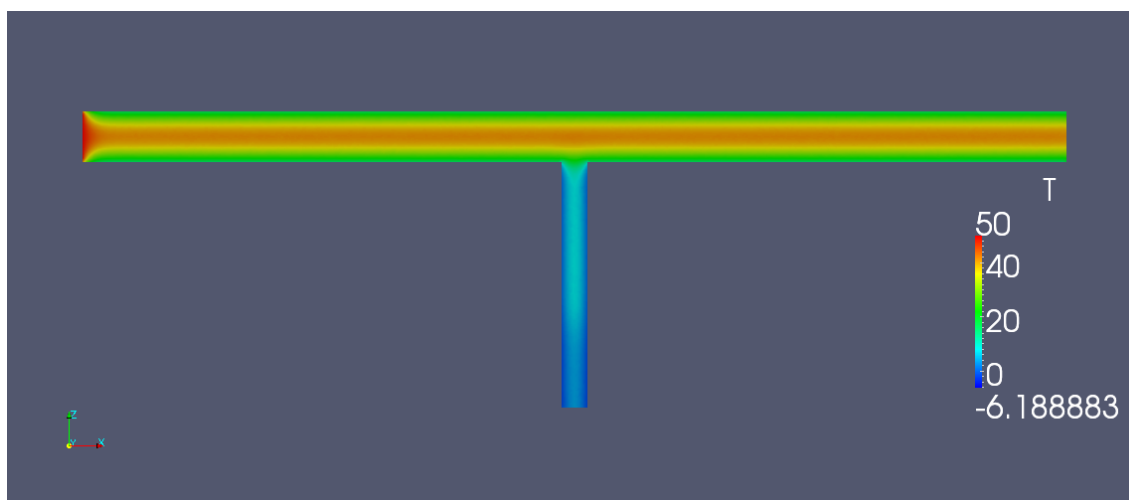


Figure 8.25: Temperature contour 2800s after shut down

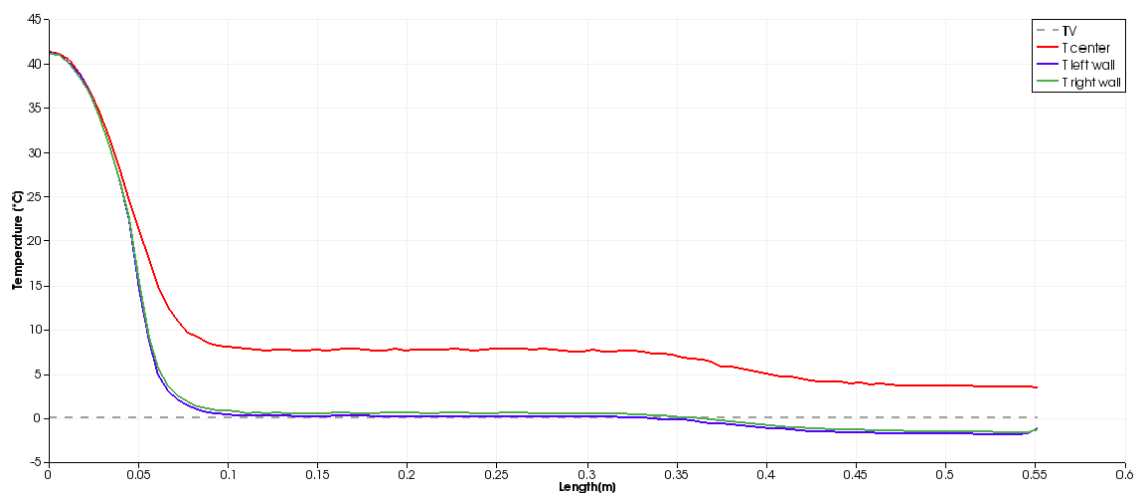
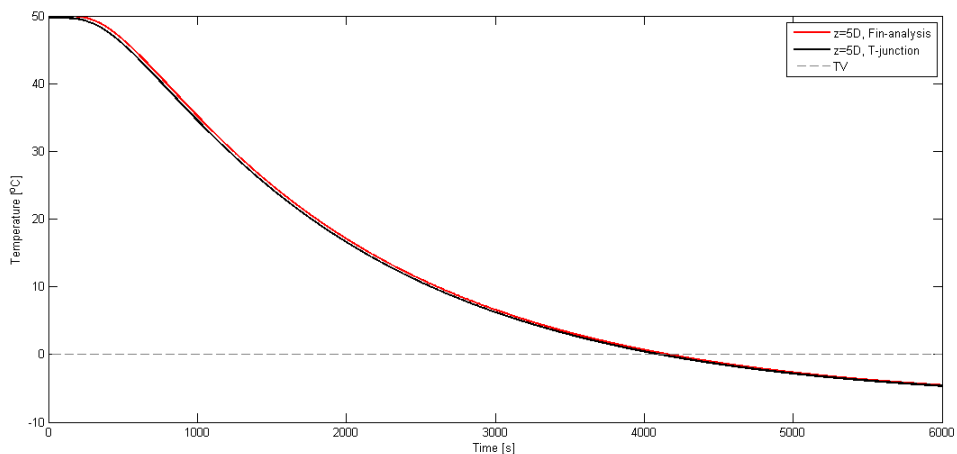


Figure 8.26: Temperature distribution in the dead leg 2800s after shut down

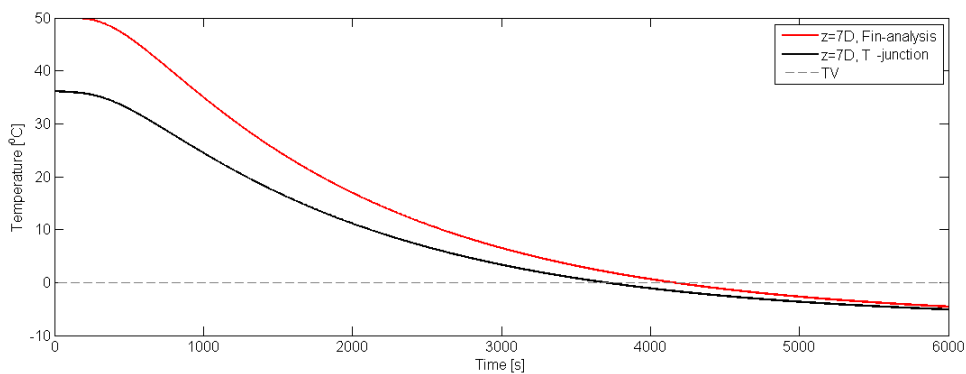
8.7. Industrial vs. realistic approach

As this thesis was in collaboration with Aker Solutions, one of the objectives was to compare the industrial approach against a realistic case. The industrial approach was similar to Case 01, and the realistic shut down was Case 03. Case 03 has been developed from the second case as the initial temperature field was obtained. This implies that the initial temperature field did not have the uniform initial temperature as Case 01. Figure 8.27 shows

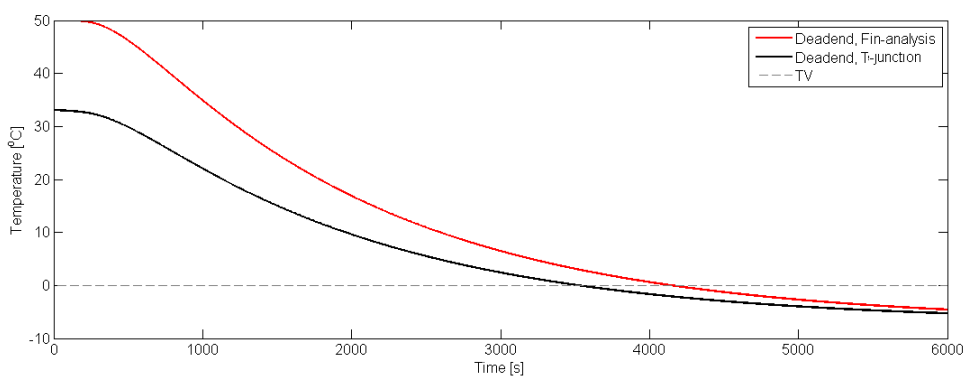
the temperature over time at a distance of $5D_i$, $7D_i$ from the inlet and at end respectively. It may be observed in Figure 8.27(b) and Figure 8.27(c) that the initial temperature for the T-junction start at 37°C and 32°C respectively. This effect is a consequence of the stagnant fluid located for distances, $L > 6D_i$, from the dead leg inlet. The result obtained shows a good agreement as it only is about 10% difference in the time which the water reaches TV. The reason why it is said to be a good agreement is the complexity of the OpenFOAM simulation. If such a complex process reach the approximate result as can be obtained by the Fin-analysis. Then the cost of running a complex system is unnecessary and the uncertainty of hidden errors will be decisive compared to the difference of the two solutions. However, the use of CFD-studies may be useful to provide a detailed knowledge of temperature behaviour during working hours.



(a) [Measurement of temperature over time at a location of $5D_i$ from the dead leg inlet



(b) Measurement of temperature over time at a location of $7D_i$ from the dead leg inlet



(c) Measurement of temperature over time at the end of dead leg

Figure 8.27: Measurement of the temperature over time for $5D_i$, $7D_i$ and end for Case 01a and Case 03

9. Conclusion

This thesis investigates the temperature development in dead leg for several approaches. The conclusion of the result can however be summarized as follows,

- The CFD model of the Fin-analysis is verified with the analytical solution and it is a 3D model of the existing industrial approach.
- The wind velocity has a significant effect on the temperature development in the pipe. This implies that it is important to take measures to prevent the wind effect when a shut down is conducted.
- The circulation effect prevents the fluid in the dead leg to reach TV. Furthermore, this indicates that the water in the dead leg only is stagnant at the very end. The design and length of the dead leg may be decided on from the calculated velocity to ensure a minimum of stagnant water.
 - $u = 2.86 \text{ m/s} \rightarrow$ water is stagnant when $L_D/D_i > 6$ from dead leg inlet.
 - $u = 2.0 \text{ m/s} \rightarrow$ water is stagnant when $L_D/D_i > 5$ from dead leg inlet.
 - $u = 1.0 \text{ m/s} \rightarrow$ water is stagnant when $L_D/D_i > 4$ from dead leg inlet.
- The shut down occurring after a normal operation shows that the temperature loss not only occurs in the dead leg, but also the main pipe.
- The comparison of the industrial approach and the realistic simulation seem to be in accordance, as there is only about 10% different in the time to reach TV. This imply that the existing industrial approach is sufficient when accounting for the ALARP principle. However, the overall conclusion is that the temperature loss when a shut down is conducted lead to rapid heat loss, and care must be taken.
- It is recommended that when conducting a planned shut down, the weather conditions must be taken into considerations. If an unexpected shut down should occur during periods of cold weather and high wind velocities. Measures such as wind shielding must be performed as long as the temperature is above design temperature. Also the design temperature should be set from weather data at the current

location.

9.1. Further work

This CFD model is created so that similar problems may be investigated, however, some important aspects should be taken into account,

- The CFD-model has been developed from NORSOK-standard design criteria, this might not be adequate. To actually be sure that the design temperature is appropriate, historic weather data from the current platform location should be used.
- As the grid independence test shows some deviations for temperature, this should be investigated more thoroughly, making sure that the results do not depend on the grid distribution.
- To fully validate the model experiments with similar parameters should be performed to see that the circulation and temperature measurements are in agreement with the result from this simulation.
- A parameter study of wall thickness and material effect should be investigated. This can be performed by the use of this developed model.
- Further studies of the velocity and circulation field should be investigated to define a design criterion for dead leg lengths.
- Tests for different mediums should also be investigated to determine the temperature development and velocity field.
- As the model is exposed to the external overall heat transfer coefficient this might lead to inadequate cooling rates for different wind velocities. Consequently a model with multiphase flow could be created and compared to the existing model.

Bibliography

- Summer Intern Project Aker Solutions. Heat transfer model documentation, June 2011. Developed a linear system of equations to solve the dead leg heat transfer.
- H. Andersen. Computational study of heat transfer in subsea deadlegs for evaluation of possible hydrate formation. 2007.
- Y.A. Cengel. *Heat and mass transfer*, chapter 3.4 & 3.6. McGraw-Hill, 2006.
- Y.A. Cengel, R.H. Turner, and R. Smith. Fundamentals of thermal-fluid sciences. *Applied Mechanics Reviews*, 54:B110, 2001.
- R.L. Daugherty, J.B. Franzini, and EJ Finnemore. Fluid mechanics with engineering applications. *Recherche*, 67:02, 1985.
- DK Edwards and I. Catton. Prediction of heat transfer by natural convection in closed cylinders heated from below. *International Journal of Heat and Mass Transfer*, 12(1): 23–30, 1969.
- OpenFOAM Foundation. Users guide. *User Guide, OpenCFD Ltd*, 2011. Downloaded PDF-file, 04.03.2012.
- Bernhard Gschaider. Contrib groovybc. http://openfoamwiki.net/index.php/Contrib_groovyBC, 2012. Contribution to GroovyBC – swak4foam (visited, 23.04).
- MA Habib, HM Badr, SAM Said, I. Hussaini, and JJ Al-Bagawi. On the development of deadleg criterion. *Journal of fluids engineering*, 127(1):124–135, 2005a.
- MA Habib, HM Badr, SAM Said, EMA Mokheimer, I. Hussaini, and M. Al-Sanaa. Characteristics of flow field and water concentration in a horizontal deadleg. *Heat and mass transfer*, 41(4):315–326, 2005b.
- MA Habib, SAM Said, HM Badr, I. Hussaini, and JJ Al-Bagawi. Effect of geometry on flow field and oil/water separation in vertical deadlegs. *International Journal of Numerical Methods for Heat & Fluid Flow*, 15(4):348–362, 2005c.
- B.H. Hjertager. Lecture notes in openfoam. *Lectures on CFD programming in Open-FOAM*, 2009.

- B.H. Hjertager, S. Herbert, and R. Vilums. Mixed bc, heat transfer, laplacianfoam. <http://www.cfd-online.com/Forums/openfoam/74593-mixed-bc-heat-transfer-laplacianfoam.html>, 2012. (last visited, 28.04).
- SW Hong. Natural circulation in horizontal pipes. *International Journal of Heat and Mass Transfer*, 20(6):685–691, 1977.
- F. Incropera and D.P DeWitt. *Introduction to heat transfer*, chapter 3.3.1 , 3.6.2 , 7.4 & 8. John Wiley and Sons Inc., New York, NY, 2005.
- Chemical safety & hazards Investigation board US. Investigation report lpg fire at valero, mckee refinery. Technical report, 2007.
- BS ISO 12241. Thermal insulation for building equipment and industrial installations - calculation rules. Standard, Energy performance of materials components and buildings, June 2008. International British Standard.
- M. Kaszniak. Oversights and omissions in process hazard analyses: Lessons learned from csb investigations. *Process Safety Progress*, 29(3):264–269, 2010.
- NORSOK L-002. Piping system layout, design and structural analysis. Standard Edition 3, Standard Norway, Strandveien 18, P.O. Box 242, N-1326 Lysaker, Norway, July 2009.
- R.J. LeVeque. *Finite difference methods for ordinary and partial differential equations*. Society for Industrial and Applied Mathematics Philadelphia, 2007.
- U.A. Mme. *Free convection flow and heat transfer in pipe exposed to cooling*. PhD thesis, NTNU, Trondheim, October 2010.
- Jørgen Osenbroch. Discussion of problem definition, 2012. Withdrawn information from discussions in monthly meetings.
- NORSOK P-001. Process design. Standard Edition 5, Standard Norway, Strandveien 18, P.O. Box 242, N-1326 Lysaker, Norway, September 2006.
- Norske Shell AS. Prosedyre for operering av nyhamna i kuldeperioder. Procedure, Operations, September 2010.
- Norske Shell AS. Piping schedules, 2011. For internal use on piping specifications.
- Norske Shell AS. Discussion with operation personnel, April 2012.
- R.K. Sinnott. *Chemical Engineering Design*, volume 6, chapter 5.5.2, pages 256–257. Elsevier Ltd., 2009.
- B.M. Sumer and J. Fredsøe. *Hydrodynamics around cylindrical structures*, volume 26. World Scientific Pub Co Inc, 2006.

H.K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Prentice Hall, 2007.

L.V. Voigt. Note on turbulence and turbulence models. Technical University of Denmark, 2001. Extract from Department note 'Comparison of Turbulence Models for Numerical Calculation of Airflow in an annex 20 Room.

Frank M. White. *Viscous Fluid Flow*. McGraw–Hill, 3rd edition, 2006.

WikiFOAM. BuoyantboussinesqPisoFoam. <http://openfoamwiki.net/index.php/BuoyantBoussinesqPisoFoam>, 2012. Explanation of the implementation and equation of the BuoyantBoussinesqPisoFoam solver (last visited, 29.04).

Appendices

A. Project Plan

Project title:

Numerical and Analytical Study of Steady State and Transient Heat Transfer in Liquid Filled Dead legs.

Background:

Why look into this? What is the reason for asking questions regarding this topic? How can this be solved in a good manner? How do different scenarios change the temperature profile? Is the difference in result rather large? What kind of standards and information exists today?

Background of this project is that the knowledge of how the temperature distribution in a dead leg is not sufficient. The formal way of preceding the questions of dead leg is today based on experiences. If a proper framework of simulating the temperature distribution this would increase the integrity of the whole pipe system.

The objective of the thesis is to develop a framework for estimating the heat transfer between liquid filled main line and dead legs such as drain points, vacuum breakers, tie in points etc. Establish an empirical model to verify against CFD simulations.

Problem formulation:

How does the heat transfer distribution evolve in a dead leg when subjected to both steady and transient flow scenario? What limits does this give the dead leg design/parameters?

Main objectives and sub-objectives:

Create a framework for estimating the heat transfer / temperature distribution

- 1 Make a literature review
- 2 Establish an empirical model for verification
- 3 Create a CFD model for different scenarios and parameters

Make this model a usable tool for people working with similar cases.

Project activities:

- 1 Find relevant literature regarding general heat transfer, both steady state and transient. Do some research on what research already done on the subject.
- 2 Establish the empirical model both for steady state and transient flow to find the temperature profile.
- 3 Use CFD simulating program, OpenFOAM to create geometry of the deadleg and finding the temperature distribution for different scenarios.
- 4 Do an analysis on how the CFD simulation is compared to the analytical empirical model. See if new results can be seen in respect with old research.
- 5 Make a short procedure on how to use the model in real work situations

Research methods:

The report perform both a theoretical and experimental studies. When saying experimental this is to be understood as a computational study, using simulation tools to create an understanding of the physical problem. Then the simulated results should be compared to either old articles regarding a similar problem or to a new empirical model created for this purpose.

Work organization and resources:

As the author is employed in the external company, Aker Solutions, therefore an office and computer is provided from them. The most important resource for this master thesis is a computer that is able to run the software for simulation, namely OpenFOAM. The writer also intends to use the text editing program called LaTeX if this is okay by the supervisor. The text editing program, Word, might be used in writing status reports and for adding ideas that are not yet to be implemented in the final report template.

Supervisor and advisor (UiS and Industrial company):

The representative supervisor from UiS is Mr. Bjørn H. Hjertager and the representative from Aker Solutions is Mr. Jørgen Osenbroch.

Reporting

Meeting with the supervisor is to be held every second week, and drafts 2 drafts to be handed in before the final report is issued.

Time schedule of activities:

19. Jan. – meeting with supervisor
01. Feb. – Hand in a formal contract and problem definition

Week 4 – Meeting with external advisor to discuss problem definition and tentative plan. Monthly status meeting between the author of the thesis and external advisor,

Discussion concerning progress and developed result should be performed every second week between the author and supervisor. If it at times is regarded as not necessary the discussion can be postponed until the next scheduled meeting. If the supervisor wants to have a status report before each meeting, this should be agreed on.

Since the master thesis is a 30 ECTS. This equals about 900 working hours. Since the thesis takes place from the 01.02.12 until the 15.06.12 this would equal working days of 9.5 hours a day. However, as there is also some work being executed before the official start, this can be subtracted from the 900 hours. As the normal week is 5 days an 7.5 hour a day this leads to a total amount of 712.5 hours. Including some work in the weekend this will eventually be fulfilled. Therefore the working plan for this thesis will be about 7.5 hours a day including 7.5 hours during the two days in the weekend. This gives a total of 855 hours.

The restoring 55 hours is been used before the official start date. They have been used for literature research regarding the subject, development of report template, planning the thesis, defining the objectives and problem definition. Figure A.1 shows the Gantt chart created to plan the activities during the project execution.

References:

References will be kept updated at all times and referenced to as needed.

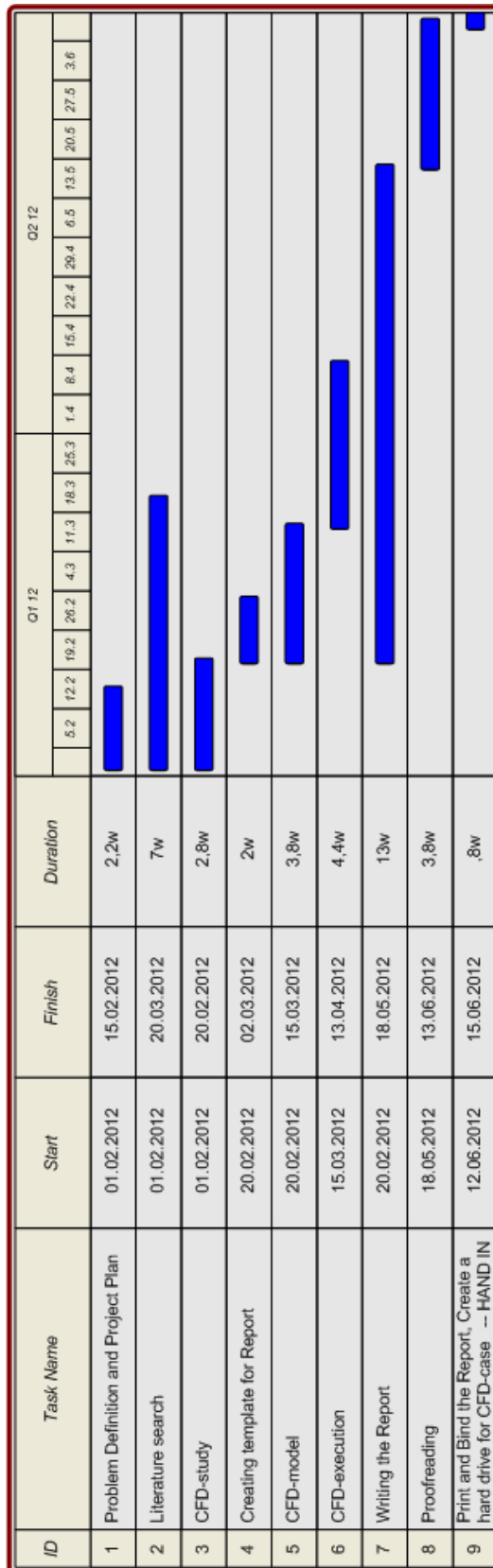


Figure A.1: Gant chart of the planned activities and time schedule

B. Initial Conditions

Velocity, U :

Table B.1: Initial conditions for velocity

	Inlet	Outlet	WallsM	WallsD	Dead end
Boundary Condition	FixedValue	InletOutlet	FixedValue	FixedValue	FixedValue
Inletvalue		\$internal Field			
Value	Uniform (2.86 0 0)	Uniform (2.86 0 0)	Uniform (0 0 0)	Uniform (0 0 0)	Uniform (0 0 0)
Internal field	uniform (0 0 0)				

Pressure, p :

Table B.2: Initial conditions for pressure, p

	Inlet	Outlet	WallsM	WallsD	Deadend
Boundary Condition	zeroGradient	FixedValue	zeroGradient	zeroGradient	zeroGradient
Value		Uniform 0			
Internal field	uniform 0				

Turbulent kinetic Energy, k :

Table B.3: Initial conditions for turbulent kinetic energy, k

	Inlet	Outlet	WallsM	WallsD	Deadend
Boundary Condition	turbulent Intensity Kinetic Energy Inlet	InletOutlet	kqRWall Function	kqRWall Function	kqRWall Function
Intensity/ Inletvalue	0.06	uniform 0.0197			
Value	uniform 0.0197	\$internalField	uniform 0.0197	uniform 0	uniform 0
Internal field	uniform 0.0197				

Turbulent dissipation, ε :

Table B.4: Initial conditions for turbulent dissipation, ε

	Inlet	Outlet	WallsM	WallsD	Deadend
Boundary Condition	turbulent Mixing Length Dissipation RateInlet	InletOutlet	epsilonWall Function	epsilonWall Function	epsilonWall Function
MixingLength/ Inletvalue	0.00714	uniform 0.0634			
Value	uniform 0.0634	\$internalField	uniform 0.0634	uniform 0	uniform 0
Internal field	uniform 0.0634				

Turbulent viscosity, ν_t :*Table B.5: Initial conditions for ν_t*

	Inlet	Outlet	WallsM	WallsD	Deadend
Boundary Condition	calculated	calculated	nutWall Function	nutWall Function	nutWall Function
Value	Uniform 0.00055	Uniform 0.00055	Uniform 0.00055	Uniform 0.0	Uniform 0.0
Internal field	uniform 0.00055				

Turbulent diffusion, α_t :*Table B.6: Initial conditions for α_t*

	Inlet	Outlet	WallsM	WallsD	Deadend
Boundary Condition	calculated	calculated	alphanWall Function	alphanWall Function	alphanWall Function
Value	Uniform 0	Uniform 0	Uniform 0	Uniform 0	Uniform 0
Internal field	uniform 0				

Temperature, T :

Table B.7: Initial conditions for Temperature, T

	Inlet	Outlet	WallsM	WallsD	Deadend
Boundary Condition	FixedValue	InletOutlet/ zeroGradient	groovyBC	groovyBC	zeroGradient
Inletvalue		\$internalField			
Value	uniform 323	\$internalField	uniform 323	uniform 323	
Internal field	uniform 323				

Table B.8: Variables for GroovyBC

GroovyBC - variables		
Parameter	Main pipe	Dead leg
h_{tot} [W/m ² K] at 9 m/s	25.53	48.39
h_{tot} [W/m ² K] at 3 m/s	13.36	25.44
ρ [kg/m ³]	998.5	
c_p [J/kgK]	4185	
$k = DT\rho c_p$ [W/mK]		
$k_t = \kappa_{EFF}\rho c_p$ [W/mK]		
T_{inf} [K]	266	

NOTE that in *groovyBC* mixed boundaries, the *fractionExpression* is expressed as,

$$\frac{1}{1 + \frac{k}{mag(Delta())}} \times h_{tot} \rightarrow k \frac{dT}{dn} + h\Delta T = 0 \quad (B.1)$$

C. Governing Equations

From fluid mechanics the governing equations for incompressible flow and constant transport properties are yielding. These are here presented in Cartesian coordinates as this is the approach used in OpenFOAM. The equations and information in this chapter are retrieved from White [2006].

C.1. Vector notation and Indicial notation

The indicial $i = j = k = [1, 2, 3] = [x, y, z]$. The usual vector differential operation are expressed as,

$$\text{Gradient operator, } \nabla, \tag{C.1}$$

$$\nabla = \frac{\partial}{\partial x_i} = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3} \right] = \left[\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right] \tag{C.2}$$

$$\text{Divergence operator, } \nabla \cdot, \tag{C.3}$$

$$\nabla \cdot = \frac{\partial}{\partial x_i} = \frac{\partial}{\partial x_1} + \frac{\partial}{\partial x_2} + \frac{\partial}{\partial x_3} = \frac{\partial}{\partial x} + \frac{\partial}{\partial y} + \frac{\partial}{\partial z} \tag{C.4}$$

$$\text{Laplacian operator, } \nabla^2, \tag{C.5}$$

$$\nabla^2 = \frac{\partial^2}{\partial x_i^2} = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{C.6}$$

Note that $\partial x_i^2 = \partial x_i \partial x_i$ and $\nabla^2 = \nabla \cdot \nabla$

C.2. Equation from the laws of conservation

Continuity equation:

$$\nabla \cdot \mathbf{V} = 0 \quad (\text{C.7})$$

where

$$\mathbf{V} = u, v, w$$

Navier-Stokes:

$$\frac{D\mathbf{V}}{Dt} = -\frac{1}{\rho}\nabla g + \frac{1}{\rho}\nabla \cdot \tau_{ij} \quad (\text{C.8})$$

Energy equation:

$$\rho c_p \frac{DT}{Dt} = k\nabla^2 T + \Phi \quad (\text{C.9})$$

where

$$\phi = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \frac{\partial u_i}{\partial x_j}$$

C.3. Turbulence equation

Reynolds average equation

This section is retrieved from Versteeg and Malalasekera [2007] The transport properties is assumed constant for incompressible turbulent flow however, there is some possible significant fluctuation, therefore, the properties is divided into a fluctuating and an average part,

$$u = \bar{u} + u' \quad p = \bar{p} + p' \quad (\text{C.10})$$

$$v = \bar{v} + v' \quad T = \bar{T} + T' \quad (\text{C.11})$$

$$w = \bar{w} + w' \quad (\text{C.12})$$

Then if considering the incompressible continuity equation (C.7), substituting u, v and w from equation (C.12) and taking the time average, the continuity becomes satisfied for the

mean velocities,

$$\frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} + \frac{\partial \bar{w}}{\partial z} = 0 \quad (\text{C.13})$$

then substituting equation (C.13) from equation (C.7) without taking the time average this gives the continuity satisfied for the fluctuating velocities.

$$\frac{\partial u'}{\partial x} + \frac{\partial v'}{\partial y} + \frac{\partial w'}{\partial z} = 0 \quad (\text{C.14})$$

This gives two separate continuity equations for the mean and fluctuating velocities.

Now looking at the Navier-Stokes equation (C.8) and preceding the same procedure to present it for the turbulent properties.

$$\rho \frac{D\bar{\mathbf{V}}}{Dt} + \rho \frac{\partial}{\partial x_j} (\overline{u'_i u'_j}) = \rho g - \nabla \bar{p} + \mu \nabla^2 \bar{\mathbf{V}} \quad (\text{C.15})$$

In this the new term $\overline{u'_i u'_j}$ is expressed presenting 6 new unknowns to account for. $\overline{u'_i u'_j}$ this is the turbulent inertia tensor, also called the Boussinesq hypothesis,

$$\overline{u'_i u'_j} = \tau_{ij}^R = \nu_t \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} + \frac{\partial \bar{u}_k}{\partial x_k} \delta_{ij} \right) - \frac{2}{3} k \delta_{ij}$$

where τ_{ij} is the Reynolds stress, ν_t is the turbulent eddy viscosity, k is the turbulent kinetic energy and δ_{ij} is the Kronecker delta function.

The turbulent momentum equation yield,

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial}{\partial x_j} (\overline{u_i u_j}) = - \frac{\partial}{\partial x_i} \left(\frac{\bar{p}}{\rho_0} \right) + \frac{\partial}{\partial x_j} \left\{ \nu_0 \left[\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) - \frac{2}{3} \left(\frac{\partial \bar{u}_k}{\partial x_k} \right) \delta_{ij} \right] \right\} \quad (\text{C.16})$$

The turbulent energy equation (C.17) is obtained by taking the time average of the energy equation (C.9),

$$\rho c_p \frac{D\bar{T}}{Dt} = - \frac{\partial}{\partial x_i} (q_i) + \bar{\Phi} \quad (\text{C.17})$$

where Φ is the total dissipation term and q_i is the total heat flux vector expressed by

equation (C.18) and (C.19) respectively.

$$\bar{\Phi} = \frac{\mu}{2} \overline{\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial u'_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} + \frac{\partial u'_j}{\partial x_i} \right)^2} \quad (\text{C.18})$$

$$q_i = \underbrace{-k \frac{\partial \bar{T}}{\partial x_i}}_{\text{Laminar}} + \underbrace{\rho c_p \overline{u'_i T'}}_{\text{Turbulent}} \quad (\text{C.19})$$

D. Flow chart SIMPLE-algorithm

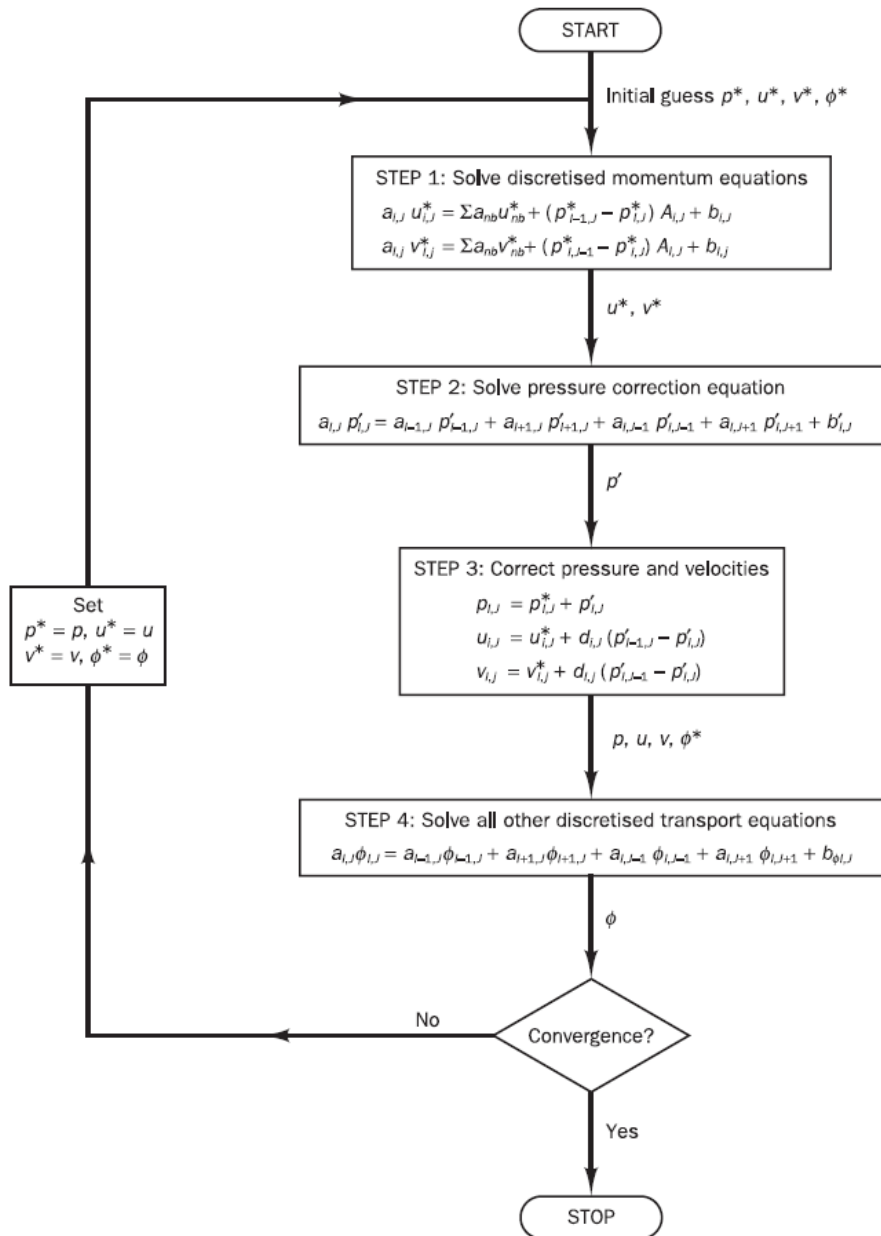


Figure D.1: Flow chart for the SIMPLE algorithm [Versteeg and Malalasekera, 2007].

E. Flow regime







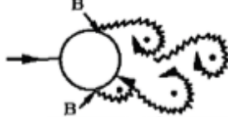


a)		No separation. Creeping flow	$Re < 5$
b)		A fixed pair of symmetric vortices	$5 < Re < 40$
c)		Laminar vortex street	$40 < Re < 200$
d)		Transition to turbulence in the wake	$200 < Re < 300$
e)		Wake completely turbulent. A: Laminar boundary layer separation	$300 < Re < 3 \times 10^5$ Subcritical
f)		A: Laminar boundary layer separation B: Turbulent boundary layer separation; but boundary layer laminar	$3 \times 10^5 < Re < 3.5 \times 10^5$ Critical (Lower transition)
g)		B: Turbulent boundary layer separation; the boundary layer partly laminar partly turbulent	$3.5 \times 10^5 < Re < 1.5 \times 10^6$ Supercritical
h)		C: Boundary layer comple- tely turbulent at one side	$1.5 \times 10^6 < Re < 4 \times 10^6$ Upper transition
i)		C: Boundary layer comple- tely turbulent at two sides	$4 \times 10^6 < Re$ Transcritical

Figure E.1: Boundary layer formation on a cylindrical cross section exposed to cross flow [Sumer and Fredsøe, 2006].

F. Matlab calculations

```
1 %% Calculation using standard theory
2
3 clear all;
4 close all;
5 clc
6
7 %% Parameters
8
9 % Pipe parameters
10 L= 0.5; %[m]
11 % Parameters for a 2" XS schedule pipe in stainless steel
12 Di = 0.052; %[m]
13 De = 0.060; %[m]
14 tw = 0.00391;
15 ks = 36.9; %[W/m*C]
16 Ae = 2*3.14*De/2 * L;
17 Ai = 2*3.14*Di/2 * L;
18 V = 3.14*Di^2 /4 * L;
19
20 % Properties of Water
21 Tiw = 50+273; %[K]
22 RHO_w = 998.5; %[kg/m^3]
23 cp = 4185; %[J/kgK]
24 m =RHO_w*V; %[kg]
25 kw = 0.6; % conductivity of water [W/mK]
26
27 % Properties of Ice
28 RHO_i = 920; %[kg/m^3]
29 hfr = 334000; %[J/kg]
30 Tfr = 0+273; %[K]
31 mi =RHO_i*V;
32
33 % Properties of outside environment
34 Te = -7+273; %[K]
35 rhoa = (1.3947+(1.1614-1.3947)*((266-250)/(300-250)));%[kg/m^3]
36 Ua = 9;
37 nua = (11.44 + (15.89-11.44)*((266-250)/(300-250)))*10^(-6); %[m^2/s]
38 mua = (159.6 + (184.6-159.6)*((266-250)/(300-250)))*10^(-7); %[kg/ms] kg
39 ka = (22.3 + (26.3-22.3)*((266-250)/(300-250)))*10^(-3); %[W/mK]
40 DTa = (15.9+(22.5-15.9)*((266-250)/(300-250)))*10^(-6);%[m^2/s]
41 cpa = (1006+(1007-1006)*((266-250)/(300-250)));%[J/kgK];
```

```

42 Pra = 0.720+(0.707-0.720)*((266-250)/(300-250)); %dimensionless
43
44 %% External flow
45 % 2" cylinder in cross section - deadleg
46
47 Rea =Ua*De / nua
48
49 if Rea ≤ 40000;
50     Ca = 0.193;%0.027; % From table 7.44 p 402
51     ma = 0.618;%0.805; % From table 7.44 p 402
52 else
53     Ca = 0.027; % From table 7.44 p 402
54     ma = 0.805; % From table 7.44 p 402
55 end
56 n = 1/3;
57 if Rea*Pra ≥ 0.2;
58     Nua = 0.3 + (0.62*Rea^0.5*Pra^(1/3))/(1+ (0.4/Pra)^(2/3))^(1/4) ...
59         * (1 + (Rea/282000)^(5/8))^(4/5);
60 else
61     Nua = Ca*Rea^(ma)*Pra^(n);
62 end
63 Nua
64
65 % Or since turbulent flow
66 Nusselt = 0.023*Rea^0.8*Pra^n
67
68 % For prandtl number > 0.7
69 ha = Nua*ka / De
70
71
72 % 4"cylinder in cross section
73 Lm = 2;
74 Dm = 0.1143
75 Dim = 0.1023
76 twm = 0.006;
77 Rem = Ua*Dm / nua
78
79
80
81
82 if Rem ≤ 40000;
83     Cam = 0.193;%0.027; % From table 7.44 p 402
84     mam = 0.618;%0.805; % From table 7.44 p 402
85 else
86     Cam = 0.027; % From table 7.44 p 402
87     mam = 0.805; % From table 7.44 p 402
88 end
89 n = 1/3;
90 if Rem*Pra ≥ 0.2;
91     Num = 0.3 + (0.62*Rea^0.5*Pra^(1/3))/(1+ (0.4/Pra)^(2/3))^(1/4) ...
92         * (1 + (Rea/282000)^(5/8))^(4/5);
93 else

```

```

94 Num = Cam*Rea^(mam)*Pra^(n);
95 end
96 Num
97
98 hm = Num*ka / Dm
99
100 A1 = 2*3.14*De/2*L;
101 A2 = 2*3.14*Dm/2*Lm;
102 %Overall Heat transfer
103 %deadleg'
104 htot = 1/(tw/ks + 1/ha)
105 % main pipe
106 htotm = 1/(twm/ks + 1/hm)
107
108 Rc = log((De)/(Di))/(2*3.14*ks)
109 h2 = 1/(ha*3.14*De)
110 Rtot = (Rc + h2) % [mK/W]
111
112 Rcm = log((Dm)/(Dim))/(2*3.14*ks) % [mK/W]
113 h2m = 1/(hm*3.14*(Dm))
114 Rtotm = (Rcm + h2m)
115 Utot = 1/Rtot % [w/mK]
116 Utotm = 1/Rtotm % [W/mK]
117
118 % Elbow
119 A3 = 2*3.14*De/2*0.9;
120
121
122 %% Internal flow
123 U = sqrt((2*998.5*9.81*0.418)/998.5); % Bernoullie, slope at a height 0.418moh.
124 Lm = 2; % [m]
125 mui = (577 + (528-577)*((323-320)/(325-320)))*10^(-6); % [kg/ms]
126 ki = 0.640 + (0.645-0.640)*((323-320)/(325-320)); % [W/mK]
127 cpi = (4180+(4182-4180)*((323-320)/(325-320))); % [J/kgK];
128 Pri = 3.77+(3.42-3.77)*((323-320)/(325-320)); %dimensionless
129 nui = mui/RHO_w
130 Rei = U*Dm / nui
131 pi = 3.14;
132 Aei = 2*3.14*Dm/2 * Lm;
133 Aii = 2*3.14*Dim/2 * Lm;
134 V = 3.14*Dim^2 /4 * Lm;
135 massF = Aii*U ;
136 Dtw = ki/(RHO_w*cpi)
137 % Assuming the flow is fully developed in the main pipe
138
139
140
141
142 %% Turbulence parameters
143
144 %kinetic turbulent energy,
145 kt = 2/3*(U*0.06)^2

```

```

146 %epsilon,
147 cmu = 0.09;
148 lel = 0.07*Dim;
149 epsilon = cmu^(3/4)*(kt^(3/2))/lel
150 nut = cmu*kt^2/epsilon
151
152 % Deadleg
153 Ud = 0;
154 %kinetic turbulent energy,
155 ktd = 2/3*(Ud*0.06)^2
156 %epsilon,
157 leld = 0.07*Di;
158 epsilond = cmu^(3/4)*(ktd^(3/2))/lel
159 nutd = cmu*ktd^2/epsilond
160
161
162
163 %% Time to freez following standard BS ISO:12241
164 PHIT = Utot*De*(50 - (-7)) %W =J/s
165 PHI = Utot*L*(50 - (-7))*3.14*De %W/mK *m*K*m =Wm = m*J/s
166
167 % Time before it start to freez
168 Tf = 0;
169 mp = 7978*V;
170 cpp = 480;
171 theta = log((50-(-7))/(0-(-7)));
172 %t = (cp*m*(50-0))/(PHI)
173
174 t=((50-(-7))*(m*cp+mp*cpp)*theta)/(PHI*3600) % Ws/Wm
175
176 % Time to freez
177 f = 100;
178 tfr = f/100 * (RHO_i*3.14*Di^2*hfr)/(PHIT*3600*4)
179
180
181 %% RESULT OPENFOAM AND EXCEL SHEET
182
183 OF=xlsread('fin9c.xls','A2:A102'); %temperature from OF
184 ex=xlsread('Ctabell.xls','D2:D102'); %teperature from excel spreadsheet
185 OFw=xlsread('fin9w.xls','A2:A102'); %temperature from OF
186
187 x=xlsread('Ctabell.xls','F2:F102'); %length scale
188 L = 0*x;
189
190 % Exact results
191 P = pi*De ;
192 le = 0.5;
193 Ac = pi*(De^2/4);
194 % Ni = 1-zeta.^2;
195 m2 = (ha*P)/(kw*Ac);
196 mu2 = m2*le^2;
197 m=sqrt(m2);

```

```

198
199 Ti = 323;
200 Theta(1) = Ti-Te;
201 %IF END IS INSULATED
202 for i= 1:length(x)
203 T(i) = cosh(m*(le-x(i))) / (cosh(m*le)) * (Ti-Te) + Te;
204 i=i+1;
205 end
206 TC=T-273;
207
208
209 figure(1)
210 plot(x,OF,'rx','LineWidth',2);
211 hold on;
212 plot(x,ex,'b+','LineWidth',2);
213 hold on;
214 plot(x,TC,'g','LineWidth',2);
215 hold on;
216 plot(x,L,'--k','LineWidth',0.5);
217 hold off;
218 legend('LaplacianFOAM temp. at pipe center','TDMA-algorithm', ...
219         'Analytical solution','TV');
220 xlabel('Length [m]');ylabel('Temperature [^oC]');
221
222 figure(2)
223 plot(x,OFw,'rx','LineWidth',2);
224 hold on;
225 plot(x,ex,'b+','LineWidth',2);
226 hold on;
227 plot(x,TC,'g','LineWidth',2);
228 hold on;
229 plot(x,L,'--k','LineWidth',0.5);
230 hold off;
231 legend('LaplacianFOAM temp. at pipe wall','TDMA-algorithm', ...
232         'Analytical solution','TV');
233 xlabel('Length [m]');ylabel('Temperature [^oC]');
234
235 %% Comparison with wind velocity at time 3600s.
236
237 % Center line
238 T1=xlsread('3600c9ms.xls','A2:A102'); %temperature from 9m/s
239 T2=xlsread('3600c3ms.xls','A2:A102'); %teperature from 3m/s
240 xg=xlsread('fin3.xls','C2:C102'); %length scale
241
242 % Wall
243 T3=xlsread('3600w9ms.xls','A2:A102'); %temperature from 9m/s
244 T4=xlsread('3600w3ms.xls','A2:A102'); %teperature from 3m/s
245
246 % Center
247
248 figure(3)
249 plot(xg,T1,'b','LineWidth',2);

```

```

250 hold on;
251 plot(xg,T2,'r','LineWidth',2);
252 hold on;
253 plot(xg,L,'--k')
254 hold off;
255 %AXIS([0 0.5 -10 50]);
256 legend('Temperature distribution, U_w = 9m/s',...
257         'Temperature distribution, U_w = 3m/s','TV');
258 xlabel('Length [m]');ylabel('Temperature [^oC]');
259
260 % Wall
261
262 figure(4)
263 plot(xg,T3,'b','LineWidth',2);
264 hold on;
265 plot(xg,T4,'r','LineWidth',2);
266 hold on;
267 plot(xg,L,'--k')
268 hold off;
269 %AXIS([0 0.5 -10 50]);
270 legend('Temperature distribution, U_w = 9m/s',...
271         'Temperature distribution, U_w = 3m/s','TV');
272 xlabel('Length [m]');ylabel('Temperature [^oC]');
273
274
275
276
277 %% Grid independence
278
279 %temperature
280 M1=xlsread('Ny170.xls','A2:A101'); %temperature from 170 000 cells
281 M2=xlsread('NY240.xls','A2:A101'); %teperature from 240 000 cells
282 M3=xlsread('Ny280.xls','A2:A101'); %temperature from 280 000 cells
283 M7=xlsread('Ny310.xls','A2:A101'); %temperature from 310 000 cells
284
285
286 % kinetic energy
287 M4=xlsread('Ny170.xls','C2:C101'); %temperature from 170 000 cells
288 M5=xlsread('NY240.xls','B2:B101'); %temperature from 240 000 cells
289 M6=xlsread('Ny280.xls','B2:B101'); %temperature from 280 000 cells
290 M8=xlsread('Ny310.xls','C2:C101'); %temperature from 310 000 cells
291
292 xg=xlsread('Ny280.xls','I2:I101'); %length scale
293
294
295 %velocity
296 M1u=xlsread('Ny170.xls','F2:F101'); %temperature from 170 000 cells
297 M2u=xlsread('NY240.xls','E2:E101'); %teperature from 238 000 cells
298 M3u=xlsread('Ny280.xls','E2:E101'); %temperature from 283 000 cells
299 M4u=xlsread('Ny310.xls','F2:F101'); %temperature from 310 000 cells
300
301 xg=xlsread('Ny280.xls','I2:I101'); %length scale

```

```

302
303
304
305
306 figure(5)
307 plot(xg,M1,'r','LineWidth',2);
308 hold on;
309 plot(xg,M2,'b','LineWidth',2);
310 hold on;
311 plot(xg,M3,'g','LineWidth',2);
312 hold on;
313 plot(xg,M7,'k','LineWidth',2);
314 hold off;
315 legend('N=170 000','N=238 000','N=283 000','N=310 000');
316 xlabel('Length [m]');ylabel('Temperature [^oC]');
317
318 figure(6)
319 plot(xg,M4,'r','LineWidth',2);
320 hold on;
321 plot(xg,M5,'b','LineWidth',2);
322 hold on;
323 plot(xg,M6,'g','LineWidth',2);
324 hold on;
325 plot(xg,M8,'k','LineWidth',2);
326 hold off;
327 legend('N=170 000','N=238 000','N=283 000','N=310 000');
328 xlabel('Length [m]');ylabel('Turbulent Kinetic energy [m^2/s^2]');
329
330 figure(7)
331 plot(xg,M1u,'r','LineWidth',2);
332 hold on;
333 plot(xg,M2u,'b','LineWidth',2);
334 hold on;
335 plot(xg,M3u,'g','LineWidth',2);
336 hold on;
337 plot(xg,M4u,'k','LineWidth',2);
338 hold off;
339 legend('N=170 000','N=238 000','N=283 000','N=310 000');
340 xlabel('Length [m]');ylabel('Velocity [m/s]');
341
342
343
344
345 %% Probe measurement
346
347 %DAEDLEG - fin 9m/s
348 D1 = xlsread('FIN9ms.xls','D5:D6004'); %probe (0 0 0) 0
349 D2=xlsread('FIN9ms.xls','F5:F6004'); %probe (0 0 -0.026) 0.5D
350 D3=xlsread('FIN9ms.xls','H5:H6004'); %probe (0 0 -0.078) 1.5D
351 D4=xlsread('FIN9ms.xls','J5:J6004'); %probe (0 0 -0.104) 2D
352 D5=xlsread('FIN9ms.xls','L5:L6004'); %probe (0 0 -0.13) 2.5D
353 D6=xlsread('FIN9ms.xls','N5:N6004'); %probe (0 0 -0.156) 3D

```

```

354 D7=xlsread('FIN9ms.xls','P5:P6004'); %probe (0 0 -0.2085) 4D
355 D8=xlsread('FIN9ms.xls','R5:R6004'); %probe (0 0 -0.261) 5D
356 D9=xlsread('FIN9ms.xls','T5:T6004'); %probe (0 0 -0.3135) 6D
357 D10=xlsread('FIN9ms.xls','V5:V6004'); %probe (0 0 -0.366) 7D
358 D11=xlsread('FIN9ms.xls','X5:X6004'); %probe (0 0 -0.4186) 8D
359 D12=xlsread('FIN9ms.xls','Z5:Z6004'); %probe (0 0 -0.4711) 9D
360 D13=xlsread('FIN9ms.xls','AB5:AB6004'); %probe (0 0 -0.5) end
361
362 %DAEDLEG - fin 3m/s
363 F1 = xlsread('FIN3ms.xls','D5:D6004'); %probe (0 0 0)
364 F2=xlsread('FIN3ms.xls','F5:F6004'); %probe (0 0 -0.026)
365 F3=xlsread('FIN3ms.xls','H5:H6004'); %probe (0 0 -0.078)
366 F4=xlsread('FIN3ms.xls','J5:J6004'); %probe (0 0 -0.104)
367 F5=xlsread('FIN3ms.xls','L5:L6004'); %probe (0 0 -0.13)
368 F6=xlsread('FIN3ms.xls','N5:N6004'); %probe (0 0 -0.156)
369 F7=xlsread('FIN3ms.xls','P5:P6004'); %probe (0 0 -0.2085)
370 F8=xlsread('FIN3ms.xls','R5:R6004'); %probe (0 0 -0.261)
371 F9=xlsread('FIN3ms.xls','T5:T6004'); %probe (0 0 -0.3135)
372 F10=xlsread('FIN3ms.xls','V5:V6004'); %probe (0 0 -0.366)
373 F11=xlsread('FIN3ms.xls','X5:X6004'); %probe (0 0 -0.4186)
374 F12=xlsread('FIN3ms.xls','Z5:Z6004'); %probe (0 0 -0.4711)
375 F13=xlsread('FIN3ms.xls','AB5:AB6004'); %probe (0 0 -0.5)
376
377
378 % wall
379 % D4=xlsread('PROBE-DL.xls','L5:L6004'); %probe -0.027 0 -0.25
380 % D5=xlsread('PROBE-DL.xls','N5:N6004'); %probe -0.027 0 -0.5
381
382 %Tee-junction pipe
383 P0=xlsread('Tprobetee.xls','H5:H6004'); %probe main after dead leg
384 P01=xlsread('Tprobetee.xls','J5:J6004'); %probe main wall
385 P02=xlsread('Tprobetee.xls','D5:D6004'); %probe main before dead leg
386 P03=xlsread('Tprobetee.xls','F5:F6004'); %probe main wall
387
388 P1=xlsread('Tprobetee.xls','T5:T6004'); %probe (0 0 -0.051) 0
389 P2=xlsread('Tprobetee.xls','V5:V6004'); %probe (0 0 -0.077) 0.5D
390 P3=xlsread('Tprobetee.xls','X5:X6004'); %probe (0 0 -0.103) 1D
391 P4=xlsread('Tprobetee.xls','Z5:Z6004'); %probe (0 0 -0.13) 1.5D
392 P5=xlsread('Tprobetee.xls','AB5:AB6004'); %probe (0 0 -0.155) 2D
393 P6=xlsread('Tprobetee.xls','AD5:AD6004'); %probe (0 0 -0.181) 2.5D
394 P7=xlsread('Tprobetee.xls','AF5:AF6004'); %probe (0 0 -0.207) 3D
395 P8=xlsread('Tprobetee.xls','AH5:AH6004'); %probe (0 0 -0.259) 4D
396 P9=xlsread('Tprobetee.xls','AJ5:AJ6004'); %probe (0 0 -0.311) 5D
397 P10=xlsread('Tprobetee.xls','AL5:AL6004'); %probe (0 0 -0.363) 6D
398 P11=xlsread('Tprobetee.xls','AN5:AN6004'); %probe (0 0 -0.415) 7D
399 P12=xlsread('Tprobetee.xls','AP5:AP6004'); %probe (0 0 -0.467) 8D
400 P13=xlsread('Tprobetee.xls','AR5:AR6004'); %probe (0 0 -0.5196) 9D
401 P14=xlsread('Tprobetee.xls','AT5:AT6004'); %probe (0 0 -0.55)end
402 %wall
403 P16 = xlsread('Tprobetee.xls','L5:L6004'); %probe 0.974 0 0
404 P17 = xlsread('Tprobetee.xls','N5:N6004'); %probe 0.974 0 -0.5
405 P18 = xlsread('Tprobetee.xls','P5:P6004'); %probe 1.026 0 0

```



```

406 P19 = xlsread('Tprobetee.xls','R5:R6004'); %probe 1.026 0 -0.5
407 Tid=xlsread('Tprobetee.xls','B5:B6004'); %time
408 L1 = 0*Tid;
409
410 % Main pipe
411 figure
412 plot(Tid,P02,'r','LineWidth',2);
413 hold on;
414 plot(Tid,P03,'b','LineWidth',2);
415 hold off;
416 legend('Probe center','Probe wall');
417 xlabel('Time [s]');ylabel('Temperature [^oC]');
418
419 % Tee-junction DAEDLEG PIPE PLOT
420 figure(8)
421 subplot(211);
422 plot(Tid,P1,'b','LineWidth',1);
423 hold on;
424 plot(Tid,P3,'m','LineWidth',1);
425 hold on;
426 plot(Tid,P6,'g—','LineWidth',1);
427 hold on;
428 plot(Tid,P7,'r—','LineWidth',1);
429 hold on;
430 plot(Tid,P8,'k—','LineWidth',1);
431 hold on;
432 plot(Tid,L1,'k—')
433 hold off;
434 legend('z=0','z=D','z=2D','z=3D','z=4D','TV');
435 xlabel('Time [s]');ylabel('Temperature [^oC]');
436
437 subplot(212);
438 plot(Tid,P9,'b','LineWidth',1);
439 hold on;
440 plot(Tid,P10,'m','LineWidth',1);
441 hold on;
442 plot(Tid,P11,'g—','LineWidth',1);
443 hold on;
444 plot(Tid,P12,'r—','LineWidth',1);
445 hold on;
446 plot(Tid,P13,'k—','LineWidth',1);
447 hold on;
448 plot(Tid,L1,'k—')
449 hold off;
450 legend('z=5D','z=6D','z=7D','z=8D','z=9D','TV');
451 xlabel('Time [s]');ylabel('Temperature [^oC]');
452
453 %Fin analysis – deadleg plot 9m/s
454 figure(9)
455 plot(Tid,D1,'b','LineWidth',1);
456 hold on;
457 plot(Tid,D3,'m','LineWidth',1);

```

```

458 hold on;
459 plot (Tid,D4, 'g—', 'LineWidth',1);
460 hold on;
461 plot (Tid,D6, 'r—', 'LineWidth',1);
462 hold on;
463 plot (Tid,D7, 'k—', 'LineWidth',1);
464 hold on;
465 plot (Tid,D8, 'b—', 'LineWidth',1);
466 hold on;
467 plot (Tid,D9, 'm—', 'LineWidth',1);
468 hold on;
469 plot (Tid,D10, 'g', 'LineWidth',1);
470 hold on;
471 plot (Tid,D11, 'r', 'LineWidth',1);
472 hold on;
473 plot (Tid,D12, 'k', 'LineWidth',1);
474 hold on;
475 plot (Tid,L1, 'k—')
476 hold off;
477 legend('z=0', 'z=1.5D', 'z=2D', 'z=3D', 'z=4D', 'z=5D', ...
478        'z=6D', 'z=7D', 'z=8D', 'z=9D', 'TV');
479 xlabel('Time [s]');ylabel('Temperature [^oC]');
480
481 %fin-analysis - 3m/s
482 figure(10)
483 plot (Tid,F1, 'b', 'LineWidth',1);
484 hold on;
485 plot (Tid,F3, 'm', 'LineWidth',1);
486 hold on;
487 plot (Tid,F4, 'g—', 'LineWidth',1);
488 hold on;
489 plot (Tid,F6, 'r—', 'LineWidth',1);
490 hold on;
491 plot (Tid,F7, 'k—', 'LineWidth',1);
492 hold on;
493 plot (Tid,F8, 'b—', 'LineWidth',1);
494 hold on;
495 plot (Tid,F9, 'm—', 'LineWidth',1);
496 hold on;
497 plot (Tid,F10, 'g', 'LineWidth',1);
498 hold on;
499 plot (Tid,F11, 'r', 'LineWidth',1);
500 hold on;
501 plot (Tid,F12, 'k', 'LineWidth',1);
502 hold on;
503 hold off;
504 legend('z=0', 'z=1.5D', 'z=2D', 'z=3D', 'z=4D', 'z=5D', ...
505        'z=6D', 'z=7D', 'z=8D', 'z=9D', 'TV');
506 xlabel('Time [s]');ylabel('Temperature [^oC]');
507
508
509

```

```

510
511 % Comparison of wind velocity at probe 5D
512 figure(11)
513 plot(Tid,D8,'b','LineWidth',2);
514 hold on;
515 plot(Tid,F8,'r','LineWidth',2);
516 hold on;
517 plot(Tid,L1,'--g')
518 hold off;
519 legend('z=4D, with U_w = 9 m/s','z=4D, with U_w = 3 m/s','TV');
520 xlabel('Time [s]');ylabel('Temperature [^oC]');
521
522 %Comparison of probe 5D in Fin-analysis and tee-junction
523 figure(12)
524 plot(Tid,D8,'r','LineWidth',2);
525 hold on;
526 plot(Tid,P9,'k','LineWidth',2);
527 hold on;
528 plot(Tid,L1,'--g')
529 hold off;
530 legend('z=5D, Fin-analysis','z=5D, Tee-junction','TV');
531 xlabel('Time [s]');ylabel('Temperature [^oC]');
532
533 % probe 7D
534 figure(14)
535 plot(Tid,D10,'r','LineWidth',2);
536 hold on;
537 plot(Tid,P11,'k','LineWidth',2);
538 hold on;
539 plot(Tid,L1,'--g')
540 hold off;
541 legend('z=7D, Fin-analysis','z=7D, Tee-junction','TV');
542 xlabel('Time [s]');ylabel('Temperature [^oC]');
543
544 %probe end
545 figure(15)
546 plot(Tid,D13,'r','LineWidth',2);
547 hold on;
548 plot(Tid,P14,'k','LineWidth',2);
549 hold on;
550 plot(Tid,L1,'--g')
551 hold off;
552 legend('Deadend, Fin-analysis','Deadend, Tee-junction','TV');
553 xlabel('Time [s]');ylabel('Temperature [^oC]');
554
555
556 % Comparison wall
557 figure
558 plot(Tid,P17,'r','LineWidth',2);
559 hold on;
560 plot(Tid,L1,'--k')
561 hold off;

```

```

562 legend('Probe (0.974 0 -0.5)- Tee-junction','TV');
563 xlabel('Time [s]');ylabel('Temperature [^oC]');
564
565 %% Result
566 y0 = Dim/2;
567 y1 = 0.5*Di+y0
568 y  = Di+y0
569 y2 = 1.5*Di+y0
570 y3 = 2*Di+y0
571 y4 = 2.5*Di+y0
572 y5 = 3*Di+y0
573 y6 = 4*Di+y0
574 y7 = 5*Di+y0
575 y8 = 6*Di+y0
576 y9 = 7*Di+y0
577 y10= 8*Di+y0
578 y11= 9*Di+y0
579
580 %vertical velocity
581 D05=xlsread('zvel.xls','A2:A102'); %y1
582 D1=xlsread('zvel.xls','B2:B102'); %y2
583 D15=xlsread('zvel.xls','C2:C102'); %y3
584 D2=xlsread('zvel.xls','D2:D102'); %y4
585 D25=xlsread('zvel.xls','E2:E102'); %y5
586 D3=xlsread('zvel.xls','F2:F102'); %y6
587 D4=xlsread('zvel.xls','G2:G102'); %y7
588 D5=xlsread('zvel.xls','H2:H102'); %y8
589 D6=xlsread('zvel.xls','I2:I102'); %y9
590 D7=xlsread('zvel.xls','J2:J102'); %y10
591 D8=xlsread('zvel.xls','K2:K102'); %y10
592 D9=xlsread('zvel.xls','L2:L102'); %y10
593
594
595 xd=xlsread('zvel.xls','M2:M102')/Di; %x/D
596
597
598 % vertical velocity
599 figure(13)
600 plot(xd,D05,'r.','LineWidth',2);
601 hold on;
602 plot(xd,D15,'b—','LineWidth',2);
603 hold on;
604 plot(xd,D2,'gx','LineWidth',2);
605 hold on;
606 plot(xd,D25,'m+','LineWidth',2);
607 hold on;
608 plot(xd,D3,'c-.','LineWidth',2);
609 hold on;
610 plot(xd,D4,'k','LineWidth',2);
611 hold on;
612 plot(xd,D5,'g','LineWidth',2);
613 hold on;

```

```

614 plot(xd,D6,'b','LineWidth',2);
615 hold on;
616 plot(xd,D7,'k','LineWidth',2);
617 hold on;
618 plot(xd,D8,'r--','LineWidth',2);
619 hold on;
620 plot(xd,D9,'b--','LineWidth',2);
621 hold off;
622 legend('z=0.5D','z=1.5D','z=2D','z=2.5D','z=3D',...
623        'z=4D','z=5D','z=6D','z=7D','z=8D','z=9D');
624 xlabel('x/D_i');ylabel('Vertical Velocity [m/s]');
625
626 %% U1
627 %vertical velocity
628 D05=xlsread('U1/zvel.xls','A2:A102'); %y1
629 D1=xlsread('U1/zvel.xls','B2:B102'); %y2
630 D2=xlsread('U1/zvel.xls','C2:C102'); %y3
631 D3=xlsread('U1/zvel.xls','D2:D102'); %y4
632 D4=xlsread('U1/zvel.xls','E2:E102'); %y5
633 D5=xlsread('U1/zvel.xls','F2:F102'); %y6
634 D6=xlsread('U1/zvel.xls','G2:G102'); %y7
635 D7=xlsread('U1/zvel.xls','H2:H102'); %y8
636 D8=xlsread('U1/zvel.xls','I2:I102'); %y9
637 D9=xlsread('U1/zvel.xls','J2:J102'); %y10
638
639
640
641 xd=xlsread('zvel.xls','M2:M102')/Di; %x/D
642
643
644 % vertical velocity
645 figure(13)
646 plot(xd,D05,'r.','LineWidth',2);
647 hold on;
648 plot(xd,D1,'b--','LineWidth',2);
649 hold on;
650 plot(xd,D2,'gx','LineWidth',2);
651 hold on;
652 plot(xd,D3,'c-.','LineWidth',2);
653 hold on;
654 plot(xd,D4,'k','LineWidth',2);
655 hold on;
656 plot(xd,D5,'g','LineWidth',2);
657 hold on;
658 plot(xd,D6,'b','LineWidth',2);
659 hold on;
660 plot(xd,D7,'k','LineWidth',2);
661 hold on;
662 plot(xd,D8,'r--','LineWidth',2);
663 hold on;
664 plot(xd,D9,'b--','LineWidth',2);
665 hold off;

```

```
666 legend('z=0.5D','z=1D','z=2D','z=3D',...  
667         'z=4D','z=5D','z=6D','z=7D','z=8D','z=9D');  
668 xlabel('x/D_i');ylabel('Vertical Velocity [m/s]');
```

G. Solver Library

G.1. LaplacianFoam

```
1  /*-----*\
2  ===== |
3  \\      / F ield      | OpenFOAM: The Open Source CFD Toolbox
4  \\      / O peration  |
5  \\      / A nd        | Copyright (C) 2011 OpenFOAM Foundation
6  \\//     M anipulation |
7  -----*\
8  License
9      This file is part of OpenFOAM.
10
11     OpenFOAM is free software: you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.
23
24  Application
25      laplacianFoam
26
27  Description
28      Solves a simple Laplace equation, e.g. for thermal diffusion in a solid.
29
30  \*-----*\
31
32  #include "fvCFD.H"
33  #include "simpleControl.H"
34
35  // * * * * *
36
37  int main(int argc, char *argv[])
```

```

38 {
39     #include "setRootCase.H"
40
41     #include "createTime.H"
42     #include "createMesh.H"
43     #include "createFields.H"
44
45     simpleControl simple(mesh);
46
47     // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
48
49     Info<< "\nCalculating temperature distribution\n" << endl;
50
51     while (simple.loop())
52     {
53         Info<< "Time = " << runTime.timeName() << nl << endl;
54
55         while (simple.correctNonOrthogonal())
56         {
57             solve
58             (
59                 fvm::ddt(T) - fvm::laplacian(DT, T)
60             );
61         }
62
63         #include "write.H"
64
65         Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
66             << "   ClockTime = " << runTime.elapsedClockTime() << " s"
67             << nl << endl;
68     }
69
70     Info<< "End\n" << endl;
71
72     return 0;
73 }
74
75
76 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

```

G.2. PotentialFoam

```

1 /*-----*
2  ===== |
3  \ \      /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox
4  \ \      /  O p e r a t i o n  |
5  \ \      /  A n d           | Copyright (C) 2011 OpenFOAM Foundation
6  \ \      /  M a n i p u l a t i o n  |
7  -----*

```



```

8 License
9     This file is part of OpenFOAM.
10
11     OpenFOAM is free software: you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.
23
24 Application
25     potentialFoam
26
27 Description
28     Simple potential flow solver which can be used to generate starting fields
29     for full Navier–Stokes codes.
30
31 \*-----*/
32
33 #include "fvCFD.H"
34
35
36 // * * * * *
37
38 int main(int argc, char *argv[])
39 {
40     argList::addBoolOption("writep", "write the final pressure field");
41     argList::addBoolOption
42     (
43         "initialiseUBCs",
44         "initialise U boundary conditions"
45     );
46
47     #include "setRootCase.H"
48     #include "createTime.H"
49     #include "createMesh.H"
50     #include "readControls.H"
51     #include "createFields.H"
52
53     // * * * * *
54
55     Info<< nl << "Calculating potential flow" << endl;
56
57     // Since solver contains no time loop it would never execute
58     // function objects so do it ourselves.
59     runTime.functionObjects().start();

```

```

60
61 adjustPhi(phi, U, p);
62
63 for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
64 {
65     fvScalarMatrix pEqn
66     (
67         fvm::laplacian
68         (
69             dimensionedScalar
70             (
71                 "1",
72                 dimTime/p.dimensions()*dimensionSet(0, 2, -2, 0, 0),
73                 1
74             ),
75             p
76         )
77         ==
78         fvc::div(phi)
79     );
80
81     pEqn.setReference(pRefCell, pRefValue);
82     pEqn.solve();
83
84     if (nonOrth == nNonOrthCorr)
85     {
86         phi -= pEqn.flux();
87     }
88 }
89
90 Info<< "continuity error = "
91     << mag(fvc::div(phi))().weightedAverage(mesh.V()).value()
92     << endl;
93
94 U = fvc::reconstruct(phi);
95 U.correctBoundaryConditions();
96
97 Info<< "Interpolated U error = "
98     << (sqrt(sum(sqr((fvc::interpolate(U) & mesh.Sf()) - phi)))
99         /sum(mesh.magSf()))).value()
100     << endl;
101
102 // Force the write
103 U.write();
104 phi.write();
105
106 if (args.optionFound("writep"))
107 {
108     p.write();
109 }
110
111 runTime.functionObjects().end();

```

```

112
113     Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
114         << "   ClockTime = " << runTime.elapsedClockTime() << " s"
115         << nl << endl;
116
117     Info<< "End\n" << endl;
118
119     return 0;
120 }
121
122
123 // ***** //

```

G.3. BuoyantBoussinesqSimpleFoam

```

1  /*-----*\
2  ===== |
3  \ \      /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox
4  \ \      /  O p e r a t i o n  |
5  \ \      /  A n d            | Copyright (C) 2011 OpenFOAM Foundation
6  \ \ /      M a n i p u l a t i o n  |
7  -----*\
8  License
9      This file is part of OpenFOAM.
10
11     OpenFOAM is free software: you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.
23
24  Application
25      buoyantBoussinesqSimpleFoam
26
27  Description
28      Steady-state solver for buoyant, turbulent flow of incompressible fluids
29
30      Uses the Boussinesq approximation:
31      \f[
32          rho_{eff} = 1 - beta(T - T_{ref})
33      \f]
34

```

```

35     where:
36         \f$ rho_{eff} \f$ = the effective (driving) density
37         beta = thermal expansion coefficient [1/K]
38         T = temperature [K]
39         \f$ T_{ref} \f$ = reference temperature [K]
40
41     Valid when:
42     \f[
43         rho_{eff} << 1
44     \f]
45
46     \*-----*/
47
48     #include "fvCFD.H"
49     #include "singlePhaseTransportModel.H"
50     #include "RASModel.H"
51     #include "simpleControl.H"
52
53     // * * * * *
54
55     int main(int argc, char *argv[])
56     {
57         #include "setRootCase.H"
58         #include "createTime.H"
59         #include "createMesh.H"
60         #include "readGravitationalAcceleration.H"
61         #include "createFields.H"
62         #include "initContinuityErrs.H"
63
64         simpleControl simple(mesh);
65
66         // * * * * *
67
68         Info<< "\nStarting time loop\n" << endl;
69
70         while (simple.loop())
71         {
72             Info<< "Time = " << runTime.timeName() << nl << endl;
73
74             // Pressure-velocity SIMPLE corrector
75             {
76                 #include "UEqn.H"
77                 #include "TEqn.H"
78                 #include "pEqn.H"
79             }
80
81             turbulence->correct();
82
83             runTime.write();
84
85             Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
86                 << " ClockTime = " << runTime.elapsedClockTime() << " s"

```

```

87         << nl << endl;
88     }
89
90     Info<< "End\n" << endl;
91
92     return 0;
93 }
94
95
96 // ***** //

```

G.3.1. Velocity equation

```

1     // Solve the momentum equation
2
3     tmp<fvVectorMatrix> UEqn
4     (
5         fvm::div(phi, U)
6         + turbulence->divDevReff(U)
7     );
8
9     UEqn().relax();
10
11     if (simple.momentumPredictor())
12     {
13         solve
14         (
15             UEqn()
16             ==
17             fvc::reconstruct
18             (
19                 (
20                     - ghf*fvc::snGrad(rhok)
21                     - fvc::snGrad(p_rgh)
22                 ) * mesh.magSf()
23             )
24         );
25     }

```

G.3.2. Temperature equation

```

1 {
2     kappat = turbulence->nut()/Prt;
3     kappat.correctBoundaryConditions();
4
5     volScalarField kappaEff("kappaEff", turbulence->nu()/Pr + kappat);
6
7     fvScalarMatrix TEqn

```

```

8      (
9          fvm::div(phi, T)
10         - fvm::Sp(fvc::div(phi), T)
11         - fvm::laplacian(kappaEff, T)
12     );
13
14     TEqn.relax();
15     TEqn.solve();
16
17     rhok = 1.0 - beta*(T - TRef);
18 }

```

G.3.3. Pressure equation

```

1  {
2      volScalarField rAU("rAU", 1.0/UEqn().A());
3      surfaceScalarField rAUf("1|A(U)", fvc::interpolate(rAU));
4
5      U = rAU*UEqn().H();
6      UEqn.clear();
7
8      phi = fvc::interpolate(U) & mesh.Sf();
9      adjustPhi(phi, U, p_rgh);
10
11     surfaceScalarField buoyancyPhi(rAUf*ghf*fvc::snGrad(rhok)*mesh.magSf());
12     phi -= buoyancyPhi;
13
14     while (simple.correctNonOrthogonal())
15     {
16         fvScalarMatrix p_rghEqn
17         (
18             fvm::laplacian(rAUf, p_rgh) == fvc::div(phi)
19         );
20
21         p_rghEqn.setReference(pRefCell, getRefCellValue(p_rgh, pRefCell));
22
23         p_rghEqn.solve();
24
25         if (simple.finalNonOrthogonalIter())
26         {
27             // Calculate the conservative fluxes
28             phi -= p_rghEqn.flux();
29
30             // Explicitly relax pressure for momentum corrector
31             p_rgh.relax();
32
33             // Correct the momentum source with the pressure gradient flux
34             // calculated from the relaxed pressure
35             U -= rAU*fvc::reconstruct((buoyancyPhi + p_rghEqn.flux())/rAUf);
36             U.correctBoundaryConditions();

```

```

37     }
38 }
39
40 #include "continuityErrs.H"
41
42 p = p_rgh + rhok*gh;
43
44 if (p_rgh.needReference())
45 {
46     p += dimensionedScalar
47         (
48             "p",
49             p.dimensions(),
50             pRefValue - getRefCellValue(p, pRefCell)
51         );
52     p_rgh = p - rhok*gh;
53 }
54 }

```

G.3.4. Transport Properties

```

1     singlePhaseTransportModel laminarTransport(U, phi);
2
3     // Thermal expansion coefficient [1/K]
4     dimensionedScalar beta(laminarTransport.lookup("beta"));
5
6     // Reference temperature [K]
7     dimensionedScalar TRef(laminarTransport.lookup("TRef"));
8
9     // Laminar Prandtl number
10    dimensionedScalar Pr(laminarTransport.lookup("Pr"));
11
12    // Turbulent Prandtl number
13    dimensionedScalar Prt(laminarTransport.lookup("Prt"));

```

G.3.5. Turbulence Library

```

1  /*-----*\
2  ===== |
3  \ \ / /   F i e l d       | OpenFOAM: The Open Source CFD Toolbox
4  \ \ / /   O p e r a t i o n |
5  \ \ / /   A n d           | Copyright (C) 2011 OpenFOAM Foundation
6  \ \ / /   M a n i p u l a t i o n |
7  -----*\
8  License
9      This file is part of OpenFOAM.
10
11     OpenFOAM is free software: you can redistribute it and/or modify it

```

```

12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.
23
24  \*-----*/
25
26  #include "turbulenceModel.H"
27  #include "volFields.H"
28  #include "surfaceFields.H"
29
30  // * * * * *
31
32  namespace Foam
33  {
34  namespace incompressible
35  {
36
37  // * * * * *
38
39  defineTypeNameAndDebug(turbulenceModel, 0);
40  defineRunTimeSelectionTable(turbulenceModel, turbulenceModel);
41
42  // * * * * * Constructors * * * * *
43
44  turbulenceModel::turbulenceModel
45  (
46      const volVectorField& U,
47      const surfaceScalarField& phi,
48      transportModel& transport,
49      const word& turbulenceModelName
50  )
51  :
52      regIOobject
53      (
54          IOobject
55          (
56              turbulenceModelName,
57              U.time().constant(),
58              U.db(),
59              IOobject::NO_READ,
60              IOobject::NO_WRITE
61          )
62      ),
63      runTime_(U.time()),

```



```

64     mesh_(U.mesh()),
65
66     U_(U),
67     phi_(phi),
68     transportModel_(transport)
69 {}
70
71
72 // * * * * * Selectors * * * * * //
73
74 autoPtr<turbulenceModel> turbulenceModel::New
75 (
76     const volVectorField& U,
77     const surfaceScalarField& phi,
78     transportModel& transport,
79     const word& turbulenceModelName
80 )
81 {
82     // get model name, but do not register the dictionary
83     // otherwise it is registered in the database twice
84     const word modelType
85     (
86         IOdictionary
87         (
88             IOobject
89             (
90                 "turbulenceProperties",
91                 U.time().constant(),
92                 U.db(),
93                 IOobject::MUST_READ_IF_MODIFIED,
94                 IOobject::NO_WRITE,
95                 false
96             )
97         ).lookup("simulationType")
98     );
99
100     Info<< "Selecting turbulence model type " << modelType << endl;
101
102     turbulenceModelConstructorTable::iterator cstrIter =
103         turbulenceModelConstructorTablePtr_>find(modelType);
104
105     if (cstrIter == turbulenceModelConstructorTablePtr_>end())
106     {
107         FatalErrorIn
108         (
109             "turbulenceModel::New(const volVectorField&, "
110             "const surfaceScalarField&, transportModel&, const word&)"
111         ) << "Unknown turbulenceModel type "
112         << modelType << nl << nl
113         << "Valid turbulenceModel types:" << endl
114         << turbulenceModelConstructorTablePtr_>sortedToc()
115         << exit(FatalError);

```

```

116     }
117
118     return autoPtr<turbulenceModel>
119     (
120         cstrIter()(U, phi, transport, turbulenceModelName)
121     );
122 }
123
124
125 // * * * * * Member Functions * * * * * //
126
127 void turbulenceModel::correct()
128 {
129     transportModel_.correct();
130 }
131
132
133 // * * * * * //
134
135 } // End namespace incompressible
136 } // End namespace Foam
137
138 // ***** //

```

```

1 /*-----*\
2  ===== |
3  \\      /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox
4  \\      /  O p e r a t i o n  |
5  \\      /  A n d      | Copyright (C) 2011 OpenFOAM Foundation
6  \\//      M a n i p u l a t i o n  |
7  -----*
8 License
9     This file is part of OpenFOAM.
10
11     OpenFOAM is free software: you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.
23
24 Namespace
25     Foam::incompressible::turbulenceModels
26
27 Description

```

```

28     Namespace for incompressible turbulence turbulence models.
29
30 Class
31     Foam::incompressible::turbulenceModel
32
33 Description
34     Abstract base class for incompressible turbulence models
35     (RAS, LES and laminar).
36
37 SourceFiles
38     turbulenceModel.C
39     newTurbulenceModel.C
40
41 \*-----*/
42
43 #ifndef turbulenceModel_H
44 #define turbulenceModel_H
45
46 #include "primitiveFieldsFwd.H"
47 #include "volFieldsFwd.H"
48 #include "surfaceFieldsFwd.H"
49 #include "fvMatricesFwd.H"
50 #include "incompressible/transportModel/transportModel.H"
51 #include "autoPtr.H"
52 #include "runTimeSelectionTables.H"
53
54 // * * * * *
55
56 namespace Foam
57 {
58
59 // Forward declarations
60 class fvMesh;
61
62 namespace incompressible
63 {
64
65 \*-----*\
66                                     Class turbulenceModel Declaration
67 \*-----*/
68
69 class turbulenceModel
70 :
71     public regIOobject
72 {
73
74 protected:
75
76     // Protected data
77
78     const Time& runTime_;
79     const fvMesh& mesh_;

```

```

80
81     const volVectorField& U_;
82     const surfaceScalarField& phi_;
83
84     transportModel& transportModel_;
85
86
87 private:
88
89     // Private Member Functions
90
91     //-- Disallow default bitwise copy construct
92     turbulenceModel(const turbulenceModel&);
93
94     //-- Disallow default bitwise assignment
95     void operator=(const turbulenceModel&);
96
97
98 public:
99
100    //-- Runtime type information
101    TypeName("turbulenceModel");
102
103
104    // Declare run-time New selection table
105
106    declareRunTimeNewSelectionTable
107    (
108        autoPtr,
109        turbulenceModel,
110        turbulenceModel,
111        (
112            const volVectorField& U,
113            const surfaceScalarField& phi,
114            transportModel& transport,
115            const word& turbulencemodelName
116        ),
117        (U, phi, transport, turbulencemodelName)
118    );
119
120
121    // Constructors
122
123    //-- Construct from components
124    turbulenceModel
125    (
126        const volVectorField& U,
127        const surfaceScalarField& phi,
128        transportModel& transport,
129        const word& turbulencemodelName = typeName
130    );
131

```

```

132
133 // Selectors
134
135 //– Return a reference to the selected turbulence model
136 static autoPtr<turbulenceModel> New
137 (
138     const volVectorField& U,
139     const surfaceScalarField& phi,
140     transportModel& transport,
141     const word& turbulenceModelName = typeName
142 );
143
144
145 //– Destructor
146 virtual ~turbulenceModel()
147 {}
148
149
150 // Member Functions
151
152 //– Access function to velocity field
153 inline const volVectorField& U() const
154 {
155     return U_;
156 }
157
158 //– Access function to flux field
159 inline const surfaceScalarField& phi() const
160 {
161     return phi_;
162 }
163
164 //– Access function to incompressible transport model
165 inline transportModel& transport() const
166 {
167     return transportModel_;
168 }
169
170 //– Return the laminar viscosity
171 inline tmp<volScalarField> nu() const
172 {
173     return transportModel_.nu();
174 }
175
176 //– Return the turbulence viscosity
177 virtual tmp<volScalarField> nut() const = 0;
178
179 //– Return the effective viscosity
180 virtual tmp<volScalarField> nuEff() const = 0;
181
182 //– Return the turbulence kinetic energy
183 virtual tmp<volScalarField> k() const = 0;

```

```

184
185     //- Return the turbulence kinetic energy dissipation rate
186     virtual tmp<volScalarField> epsilon() const = 0;
187
188     //- Return the Reynolds stress tensor
189     virtual tmp<volSymmTensorField> R() const = 0;
190
191     //- Return the effective stress tensor including the laminar stress
192     virtual tmp<volSymmTensorField> devReff() const = 0;
193
194     //- Return the source term for the momentum equation
195     virtual tmp<fvVectorMatrix> divDevReff(volVectorField& U) const = 0;
196
197     //- Solve the turbulence equations and correct the turbulence viscosity
198     virtual void correct() = 0;
199
200     //- Read LESProperties or RASProperties dictionary
201     virtual bool read() = 0;
202
203     //- Default dummy write function
204     virtual bool writeData(Ostream&) const
205     {
206         return true;
207     }
208 };
209
210
211 // * * * * *
212
213 } // End namespace incompressible
214 } // End namespace Foam
215
216 // * * * * *
217
218 #endif
219
220 // *****

```

K- ϵ model

```

1 /*-----*\
2  ===== |
3  \ \      / F i e l d      | OpenFOAM: The Open Source CFD Toolbox
4  \ \      / O p e r a t i o n |
5  \ \      / A n d           | Copyright (C) 2011 OpenFOAM Foundation
6  \ \      / M a n i p u l a t i o n |
7  -----*
8 License
9     This file is part of OpenFOAM.
10

```

```

11     OpenFOAM is free software: you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
23
24     \*-----*/
25
26     #include "kEpsilon.H"
27     #include "addToRunTimeSelectionTable.H"
28
29     #include "backwardsCompatibilityWallFunctions.H"
30
31     // * * * * *
32
33     namespace Foam
34     {
35     namespace incompressible
36     {
37     namespace RASModels
38     {
39
40     // * * * * * Static Data Members * * * * *
41
42     defineTypeNameAndDebug(kEpsilon, 0);
43     addToRunTimeSelectionTable(RASModel, kEpsilon, dictionary);
44
45     // * * * * * Constructors * * * * *
46
47     kEpsilon::kEpsilon
48     (
49         const volVectorField& U,
50         const surfaceScalarField& phi,
51         transportModel& transport,
52         const word& turbulenceModelName,
53         const word& modelName
54     )
55     :
56     RASModel(modelName, U, phi, transport, turbulenceModelName),
57
58     Cmu_
59     (
60         dimensioned<scalar>::lookupOrAddToDict
61         (
62             "Cmu",

```

```

63         coeffDict_,
64         0.09
65     )
66 ),
67 C1_
68 (
69     dimensioned<scalar>::lookupOrAddToDict
70     (
71         "C1",
72         coeffDict_,
73         1.44
74     )
75 ),
76 C2_
77 (
78     dimensioned<scalar>::lookupOrAddToDict
79     (
80         "C2",
81         coeffDict_,
82         1.92
83     )
84 ),
85 sigmaEps_
86 (
87     dimensioned<scalar>::lookupOrAddToDict
88     (
89         "sigmaEps",
90         coeffDict_,
91         1.3
92     )
93 ),
94
95 k_
96 (
97     IOobject
98     (
99         "k",
100        runtime_.timeName(),
101        mesh_,
102        IOobject::NO_READ,
103        IOobject::AUTO_WRITE
104    ),
105    autoCreateK("k", mesh_)
106 ),
107 epsilon_
108 (
109     IOobject
110     (
111         "epsilon",
112        runtime_.timeName(),
113        mesh_,
114        IOobject::NO_READ,

```



```

115         IOobject::AUTO_WRITE
116     ),
117     autoCreateEpsilon("epsilon", mesh_)
118 ),
119 nut_
120 (
121     IOobject
122     (
123         "nut",
124         runTime_.timeName(),
125         mesh_,
126         IOobject::NO_READ,
127         IOobject::AUTO_WRITE
128     ),
129     autoCreateNut("nut", mesh_)
130 )
131 {
132     bound(k_, kMin_);
133     bound(epsilon_, epsilonMin_);
134
135     nut_ = Cmu_*sqr(k_)/epsilon_;
136     nut_.correctBoundaryConditions();
137
138     printCoeffs();
139 }
140
141
142 // * * * * * Member Functions * * * * * //
143
144 tmp<volSymmTensorField> kEpsilon::R() const
145 {
146     return tmp<volSymmTensorField>
147     (
148         new volSymmTensorField
149         (
150             IOobject
151             (
152                 "R",
153                 runTime_.timeName(),
154                 mesh_,
155                 IOobject::NO_READ,
156                 IOobject::NO_WRITE
157             ),
158             ((2.0/3.0)*I)*k_ - nut_*twoSymm(fvc::grad(U_)),
159             k_.boundaryField().types()
160         )
161     );
162 }
163
164
165 tmp<volSymmTensorField> kEpsilon::devReff() const
166 {

```

```

167     return tmp<volSymmTensorField>
168     (
169         new volSymmTensorField
170         (
171             IOobject
172             (
173                 "devRhoReff",
174                 runTime_.timeName(),
175                 mesh_,
176                 IOobject::NO_READ,
177                 IOobject::NO_WRITE
178             ),
179             -nuEff() * dev(twoSymm(fvc::grad(U_)))
180         )
181     );
182 }
183
184
185 tmp<fvVectorMatrix> kEpsilon::divDevReff(volVectorField& U) const
186 {
187     return
188     (
189         - fvm::laplacian(nuEff(), U)
190         - fvc::div(nuEff() * dev(T(fvc::grad(U))))
191     );
192 }
193
194
195 bool kEpsilon::read()
196 {
197     if (RASModel::read())
198     {
199         Cmu_.readIfPresent(coeffDict());
200         C1_.readIfPresent(coeffDict());
201         C2_.readIfPresent(coeffDict());
202         sigmaEps_.readIfPresent(coeffDict());
203
204         return true;
205     }
206     else
207     {
208         return false;
209     }
210 }
211
212
213 void kEpsilon::correct()
214 {
215     RASModel::correct();
216
217     if (!turbulence_)
218     {

```

```

219         return;
220     }
221
222     volScalarField G("RASModel::G", nut_*2*magSqr(symm(fvc::grad(U_))));
223
224     // Update epsilon and G at the wall
225     epsilon_.boundaryField().updateCoeffs();
226
227     // Dissipation equation
228     tmp<fvScalarMatrix> epsEqn
229     (
230         fvm::ddt(epsilon_)
231         + fvm::div(phi_, epsilon_)
232         - fvm::Sp(fvc::div(phi_), epsilon_)
233         - fvm::laplacian(DepsilonEff(), epsilon_)
234         ==
235         C1_*G*epsilon_/k_
236         - fvm::Sp(C2_*epsilon_/k_, epsilon_)
237     );
238
239     epsEqn().relax();
240
241     epsEqn().boundaryManipulate(epsilon_.boundaryField());
242
243     solve(epsEqn);
244     bound(epsilon_, epsilonMin_);
245
246
247     // Turbulent kinetic energy equation
248     tmp<fvScalarMatrix> kEqn
249     (
250         fvm::ddt(k_)
251         + fvm::div(phi_, k_)
252         - fvm::Sp(fvc::div(phi_), k_)
253         - fvm::laplacian(DkEff(), k_)
254         ==
255         G
256         - fvm::Sp(epsilon_/k_, k_)
257     );
258
259     kEqn().relax();
260     solve(kEqn);
261     bound(k_, kMin_);
262
263
264     // Re-calculate viscosity
265     nut_ = Cmu_*sqr(k_)/epsilon_;
266     nut_.correctBoundaryConditions();
267 }
268
269
270 // * * * * *

```

```

271
272 } // End namespace RASModels
273 } // End namespace incompressible
274 } // End namespace Foam
275
276 // ***** //

1 /*-----*\
2 ===== |
3  \ \      /  F i e l d      | OpenFOAM: The Open Source CFD Toolbox
4  \ \      /  O peration     |
5   \ \      /  A nd          | Copyright (C) 2011 OpenFOAM Foundation
6    \ \      /  M anipulation |
7 -----*
8 License
9     This file is part of OpenFOAM.
10
11     OpenFOAM is free software: you can redistribute it and/or modify it
12     under the terms of the GNU General Public License as published by
13     the Free Software Foundation, either version 3 of the License, or
14     (at your option) any later version.
15
16     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
17     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
18     FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License
19     for more details.
20
21     You should have received a copy of the GNU General Public License
22     along with OpenFOAM. If not, see <http://www.gnu.org/licenses/>.
23
24 Class
25     Foam::incompressible::RASModels::kEpsilon
26
27 Description
28     Standard k-epsilon turbulence model for incompressible flows.
29
30     The default model coefficients correspond to the following:
31     \verbatim
32         kEpsilonCoeffs
33         {
34             Cmu          0.09;
35             C1           1.44;
36             C2           1.92;
37             sigmaEps     1.3;
38         }
39     \endverbatim
40
41 SourceFiles
42     kEpsilon.C
43
44 /*-----*/

```

```

45
46 #ifndef kEpsilon_H
47 #define kEpsilon_H
48
49 #include "RASModel.H"
50
51 // * * * * *
52
53 namespace Foam
54 {
55     namespace incompressible
56     {
57         namespace RASModels
58         {
59
60             /*-----*\
61                          Class kEpsilon Declaration
62             \*-----*/
63
64             class kEpsilon
65             :
66                 public RASModel
67             {
68
69             protected:
70
71                 // Protected data
72
73                 // Model coefficients
74
75                 dimensionedScalar Cmu_;
76                 dimensionedScalar C1_;
77                 dimensionedScalar C2_;
78                 dimensionedScalar sigmaEps_;
79
80
81                 // Fields
82
83                 volScalarField k_;
84                 volScalarField epsilon_;
85                 volScalarField nut_;
86
87
88             public:
89
90                 //-- Runtime type information
91                 TypeName("kEpsilon");
92
93                 // Constructors
94
95                 //-- Construct from components
96                 kEpsilon

```

```

97         (
98             const volVectorField& U,
99             const surfaceScalarField& phi,
100            transportModel& transport,
101            const word& turbulenceModelName = turbulenceModel::typeName,
102            const word& modelName = typeName
103        );
104
105
106    //- Destructor
107    virtual ~kEpsilon()
108    {}
109
110
111    // Member Functions
112
113    //- Return the turbulence viscosity
114    virtual tmp<volScalarField> nut() const
115    {
116        return nut_;
117    }
118
119    //- Return the effective diffusivity for k
120    tmp<volScalarField> DkEff() const
121    {
122        return tmp<volScalarField>
123            (
124                new volScalarField("DkEff", nut_ + nu())
125            );
126    }
127
128    //- Return the effective diffusivity for epsilon
129    tmp<volScalarField> DepsilonEff() const
130    {
131        return tmp<volScalarField>
132            (
133                new volScalarField("DepsilonEff", nut_/sigmaEps_ + nu())
134            );
135    }
136
137    //- Return the turbulence kinetic energy
138    virtual tmp<volScalarField> k() const
139    {
140        return k_;
141    }
142
143    //- Return the turbulence kinetic energy dissipation rate
144    virtual tmp<volScalarField> epsilon() const
145    {
146        return epsilon_;
147    }
148

```

```

149     //- Return the Reynolds stress tensor
150     virtual tmp<volSymmTensorField> R() const;
151
152     //- Return the effective stress tensor including the laminar stress
153     virtual tmp<volSymmTensorField> devReff() const;
154
155     //- Return the source term for the momentum equation
156     virtual tmp<fvVectorMatrix> divDevReff(volVectorField& U) const;
157
158     //- Solve the turbulence equations and correct the turbulence viscosity
159     virtual void correct();
160
161     //- Read RASProperties dictionary
162     virtual bool read();
163 };
164
165
166 // * * * * *
167
168 } // End namespace RASModels
169 } // End namespace incompressible
170 } // End namespace Foam
171
172 // * * * * *
173
174 #endif
175
176 // *****

```


H. Procedure to run the case files in OpenFOAM

- 1 Open Appendix I. There are one folder for each case.
- 2 Case 01, there are again two folders, a and b. These indicate the two wind velocity cases. Further there are two folders within each folder. They are named pipe1 and pipe2 which is the steady-state and transient simulation respectively.
- 3 Case 02, there are two folders, one to initialize the potential flow field and one to execute the heat transfer and fluid flow simulation.
- 4 Case 03, contains the simulation files directly.
- 5 To calculate the initial values set in the 0 -folder, the Matlab script in Appendix F should be executed. In the script all needed fluid variables are stated, this creates the possibility to change medium. Additionally, the velocities of the external and internal flow may be set as desired. Consequently, the parameters dependent will change automatically when executing the script.
- 6 Insert the initial values to the 0 -folder in the case folders. If the same grid is used no change in the other files are necessary before an execution may be performed.
- 7 Wait for OpenFOAM to finish.
- 8 Open paraView and post-process the results as desired.

If a new geometry is desired, this may be made in Salome. To make it simple to adapt the developed solver make sure you name the boundaries, inlet, outlet, wallsM, wallsD and deadend. If you do so, then the only thing you have to do is to run the .unv file and change the base patches for the wall boundaries and update the initial values if desired.

I. Case Files