Magnus Johannesen

Automated Prediction of Fractional Flow Reserve through Numerical Simulation of Coronary Artery Flow

Master’s thesis in Mechanical Engineering
Supervisor: Reidar Kristoffersen, Fredrik Eikeland Fossan
June 2019

Magnus Johannesen

Automated Prediction of Fractional Flow Reserve through Numerical Simulation of Coronary Artery Flow

Master's thesis in Mechanical Engineering
Supervisor: Reidar Kristoffersen, Fredrik Eikeland Fossan
June 2019

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

Preface

With this thesis, I can finally conclude my six years at NTNU with a thesis that I have grown quite proud of. After spending countless hours not believing the method will ever work, it is lovely to be able to hand in this work, knowing it finally did. Working with biomechanics has really been a fun journey, and I hope that I will be able to work with similar challenges some time in the future as well!

I would like to thank my supervisors Associate Professor Reidar Kristoffersen and PhD Candidate Fredrik Eikeland Fossan for all the help I've gotten during this work. You have both been very helpful in showing me the way, and also allowing me to discuss ideas to improve my understanding of the subjects. A special thanks also to Assistant Professor Lucas Omar Mueller that supervised the project and followed my thesis until the end of February.

Lastly, I would like to thank my family and Sigrid for encouraging me and helping me during this work. Even though I sometimes digress away into student politics you have helped me focus on the important things in life and guiding me to the completion of this work.

Trondheim, 09.06.2019
Magnus Johannesen

Abstract

In this thesis, a method for determining Fractional Flow Reserve for quantification of functional reduction in human coronary artery trees have been developed. The method has been produced in the Computational Fluid Dynamics software ANSYS Fluent and is specialised to handle reconstructed meshes based on Computed Tomography imaging. The goal was to produce similar results as previously done in a semi-transient Finite Element Solver, but using Finite Volume Elements and steady-state conditions. With a resistance analogy representing the hyperemic conditions of the fluid flow in coronary artery trees. A linearly increasing bias is observed when trying to simulate values lower then the cut-off value of 0.8, with a maximum observed error at 0.059. The method is performing well in 77 out of 78 available domains and with sufficiently accurate results to be used for further research in diagnostic tools of Coronary Artery Disease.

Sammendrag

I denne masteroppgaven har det blitt utviklet en metode for å bestemme ”Fractional Flow Reserve” indeksen for kvantifisering av funksjonell reduksjon i menneskelige kransarterier. Metoden har benyttet seg av den strømningstekniske programvaren ANSYS Fluent, og er spesialisert for å håndtere rekonstruerte domener basert på CT bilder. Målet har vært å reprodusere resultatene fra en semi-transient løser basert på elementmetoden, men ved bruk av en tidsuavhengig løser basert på volummetoden. Ved å kvantifisere strømningen basert på perifer motstand har det lykkes i å reprodusere gode resultater i 77 ut av 78 tilgjengelige domener. En lineært økende skjevhet er observert ved verdier under grensepunktet på 0.8, med en maksimal feil på 0.059, men metoden er likevel treffsikker nok til at den kan benyttes videre i forskning på diagnosemetoder for koronarsykdom.

Table of Contents

List of Figures

List of Tables

Abbreviations

1D-0D	=	Reduced order 1 to 0 Dimensional
2D	=	Two Dimensional
3D	=	Three Dimensional
ANSYS	=	Software package for numerical simulations
BM	=	Reasearch group of Biomechanics
CABG	=	Coronary Artery Bypass Graft
CAD	=	Coronary Artery Disease
CCTA	=	Coronary Computed Tomography Angiography
CFD	=	Computational Fluid Dynamics
CFFR	=	Computational Fractional Flow Reserve
CO	=	Cardiac Output
CT	=	Computed Tomography
FEM	=	Finite Element Method
FFR	=	Fractional Flow Reserve
FVM	=	Finite Volume Method
ITK-SNAP	=	Insight Segmentation and Registration Toolkit
HPC	=	High Performance Computing
ICA	=	Invasive Coronary Angiography
LM	=	Left Main artery
MD	=	Mean Difference
MSH	=	Fluent mesh type
NS	=	Navier-Stokes equations
OMT	=	Optimal Medical Therapy
OpenFOAM	=	Open source Field Operation And Manipulation
PCI	=	Percutaneous Coronary Intervention
Re	=	Reynolds number
RM	=	Right Main artery
SA	=	Sensitivity Analysis
SD	=	Standard Deviation
SIMPLE	=	Semi-Implicit Method for Pressure-Linked Equations
TAG	=	Transluminal Attenuation Gradient
UDF	=	User Defined Function
UQ	=	Uncertainty Quantification
VMTK	=	Vascular Modeling ToolKit
VTK	=	Visualization ToolKit

Chapter 1

Introduction

1.1 Background and motivation

In 2016 the World Health Organisation reported 56.9 million deaths in the world. Ischaemic heart disease [1] caused 9.4 million of these deaths. The most common cause for this is Coronary Artery Disease (CAD). Improvements in the diagnosis and treatment of this disease will have a massive impact on both general health and fatality rate in the population. CAD is often materialised as stenotic regions in the coronary arteries in the heart. Stenosis describes a region where the diameter of the vessel has been reduced or obstructed, either by a buildup of fat, cholesterol or other waste products. The Fractional Flow Reserve index (FFR) is considered the gold standard for diagnosing patients suffering from stable CAD [2], and gives a good indication whether the artery is supplying enough blood flow to sufficiently support the muscles of the heart or not. The index is the ratio of pressure upstream and downstream of the stenotic region, namely the arterial $\bar{P}_a$ and distal pressure $\bar{P}_d$. The measurements are obtained with a pressure wire, as can be seen in Figure 1.1. When measuring the index, one first has to induce a hyperemic state of flow, which is a state of maximum coronary flow. Finally, the measurements are averaged over a series of heart cycles as [3]

$$\text{FFR} = \frac{\bar{P}_d}{\bar{P}_a}. \tag{1.1}$$

The threshold value is 0.8 [4] and an FFR value lower than this would indicate that the artery has functionally significant stenosis, meaning further exploration of the patient is necessary to determine the correct treatment [5]. If the value is above 0.8, the standard action is to advise Optimal Medical Therapy (OMT)[1]. When OMT is the preferred action, the patient has already been through the invasive procedure of measuring coronary pressure, hence exposed to unnecessary risk and discomfort. To improve patient satisfaction and reduce the overall cost of medical procedures, the possibility of making these measurements with less invasive procedures is preferred. Here Computational FFR (CFFR) has been introduced as a very promising option [6]. Using advanced image techniques within Computed Tomography (CT), a reconstruction of the coronary arteries can be the basis of Computational Fluid Dynamics (CFD) simulations. With these simulations, one can predict the result of invasive pressure measurements with far less invasive procedures, making it possible to exclude many patients from costly and straining procedures.

Deferring patients from further surgery can mean as much as 30% reduction of cost and 12% fewer cardiac events [8]. The potential gain when applying this to a global scale is massive. Similar values have also been found by HeartFlow[2], where they report a 26% reduction of cost, though including their cost of \$1500 to produce the CFFR results.

[1]Treatment with medication or physical activity that can reduce risk factors

[2]Largest commercial actor using CFFR as a diagnostic tool

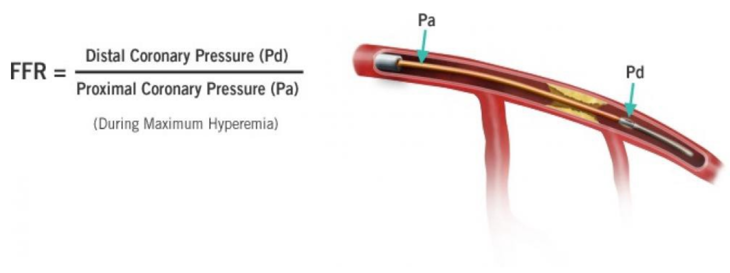

FFR = Distal Coronary Pressure (Pd) / Proximal Coronary Pressure (Pa)
(During Maximum Hyperemia)
Pa
Pd

Figure 1.1: FFR measurement with pressure wire [7].

The statistics presented earlier discusses CAD as a whole, but the usage of FFR concentrates mainly around the diagnosis and treatment of stable CAD. Here the reduction of function and increase in pain is mainly caused by obstructions. It is also differing from more acute conditions were inserting a pressure wire is more likely to cause myocardial infarctions or acute pain. Also, the likelihood of deferring someone in an acute condition is low, which reduces the potential gain significantly.

When it comes to stable CAD, the evidence base for FFR as a predictor is strong [2, 4, 9, 10, 11, 12], and both American and European guidelines for diagnosis and treatment of CAD [5, 13] have acknowledged FFR as an important diagnostic tool. Where FFR is mainly proposed as a test to check whether revascularisation [3] is the preferred action. The most common methods of revascularisation include Percutaneous Coronary Intervention (PCI)[4] and Coronary Artery Bypass Graft (CABG)[5]. These two methods are both heavily invasive and cause a greater risk of harm when performed, therefore only preferable when OMT is not an option for lasting improvement of the condition.

Introduced in 1993, FFR is still a rather new tool in the medical world. As the rundown by Pijls. et al. [14] shows, there are multiple challenges and pitfalls when trying to determine the index. It also requires somewhat expensive equipment, and skilled practitioners to ensure that the results are correct. These are some of the reasons why it has not taken preference in the medical community. A study from 2014 [15] showed that from 72% of the respondents, there was only about 1/3 of the cardiologists that used FFR to guide their decision to perform PCI surgery. The rest did not use it at all. This was backed up in 2015 [16], where FFR was reported in only 10% of cases where PCI was the resulting treatment. Showing that there is a need for simplification of the process, and CFFR can be the simple solution to the complex problem. By breaking down the different factors of coronary and myocardial physiology, a more holistic approach to the current state of the patient can be achieved. When enough knowledge about the individual physiology of the patient is available, the chances of giving a correct diagnosis increases.

1.2 Objective

There is still much work that needs to be done to have a complete understanding of how individual factors determine the uncertainty in the computational models that are being used to predict FFR.

[3]Invasive surgery to open up the artery in question or bypass the obstructed section.

[4]Inserting a stent in the artery to increase the diameter and mitigate the flow.

[5]Moving part of an artery (mainly from the leg) inserting across the stenosis

This thesis will be looking at ways of simplifying the use of CFFR by improving the knowledge of the methods, and also adding to the available tools for prediction. The project is in collaboration with the Research group of Biomechanics (BM) at NTNU who is working on implementing a model-based method for FFR determination [17]. With a reduced order model (1D-0D) FFR predictions can be performed with minimal computational effort. As current methods often require a full three dimensional (3D) transient simulations there is a lot to gain on reducing the complexity without losing the validity of the method. In their work, a 3D solver based on the Finite-Element Method (FEM) [18] has been produced and is used to validate the results from the 1D-0D solver. To enhance the validity of this solver, and possibly reduce the computational power required to perform the simulations, a steady-state solver based on the Finite Volume Method (FVM) will be utilised to solve the same problem. Results will be compared against the clinical values as well as the FEM results.

1.3 Simplifications and setup

A simple model has the above discussed benefits in terms computational time, however it is important that the simplifications does not compromise accuracy. A discussion on the relevant simplifications follows.

1.3.1 Rigid domain

The first assumption is that the domain is not moving. During one heart cycle, the muscular arteries are expanding and contracting to ease the movement of blood through the domain. In a 3D simulation, this type of fluid-structure interaction would be immensely computationally intensive and not preferable. The effect of moving artery walls have been looked into previously [19] when it comes to blood flow in the brain, and specifically related to brain aneurysms. They found that personalised methods and compliant tubes showed no difference in the resulting flow conditions. Related to FFR the same can be found when working with rigid and compliant tubes [16]. When working in 1D-0D, the radius of a tube is just a property, and changing this to accommodate the elastic effects of pressure change is much easier. However, this will not be implemented here when working in 3D.

The domain

The resolution of Computed Tomography (CT) scans limits recreation of arteries with diameter much smaller than 1 mm, which gives a natural restriction on the size of the domain. This is problematic, since it is the smaller arteries which has the ability to expand, and thus regulate the supply of coronary flow, through the resulting change in peripheral resistance. This is an important feature of coronary circulation and has to be incorporated through boundary conditions. However, studies show that the vasodilating abilities of an artery downstream of stenosis might be reduced due to the stenosis [20, 21]. While the stenosis is growing more significant, the downstream arteries are attempting to reduce the peripheral resistance by expanding in size. Therefore when attempting to dilate these arteries chemically in the clinic, they might already be experiencing maximum hyperemic conditions. Also, much of the blood flow might be redirected through collateral arteries, as will be explained in Section 2.1.1. All of these effects are attempted to be covered by the resistance analogy in Section 1.3.5, incorporating it in the peripheral resistance of the coronary tree.

1.3.2 Steady-state

Next, the simulation will be steady-state. There have been some studies indicating that steady-state should suffice when reproducing the FFR results [22, 23]. Since the FFR index is an average value over many heart cycles, it is reasonable that the simulations also manages to represent the flow as an average value. This thesis attempts to support that conclusion with more data on a large patient population.

1.3.3 Laminar flow

With the complex geometries, the nature of the flow could be approaching turbulent conditions. With regular flow conditions in a left main (LM) artery the average velocity is $U = 0.140 m/s$ and the average diameter is $D = 4.5 mm$ [24, 25]; this gives a Reynolds number of 189 at the inlet. The flow is therefore clearly laminar at the inlet. However, the regions where the flow might become turbulent is in proximity to the area with stenosis which should be looked into.

1.3.4 Newtonian fluid

Blood is considered a shear-thinning Non-Newtonian fluid [26], but this is most prominent when the flow is passing through smaller vessels. When simulating the flow through the coronary arteries, the Non-Newtonian effects are minimal and can therefore safely be neglected [6].

1.3.5 Resistance analogy

With a real heart in maximum hyperemic condition, the peripheral resistance of the coronary tree is the most important factor in the flow and pressure relation. The difficult part is to incorporate all the important factors influencing the peripheral resistance. Therefore the resistance is based on the pressure and flow at the outlets. Thereby catching as much of the peripheral effects as possible. The relation is based on Ohm's law, and can be stated as

$$R_i = \frac{P_i - P_v}{Q_i}, \tag{1.2}$$

where P_i is the outlet pressure, $P_v = 5\text{mmHg} = 666.61\text{Pa}$ is the venous pressure, Q_i is the calculated volumetric flow at the given outlet. Resulting in a resistance R_i for each outlet that can be used to simulate hyperemic conditions.

1.3.6 Tools

Several different softwares will be utilised to perform the operations outlined in this chapter. They can be seen in Table 1.1 together with the purpose they will be serving in relation to this thesis.

Software	Purpose	Where
OpenFOAM	CFD Simulations	Project
ITK-SNAP	Geometry segmentation	BM
Vaskular Modelling ToolKit	Meshing and surface handling	BM
Visualization Toolkit	Filehandling and visualization	BM
ANSYS SpaceClaim	Geometry generation & meshing	Thesis
ANSYS Fluent	CFD Simulations & meshing	Thesis
Matlab	Post-processing	Thesis
Python	Filehandling and postprocessing	Thesis

Table 1.1: Software packages used in different parts of this work (ITK-SNAP = Insight Segmentation and Registration Toolkit, BM = Research group of Biomechanics).

Chapter 2

Theory

When simulating what is happening inside human hearts, there is a limitation to what types of values that can be measured in a clinic. Therefore the amount of validation data is also quite limited. With this in mind, the theory behind the models will be presented in this chapter to ensure valid and trustworthy results in the end.

2.1 Coronary physiology

First an introduction to the coronary circulation system of the heart. The two main coronary arteries of the heart are the Right Coronary Artery (RCA) and Left Coronary Artery (LCA) often denoted also as Right Main (RM) and Left Main (LM). They comprise the vascular system of the heart and are providing blood to the muscles of the heart. An example of a heart with the most important names is shown in Figure 2.1.

Introducing the knowledge of human coronary arteries, the most notable historical developments have been outlined by Jos A. E. Spaan in collaboration with several others in the book Coronary Blood Flow: Mechanics, Distribution, and Control [27], which was released in 1991. This book outlines the developments from the 1600s until 1990 and gives a good background for the interested reader. A summary was also given in the project work [28]. Here the focus will be on the LM and RM arteries and following the branches until they get close to the microvascular circulation. Going this far will be sufficient to understand FFR and its relation to stable CAD [29] and these larger branches will be possible to reconstruct using CT; specifically Coronary Computed Tomography Angiography (CCTA).

With the reconstructed artery tree, the work on representing the physics of the flow is the next step. The different ways to proceed have been discussed thoroughly in the thesis by Bulant

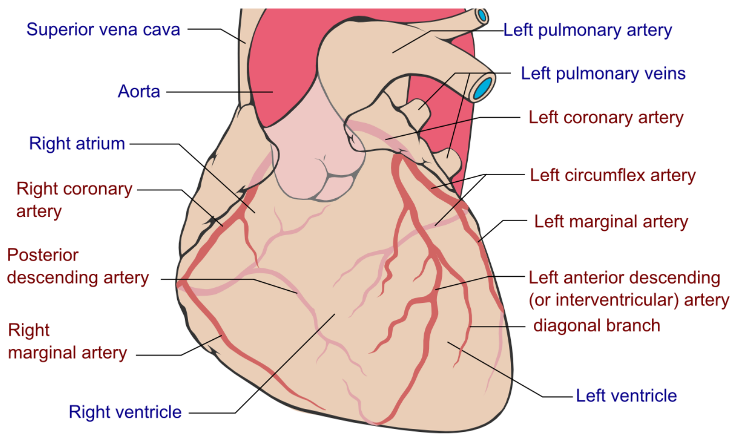


Figure 2.1: The human heart, as seen from the front [30].

[6]. The chosen method here is using Ultrasound, measuring the amount of blood the heart is pumping out, namely the Cardiac Output (CO) and then distributing this to the peripheral areas of the coronary artery tree. Only a small fraction of the CO is entering the coronary arteries. With the presence of one dominant side of the heart and normal values for flow distribution into the arteries, one can describe the amount entering the correct artery based on values calculated from a normal population [24, 25]. When representing the bifurcations in the arterial tree, the most common method is Murray's law. Proposed in 1926, the principal of minimal work relates the flow Q in an artery directly to the radius r of the vessel, where $Q \propto r^c$ and c is Murray's exponent [31, 32]. With an average value for CO, this is can be distributed to the peripheral vessels.

The BM group have tried to determine what factors effet their model the most. This was done through an Uncertainty Quantification (UQ) and Sensitivity Analysis (SA) of the setup [33]. This showed that the most important factors were the calculation of the CO, the distribution of the flow in the peripheral vessel λ_{cor} and the reduction factor α. These values were the three most influential values, with the reduction factor giving the strongest effect.

In this thesis, only one method for determining CO and λ_{cor} will be used, with only one value of α. With most of the uncertainty coming from what happens in the creation of input parameters, it is even more important to assimilate the FEM results, instead of the clinical ones. The goal of developing a 1D-0D model is that improving the patient-specific parameters will be a lot easier when the simulations complete in seconds and not minutes or hours.

With a study showing how important the factors influencing flow can be, a new method for calculating the flow distribution was tested, namely Transluminal Attenuation Gradient (TAG). This method has been proven to give even better results in FFR [34], and showed promise when implemented into this model as well. TAG is based on the gradient of contrast fluid, which is observed when propagating in the coronary tree. The concentration of contrast fluid is directly proportionate to the attenuation of the intensities of the CCTA. Therefore it is possible to measure a gradient propagating in the artery tree. This gradient can then be used to calculate how much of the flow is exciting through each outlet. The simulations and FEM results for comparison in this thesis are all based on this calculation.

Clinically measuring FFR

There are also several uncertainties in the values that are measured on patients. Clinical results will not be a large part of this thesis, but some comparison is relevant, and therefore also some theory on how certain the methods of gathering data are.

As explained in Section 1.1, there are several ways of getting the wrong values when measuring pressure drops on live patients. The clinical values gathered to support this work is all performed by skilled practitioners and with sufficiently new equipment [33]. Therefore it can be assumed that all the known ways of improving the accuracy of the measurements are already in place. Looking at the factors that cannot be changed, the uncertainty of the measuring tool [35] is within the range of ± 3 mmHg$= 400$Pa which is not critical when values are low, but with FFR in the area around the cutoff value of 0.8, this could be of clinical importance. Then the pressure wire itself, as this is an intrusive method of measuring, the presence of the wire can alter the state of the flow quite significantly. This alteration is not something that can be quantified here but should be looked into when expanding the knowledge of the model and its relation to clinical data.

When it comes to repeatability of the measurements, it has been shown that by using an algorithm for extracting the minimum value of FFR, the measured values are highly repeatable when measuring the same vessel twice during one intervention [36]. The study showed that the results were even valid when the patient did not experience a stable hyperemic state. However, the method could be automated to extract the FFR values based on pressure tracing that was done during the procedure.

2.1.1 Collateral circulation and arteriogenesis

Continuing with the understanding of the heart. The heart in itself has some effects that are difficult to model and quantify. With reduced flow in one artery, the heart can expand the network of smaller vessels to redirect the flow to the affected areas [21], this is called collateral circulation. The phenomena has also be seen in infant's hearts, meaning it is not only a result of obstructed arteries. However, the extent of the collateral vessels is much greater in the presence of a stenosis. It is an important feature of the heart to reduce fatality when experiencing an infarction. As these collateral vessels can develop over some time, and during an infarction, the collateral vessel can dilate momentarily and mitigate the reduced blood supply. If the obstructed artery is still hampering flow after the infarction, they can increase in size and become muscular arteries. If they later become a regular part of the vascular system, it is termed arteriogenesis. This thesis will not be covering vessels of that size, but it is notable when it comes to functional reduction. Moreover, it is important to know about the ability of the heart to mitigate the loss of blood in one location by redirecting it through other vessels.

2.2 ANSYS Fluent

In this thesis, ANSYS Fluent was chosen as the preferred CFD-solver. As this is commercial software, the framework is not open to the public. Nonetheless, the solver has been thoroughly validated, and as long as the setup is reasonable, it should be covered by the Theory Guide [37]. The solver is based on the Finite Volume Method (FVM) making it readily available for handling unstructured meshes of complex geometries [18]. With a simple batch-based language, going from one or two functional simulations to adapting it to a population of similar simulations will be rather simple. It is based on the programming language Scheme, which is a dialect of Lisp, and passes the arguments linearly.

There is also the option of using polyhedral meshes. A conversion method maintains the domain boundaries, but reduces the number of total cells by a factor of around 6, thereby improving the runtime of the simulation substantially. Fluent can handle both external polyhedral meshes, but also convert meshes and combine the cells to construct a polyhedral mesh.

The software provides a pressure based solver, and a density based solver. In the pressure-based solver a projection method is used to ensure continuity by solving a pressure correction equation. The governing equations are nonlinear and coupled together. Fluent provides two algorithms for solving this system. Either the segregated way based on the Semi-Implicit Method for Pressure Linked Equations (SIMPLE) algorithm [38] or a direct solver where the system is coupled. In the segregated algorithm the equations for each variable is being solved sequentially, then the pressure correction equation is solved and finally updating the fluxes, pressure and velocity. In the coupled solver, a system of momentum and pressure-based continuity is solved simultaneously, then mass flux is updated afterwards. The convergence rate of the coupled solver is higher than the segregated solver, but uses 1.5-2 times more in memory allocation.

In the density based solver, there is only a coupled option for solving the equation. There is rather the question of solving the equations implicitly or explicitly. Explicit being solved purely based on known variables, and implicit being solved with variables that are unknown and rather determined iteratively within each iteration.

2.3 3D Flow in pipes

To validate the setup, the analytical solution of laminar flow in straight pipes is a good way of starting. In the project work [28], both 2D and 3D flow in pipes were explored. Here only 3D flow will be the focus, and at first a straight pipe with the lengthwise direction along the z-axis. As this flow is assumed to be steady, Newtonian, laminar and flowing through a rigid domain, the Navier-Stokes equations (NS) in cylindrical coordinates are reduced to

$$\frac{\mu}{r}\frac{d}{dr}\left(r\frac{du_z}{dr}\right) = -\frac{dP}{dz}\frac{1}{\mu}. \tag{2.1}$$

Where u_z is the axial velocity, $\frac{dP}{dz}$ is the change in pressure along the z-axis, r the radial position, and μ is the dynamic viscosity. Integrating and simplifying this relation gives an analytical expression for the axial velocity profile [26]

$$u_z(r) = -\left(\frac{dP}{dz}\right)\frac{1}{4\mu}\left(R^2 - r^2\right), \tag{2.2}$$

where *R* is the pipe radius. This is called Hagen-Poiseulle flow and has a parabolic shape and the maximum velocity is located at the centre where $r = 0$. Integrating over the pipe area and rearranging, gives an expression for the pressure gradient along the pipe based on the total flow,

$$\frac{dP}{dz} = \frac{8\mu Q}{\pi R^4}, \tag{2.3}$$

where *Q* is the volumetric flow. This relation can be used to validate the solver and setup by looking at the change over a specific length of the pipe when the flow is fully developed. However, the entrance effects of going from a plug profile until the flow is fully developed will effect the flow. With laminar flow the entrance length is correlated with the Reynolds number as [39][1]

$$L_e \approx 0.05Re \cdot D = 0.05\frac{\rho\bar{u}D^2}{\mu}, \tag{2.4}$$

where ρ is density, $\bar{u}$ is the average velocity, μ is the dynamic viscosity and *D* the diameter.

[1]Here [39] and [26] disagree on whether it is 0.05 and 0.06. The edition of [39] referred to newer studies and was therefore chosen.

Chapter 3

Method

A summary of different simulations that will be performed is provided in Table 3.1 to give an overview of what is to be performed during this thesis.

Simulation	Purpose
3D test-case	Understand Fluent, prepare UDF
PS Baseline	Create setup, calculate resistances
PS Baseline MI	Check convergence of results
PS Baseline Res	Verify resistances
PS Hyperemic	Develop robust setup
Population PS Hyperemic	Validate and verify batch setup

Table 3.1: Overview of simulations and purpose (UDF = User Defined Function, PS = Patient Specific, MI = Mesh independence).

3.1 Solver

During the project work, an attempt was made to solve the same problem using OpenFOAM [28]. Unfortunately, this did not seem to produce satisfactory results with the geometries and setup. Therefore a commercial solver was chosen, as it is assumed to have a more complete package to approach the problem. The opportunities are limited to what NTNU software can provide. Here, ANSYS package covers all CFD areas this project would need. Within ANSYS there are two main solvers for regular fluid flow: CFX and Fluent. The original solver of the BM group is based on FEM. Choosing a FVM solver will then increase the difference between the two solvers. Therefore achieving the same results will be even more conclusive. As CFX is based on FEM, Fluent was chosen as the solver. In a similar setup [34], ANSYS Fluent has also given reasonable results in simulating coronary blood flow.

When the solvers were developed, the pressure-based was intended for lower velocities, and incompressible flows and the density-based for higher velocities and compressible flow. Lately they have been rewritten to handle all flow regimes, but to reduce the scope of this work only the pressure-based solver will be utilised.

Ensuring that solver has the most efficient and robust setup, the two different ways of coupling pressure-velocity schemes were tested. First, a direct way of solving pressure and velocity is with a coupled system, which is more memory heavy and requires more computational power for each iteration. Next, the SIMPLE was tested for increasing the simplicity of the solving but looses some robustness as it might oscillate more before a correct solution is found. The setup is similar to the one used in the previously mentioned study when using the coupled solver.

3.2 3D test-case

When working with the 3D test-case, the meshes could be created based directly on precise geometry provided from ANSYS SpaceClaim. Meshes were created in both hexahedral and tetrahedral base elements. These were then tested with different boundary conditions to check internal and boundary effects. Later the tetrahedral mesh was re-meshed using the meshing tool of Fluent. Here the surface and interior are being re-meshed using scoped sizing functions. Then by plotting the pressure drop over the length of the pipe, and the velocity profile at different positions, the boundary effects and development of flow profile could be inspected and compared to the analytical solution. The domain was initially 50mm long, but when calculating the entrance length using Equation 2.4, $L_e = 0.03 = 30mm$, which would indicate that the Hagen-Poiseuille profile would only be observable after more than half of the domain. Therefore a pipe with 100 mm in length was also tested to see if it effected the solution. Figure 3.1 shows a cross-section of the different types of meshes that were used. When trying to test the re-meshed properties the tetrahedral meshes were tested with very refined meshes to ensure that no mesh effects would affect the solution when looking at the properties of re-meshing.

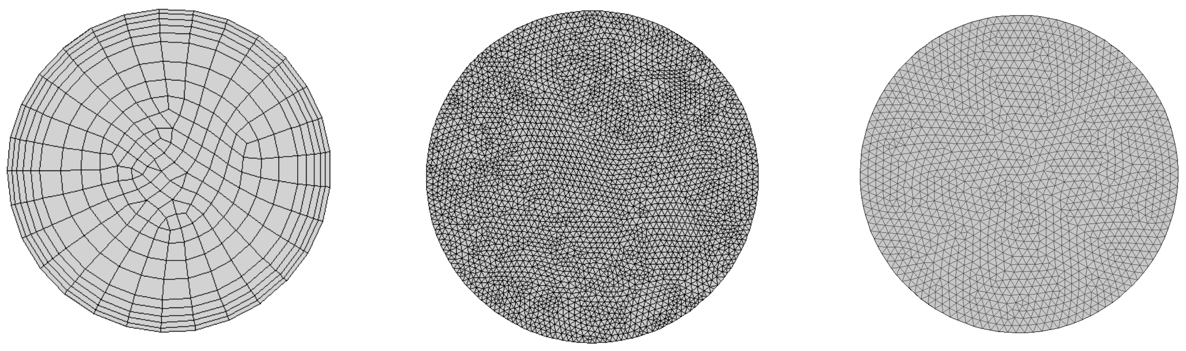


Figure 3.1: Meshes used for test-case simulation. From right to left: structured, tetrahedral, re-meshed.

3.2.1 Preparation of User Defined Functions

When working out an understanding of how to use a ”User Defined Function” (UDF), the trial and error phase is much easier to apply when the mesh is simple and completes the simulations in seconds. Therefore the initial work of UDF preparation was done with the 3D test-case. The base language for UDF writing is C, and the UDF can be either compiled or interpreted to work with the solution. Compilation requires more time to introduce but saves time when running the simulation, therefore the preferred method. Besides, when moving the simulation between Windows, Linux and Linux clusters, the safer option is to use compilation as this will be adapted to the operating system in question.

3.3 Mesh generation

When setting up the mesh independence study, the goal was to be able to read the surface geometries generated using Insight Segmentation and Registration Toolkit (ITK-SNAP) and

Vascular Modelling Toolkit (VMTK) modules, but re-mesh the surfaces and volumes with ANSYS software. The meshes were provided with tetrahedral cells and also as a more generalised surface geometry.

Working with this on different test-cases, the tools to perform mesh independence based on several factors were developed. The different meshes were tested to figure out what was possible to create using both Fluent Meshing and ICEM Meshing. However, none of the resulting meshes managed to extrapolate the bounding surfaces and re-mesh independently of the initial tetrahedral surface. It was making the new mesh only internally re-meshed, and did not increase the precision of the bounding surface. The internal mesh was possible to change, but without refinement at the boundary, the chances of improved results were small. Therefore, it was not possible to provide this method in the scope of this project, especially when the solution had to be possible to automate to handle an arbitrarily shaped coronary tree. More on the scripts and journals that was produced can be found in Appendix A.

3.3.1 Tetrahedral vs Polyhedral cells

During the setup of the simulations, the initial solution was based on using the tetrahedral meshes directly and running the simulations on these. However, as the project developed, the use of polyhedral cells gained preference in many ways. Firstly for the speed of calculation, as the number of cells is reduced by almost a factor 6. With this reduction, a faster convergence is observed as well, as the matrix sizes are reduced. After some testing, it was also observed that the robustness of the solver also increased with polyhedral cells. Therefore a comparison between tetrahedral and polyhedral cells will also be presented.

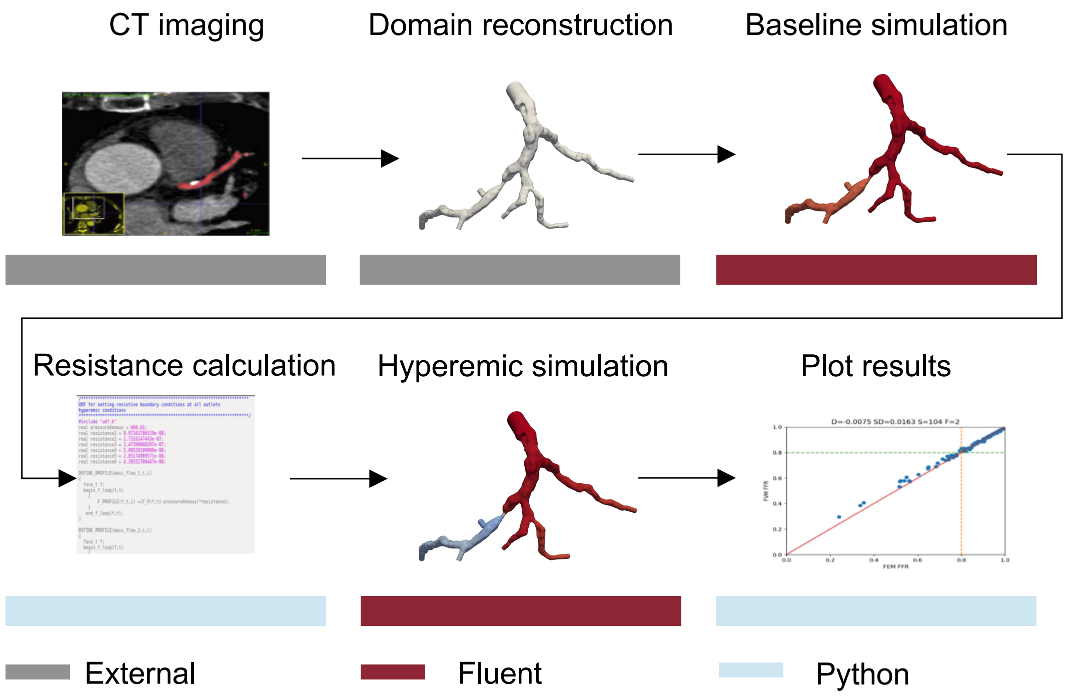
3.4 Patient specific coronary arteries

The rest of this chapter will explain the different parts of the pipeline for determining FFR values. With initial parameters prepared for the FEM solver, running the case, and presenting the final result of the solution.

With a working 3D test-case, the simulations can be expanded to include full patient-specific geometries. Starting with the Pilot 1 vessel as a benchmark for the setup. With six outlets, the simulation complexity is significantly increased. Different tests were performed with the Pilot 1 case to make sure that the setup is robust. This case was chosen as a starting point because it had most of the challenges this kind of simulation should manage, which was necessary when the different aspects of Fluent were to be tested. The complete procedure of determining FFR is given in Figure 3.2 and the parts that are performed in this project are given in Section 7.

3.4.1 Mesh independence

With no opportunity to do the mesh independence study solely based ANSYS packages, reference meshes produced with VMTK were utilised to test the converge based on mesh size. VMTK uses an edge length factor l_f as the basis for deciding cell sizes in the mesh generation. The domain is also altered to smooth the surfaces, and extend the inlet and outlets to reduce the boundary effects. Here the meshes have been extended with a length equal to two diameters of the boundary surface. During the early stages of the research on reduced-order models, a mesh independence study was performed on similar meshes that will be used in this thesis [40]. In



```
/*****************************************************************
UDF for setting resistive boundary conditions at all outlets
hyperemic conditions
*****************************************************************/
#include "udf.h"
real pressureVenous = 666.61;
real resistance1 = 8.97343706519e-08;
real resistance2 = 1.7316147443e-07;
real resistance3 = 2.473908682297e-07;
real resistance4 = 5.005201089009e-08;
real resistance5 = 2.85174989571e-08;
real resistance6 = 4.203327894447e-08;

DEFINE_PROFILE(mass_flow_1,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance1;
    }
   end_f_loop(f,t);
}

DEFINE_PROFILE(mass_flow_2,t,i)
{
  face_t f;
  begin_f_loop(f,t)
```

Figure 3.2: FFR pipeline with the software used for the action (CT image from [33]).

the previous study, necessary refinement level was set to $l_f = 0.21$. Four different levels of $l_f = [0.15, 0.18, 0.21, 0.25]$ were provided for the Pilot 1 mesh, to test the mesh independence of the solution in Fluent and was used in the same way.

3.4.2 Simulation pipeline for each coronary artery tree

1. Read discretised domain
2. Set boundary conditions
3. Baseline simulation
4. Calculate resistances
5. Reduce resistances by a factor $\alpha = 4$
6. Hyperemic simulation
7. Plot FFR results

The preferred way is to perform the entire pipeline only on self-sustaining parameters so that there is no dependency to the 3D FEM solver. Therefore the baseline simulation needs to be performed even though the values for outlet resistances are present in the configuration files for the hyperemic simulations.

The value $\alpha = 4$ is one of the critical parameters in the FEM solver setup, defining the total reduction in peripheral resistance when inducing hyperemic conditions. During the UQSA study, this was proven to be the most important parameter as it defines how the artery expansion is being controlled.

3.4.3 Reading domain and setting boundaries

Conversion from Visualization Toolkit (VTK) to Fluent mesh (MSH) is necessary to read the mesh. During this conversion, the numbering of the surfaces is going from 1-9, then starting on a-f, then continuing further with 10-N. As long as the number of outlets was lower than 10, everything worked fine. However, for N>10, the indexes had to be manually changed to follow a normal numerical order from 1-N.

When setting inlet pressures for Fluent, the only option is to set it as a total pressure

$$P_t = P_s + \frac{1}{2}\rho U^2. \tag{3.1}$$

For a controlled flow, this is a typical setup when measuring flow, but when measuring blood pressure, it is normal to get the static pressure. The total flow is known through the configuration, leaving inlet area as the only thing missing for the calculation. If the simulations are to be completely independent, with no values extracted from the FEM solver, this needs to be taken from somewhere else. Therefore an average value for all vessel was chosen based on a population average [24] giving a standard LM artery diameter of 4.5mm. This results in an average area of 1.590e-5 m^2. The other option would be to write this as a UDF setup that can be run in the beginning and calculate the total pressure based on the actual inlet area. One could also argue that getting the radius from the FEM file is not a result, but rather a preparation similar to the one that has been done to get the outlet flows. Thus the radius at the inlet can also be used to set specific inlet pressure.

3.4.4 Post-processing

To be able to sample the pressure values at the correct positions in the domain, the FEM solver is registering flow and pressure in cross-sectional areas in the domain. When working with Fluent, there is no automatic way of doing this. The closest option is using bounded planes that are created using a parallelepiped function. The function to create the surfaces takes three points and calculates a plane. With this function, bounded surfaces could be created based on the input data and using the location of two following nodes and the given radius in the point. The resulting planes with a multiplier for the radius of 3.2 can be seen in Figure 3.3. Here the junction is without surfaces, and there are also some minor discrepancies where a plane extends outside the desired artery. When checking for average pressure over all the points on the surface, the deviation is rather small. One can also see some straight edges, as the bounded function is not directly related to the outer edges of the artery. This discrepancy will cause some loss of flow near the edges, but the most important value is the pressure. Therefore, it is assumed to be sufficient. These planes are created before the simulation, enabling the opportunity to monitor the development during runtime if necessary. With all the information prepared, it is combined in the Fluent scripting language. Then it is printed out as a journal file that can do all of the setups and run the simulation; either locally or on an HPC Cluster.

3.4.5 Batch setup

By testing the solver for a larger patient population the solver can be proved functional for any patient. To connect the simulations, a database was provided with information on which domains and results matched each other. A script was made to connect the information about

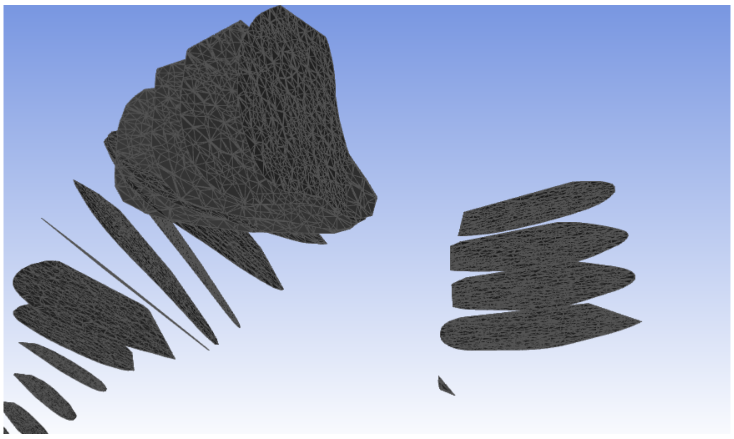


Figure 3.3: Example of surfaces created from the domain information.

which simulation was related to which mesh and additional information necessary for the file-handling. The same was done for connecting the correct baseline results with the hyperemic setup. With this database, the mass-flow-rates and static pressure, as well as surface pointers and other simulation specific information can be read from the correct configuration file.

Each simulation is not running for long, but with 78 simulations to run, it is natural to make use of HPC resources. The system of simulations is therefore prepped to be run on the IDUN cluster of NTNU [41] which is running a Slurm workload manager [42].

3.4.6 Baseline

With the journal setup of Pilot 1, the testing of different inputs was performed to ensure stability before applying it to the entire patient population. The setup was written in Python and tested for different boundary definitions, boundary conditions, physical conditions, solver parameters and initialisation types. The results in the baseline simulations were compared to the values extracted from the available FEM results.

3.4.7 Hyperemic

Resistance calculation

The calculation of resistances was explained in Section 1.3.5. The pressure and flow variables will now be based on the results of the baseline simulations. When running tetrahedral simulations, the resistances will be based on the tetrahedral baseline and the other way around for polyhedral simulations. As the surfaces used for post-processing are not completely reliable for the flow values, a native function was used to sample the values at the boundaries. This way, average values for pressure and volumetric flow can be extracted precisely and used to calculate the resistances.

UDF utilisation

The boundary condition is still going to be a set mass flow at the outlet, but with the resistance analogy, it is no longer a constant value. The resistance is the constant, but the flow used as boundary condition is based on the runtime value of the pressure

$$\dot{m} = \frac{P_i - P_v}{R_i}. \tag{3.2}$$

This method can be quite oscillating as the pressure value in each iteration is changing, as well as the flow that it is supposed to be used as a boundary condition for. In the next iteration, a new value for the flow has been set, and so on. Optionally, one can adjust the value at a given number of iterations or to add a relaxation term to only change the value a little for each iteration, thereby mitigating some of the effects. There are also several other ways to reduce this. Within the scope of this thesis, there was a rather successful solution with the function presented in Equation 3.2, and it was therefore not explored further. As the solver is working with mass flow, but the calculation of R_i is done in volumetric flow, a conversion must be made, then inverted to get a multiplication, this resistance is denoted R_i^ρ. In the initial work this was done by implementing

$$\dot{m} = P_i R_i^\rho. \tag{3.3}$$

Later it became clear that the FEM implementation also used the venous pressure when setting the boundary conditions during runtime. The final implementation was then to set

$$\dot{m} = (P_i - P_v) * R_i^\rho. \tag{3.4}$$

Part of one UDF is given below, to show how the implementation is performed for one outlet.

```
#include "udf.h"
real pressureVenous = 666.61;
real resistance1 = 8.97469967043e-08;
DEFINE_PROFILE(mass_flow_1,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance1;}
   end_f_loop(f,t);
}
```

The UDF is utilising the built in functions of Fluent to read and set values. Here ”DEFINE_PROFILE” is a general macro to set boundary conditions, ”begin_f_loop” is looping over all the faces in the thread that is the boundary and ”F_P” is reading the pressure in the given face.

This procedure is then repeated for each outlet with an individual resistance, and adjusting the boundary condition for mass flow in every iteration. The rest of the setup is identical to the setup in the baseline simulations, and the script for producing it is the same. The only difference is the writing of the C file to create boundary conditions and introducing the UDF compilation and loading in the journal. The hyperemic cases can then be performed locally or on an HPC cluster.

Some slightly simplified versions of the scripts have been added in Appendix B, to show one full setup of a simulation.

Chapter 4

Results and Discussion

4.1 3D test-case

Starting with the test-case, the comparison with analytical solutions can be seen in Figure 4.1. The theoretical entrance length is $L_e = 0.03m$, and looking at the velocity profile plot, this is aligning well at 30 mm. The profile at 50 mm is overshooting the theoretical. This overshoot is related to the calculation of the velocity profile and the difference in the discretisation of the domain. This difference in the area of the outlet is causing some discrepancies between the theoretical calculations and the simulation results. If the actual area (calculated from the mesh) is used in the calculation, the velocity profile aligns perfectly.

Looking at the plot of pressure in Figure 4.1, the inlet effects are depicted in the area between z =[0, 0.02], after this one can see the change slowly approaching the theoretical linear solution.

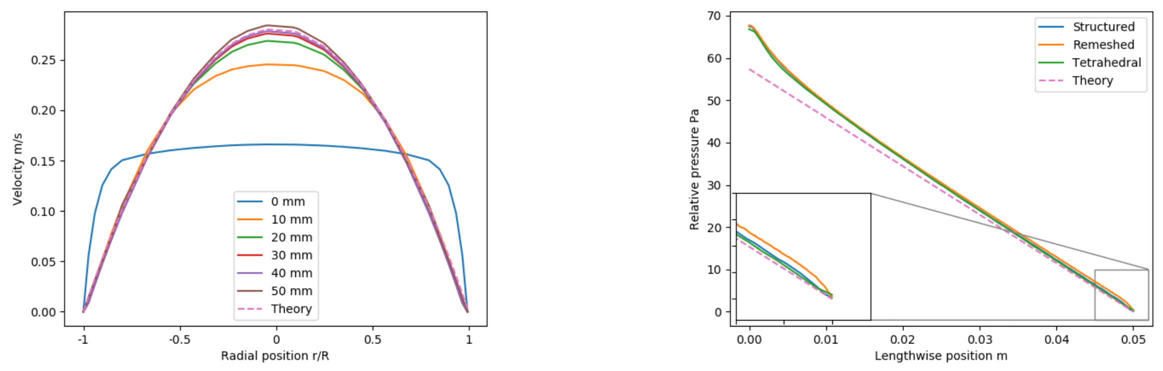


Figure 4.1: Velocity profile for the structured mesh and pressure drops for all mesh types.

Since the entrance length and the pipe length is only 20 mm apart, a 100 mm pipe was also tested. The drop in pressure continued along the straight line and is therefore not included. This feature indicates that somewhere between 30-40 mm pipe would be enough to get a fully developed profile and not influence the internal results.

Looking at the zoomed in part of the pressure plot, there is a clear difference in the outlet pressure of the re-meshed surface. Where the original meshes are maintaining the straight line, the re-meshed version is dropping at the last part of the domain. This discrepancy was another reason to why the re-meshing of tetrahedral meshes was abandoned, as there were some un-physical reactions when running with the same boundary conditions.

4.1.1 Preparation of the User Defined Function

When preparing the UDF for running with the straight, it was a simple one-function based setup where the programming for one outlet was written with the resistance from a previous simulation. Here the venous pressure was not introduced, as the resistance in a straight pipe is far less than in real vessels, and would therefore have changed the flow far more. At this point, the simulations were set up in Windows and to be able to compile functions in Windows command line tools as Visual Studio had to be installed for the compilation to be possible. One important factor for simulation setup is that when running the initialisation, the solver first needs to be initialised with native boundary conditions before adding the UDF. Without something to base the initialisation on, the solver crashes.

4.2 Patient specific geometries

There is naturally a long way from the basic straight pipe setup to an arbitrarily shaped geometry with intricate details. Still, one of the upsides of using a commercial solver is that it should be able to handle these kinds of difficult cases. Besides the FEM solver to compare the results with, it is possible to compare the FFR values to the clinical values that have been measured as explained in Section 1.1.

4.2.1 Mesh independence study

The results of the mesh independence study are summarised in Table 4.1. As the essential factor in the simulations is the difference in FFR, this is also what is used to check for mesh independence of the solution. This particular coronary tree has three different lesions that have been measured and can test both severe and less severe stenosis. Here the solutions for Pilot 1 have been provided for both the tetrahedral and polyhedral versions of the meshes. In all the different quantifications of error, the polyhedral mesh is approaching the solution in the finest mesh at a faster rate than the tetrahedral mesh. Comparing to the FEM results, the deviation is rather large at the most severe stenosis. This deviation might be possible to mitigate with a finer mesh as the solution is still moving with 0.005 and 0.009 at the last iteration as well. However, it is not probable that it will reach 0.519.

Increasing the boundary extensions could also benefit the solution, looking at the velocity profile of the outlets, the flow is not necessarily fully developed at the outlets. This could be disrupting the way the boundary conditions are interpreting the data as well as effecting the solution in itself.

When comparing the differences here with the mesh independence study performed on the FEM solver, the difference is close to one order of magnitude. Where the maximum error for all refinements are 4e-3 in their study, while here it is 3e-2. With this difference, it is clear that the Fluent solver is more mesh sensitive.

Unfortunately, only the $l_f = 0.21$ meshes were available for the population when the work in this thesis was carried out. Reiterating the same with a stronger refinement should be simple when the batch procedures have been developed.

Mesh	FFR			D	SD	Max	NCells
Tetrahedral							
l_f= 0.25	0.581	0.587	0.948	-0.0189	0.0137	0.0313	991977
l_f= 0.21	0.566	0.574	0.948	-0.0096	0.0070	0.0162	1480807
l_f= 0.18	0.565	0.571	0.948	-0.0079	0.0062	0.0149	2418239
l_f= 0.15	0.550	0.562	0.949				4819054
Polyhedral							
l_f= 0.25	0.586	0.588	0.948	-0.0148	0.0109	0.0250	180457
l_f= 0.21	0.576	0.580	0.948	-0.0088	0.0065	0.0149	262116
l_f= 0.18	0.572	0.578	0.948	-0.0067	0.0049	0.0111	418001
l_f= 0.15	0.561	0.568	0.948				812261
FEM	0.519	0.538	0.946				

Table 4.1: Difference in FFR from the four meshes (D = Mean difference, SD = Standard deviation, Max = Max difference)

4.2.2 Baseline

The results from the baseline simulations are presented in Figure 4.2. Here the result agrees very well above the regular cutoff value of 0.8, with the largest difference being 0.02. However, in the area where the pressure drop is larger over the stenosis, the difference between FVM and FEM increases[1]. The $\text{FFR}_{\text{FEM}} - \text{FFR}_{\text{FVM}}$ bias is here -0.0036, and the standard deviation is at 0.0123. Only the graphs for the polyhedral meshes have been presented here, as they are visually identical to the tetrahedral. The only difference is that the bias is reduced to -0.0035 and the standard deviation increased to 0.0124. When it comes to mitigating the error from baseline simulations, the extension of the outlets would be relevant, if longer outlets would provide closer to a fully developed flow.

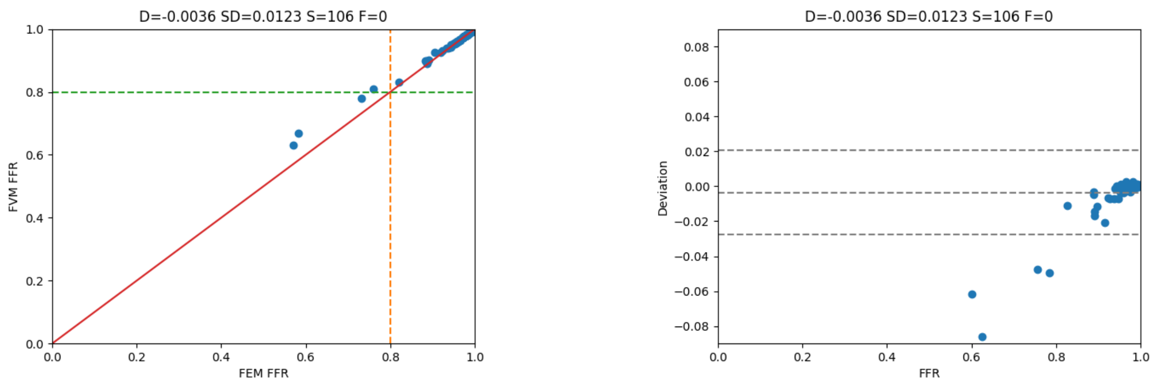


Figure 4.2: Comparing polyhedral baseline simulation with the FEM results (D=Mean difference, SD=Standard deviation, S=success, F=Failed).

[1]This is distributed over several coronary trees

Resistances

As the baseline simulations are only there to produce a value of resistance for each outlet, the FFR values are only there to check if it would be reasonable to assume that the simulations are usable. Another way to check how the simulations are performing, is to compare the resistances that are being produced from the baseline simulation. The resistances are calculated according to Equation 1.2, and in Figure 4.3 a log-log plot of the FVM and FEM resistances are showing where they come out with different results. Here there are some clear differences between the tetrahedral results (left) and the polyhedral results (right). This difference would indicate that there are at least two more resistances that have been calculated better with polyhedral results from baseline. Which shows that even though the FFR values in the baseline simulations do not align completely, the resulting resistances are correlating in all but one instance[2].

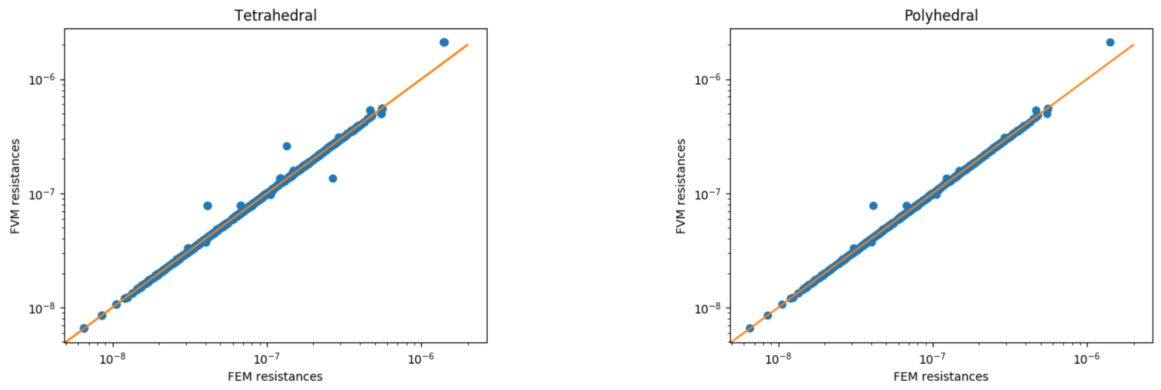


Figure 4.3: Deviations for resistances in tetrahedral(left) and polyhedral (right).

Baseline with resistance

With some resistances having larger difference it is natural to check how they perform in the initial case. After calculating the resistances they can be used to run a simulation of baseline as well. The resulting difference and standard deviation are only changes slightly in the last digit giving D = -0.0034 and SD = 0.0120. To test this similarity the results of the baseline with flow and baseline with resistance were compared to each other as well. This resulted in a mean difference of -0.0002 and a standard deviation of 0.0004. The graphs are not shown here, as the first one is visually identical to the baseline results and the last one is just a straight line. With negligible difference between the two simulations one can conclude that the resistance analogy is representing the same state as the original baseline simulation.

4.2.3 Hyperemic

With the calculated resistances from the baseline case, the hyperemic conditions were now introduced in the same meshes, and run for 2000 iterations. The simulations took on average 1157 seconds on the IDUN cluster using one node with 20 cores. However, the mean difference and the standard deviation did not change from 500 to 2000 iterations, making it possible to

[2]This instance was in patient CT_FFR_44

complete the simulations a lot faster. Where the runtime at 500 iterations averaged at 320 seconds. Some stability issues were observed with the UDF's enabled, as they are reading the pressure during runtime and adjusting the mass-flow at the outlets for each iteration. The solver is running with absolute pressures as the operating pressure is set to 0, and therefore an intermediary pressure drop that is higher than the inlet pressure will result in a negative pressure when reading values from the domain. When using pressure as a relative drop, this is not a problem, but when using the pressure value as a factor in the direct calculation, it is not as simple. When the flow is increasing, the solver response is to reduce the pressure, and when the pressure is negative, this results in an amplifying effect where the flow is increased, and the pressure is reduced until the floating point exception is invoked. In the beginning, this was a large issue causing almost half of the vessels to fail during simulations. However, with a zero-initiation of pressure and reduced relaxation of the solver gave the results in Figure 4.6. Here, 103 FFR measurements were possible to perform with tetrahedral meshes. The remaining three are from two meshes that it did not succeed to simulate with tetrahedral cells and FVM resistances.[3]

Difference in prescribed pressure

When calculating the pressure at the inlet, some population-based factors were used to set the total pressure according to Equation 3.1. The initial errors were at maximum 300 Pa. After running the simulations, a comparison was made and gave an error of 450 Pa at maximum. As FFR is a relative measurement it is dampening the effects, and for the lower values of FFR, this is not giving a relevant contribution. If this is related to a simulation for FFR with a value close to 0.8, this could be of greater importance. However, it is more likely that this is effecting the stability of the solver, as the possibility for negative pressures increase when the inlet pressure is reduced. This is more important for the borderline cases where the outlet pressure is very close to 0. For future versions, this should either be programmed as a UDF or calculated based on the FEM information.

Differences based on resistances

When calculating the resistances, there were some deviating values, but the vast majority were indistinguishable. In order to quantify this difference between resistances calculated using FVM or FEM solver, a simulation using the resistances from the FEM (here denoted Conf) solver were performed. The results of this can be seen in Figure 4.4. Where the $\mathrm{FVM}_{Conf} - \mathrm{FVM}_{Calc}$ bias is reduced to 0.0007 and the standard deviation is at 0.0037. The bias from baseline has propagated when looking at the lower levels of FFR. Which indicates that there could be a lot to gain in improving the baseline simulation also when it comes to the final results.

Difference in UDF calculation

As presented in Section 3.4.7, two different methods of implementing the boundary conditions were performed on the patient population. The method which is identical to the FEM solver is the one depicted in Equation 3.4 and this will be used in the remaining part of the results. However, as the results from Equation 3.3 are available as well, a small discussion on that will follow. Looking at Figure 4.5, the difference in FFR is amplified with lower values of

[3]This was CT_FFR_44 and CT_FFR_55

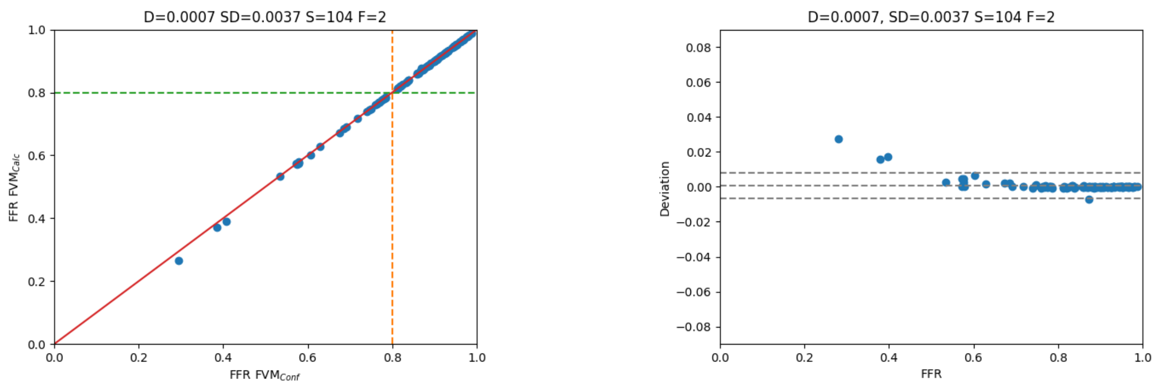


Figure 4.4: Comparing resistances calculated based FEM and FVM simulations (D=Mean difference, SD=Standard deviation, S=success, F=Failed).

FFR, where the $P_i - P_v$ is simulating a lower amount of flow in the domain. The overall bias $\text{FFR}_{P_i-P_v} - \text{FFR}_{P_i}$ is 0.0115, and the standard deviation is 0.089. This means that a lower value of flow is being imposed on the domain, and thereby reducing the drop in pressure. As the setup is not identical it is not conclusive, but can be something to focus more on in later studies. Looking at the Bland-Altman plot to the right in Figure 4.5, it is clear that difference is linearly related.

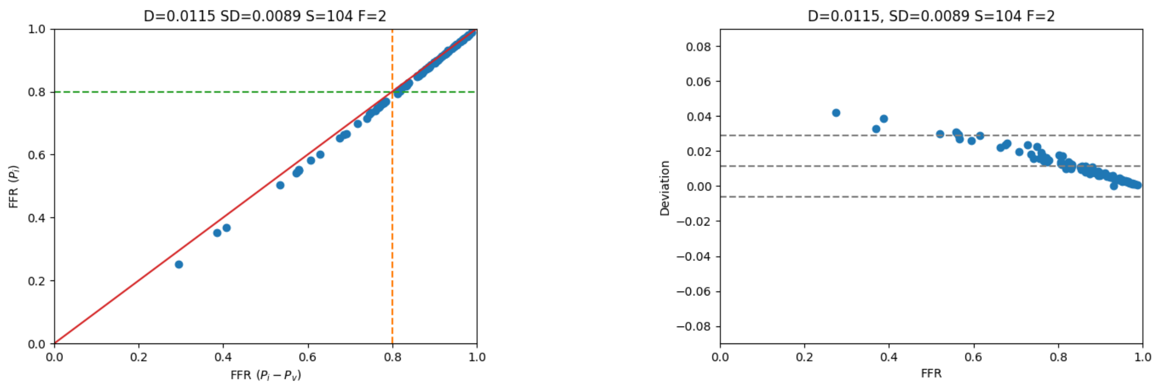


Figure 4.5: Comparing the flow set with $(P_i - P_v)R_i^\rho$ and $P_i R_i^\rho$.

Tetrahedral cells

The rest of the results will be using the method which is similar to the FEM solver. Using the tetrahedral mesh provided a clear bias is seen in Figure 4.6. The FVM solver is giving a lower pressure drop, and therefore, higher values of FFR when going lower than 0.8. This gives a $\text{FFR}_{\text{FEM}} - \text{FFR}_{\text{FVM}}$ bias of -0.0079 and a standard deviation of 0.0152. Looking at the Bland-Altman plot to the right, the linear relationship is still clear, but with a somewhat larger spread between the two solvers. However, it is clear that over the cutoff value of 0.8, which is normally used for diagnostic purposes, the variations are located quite close to zero.

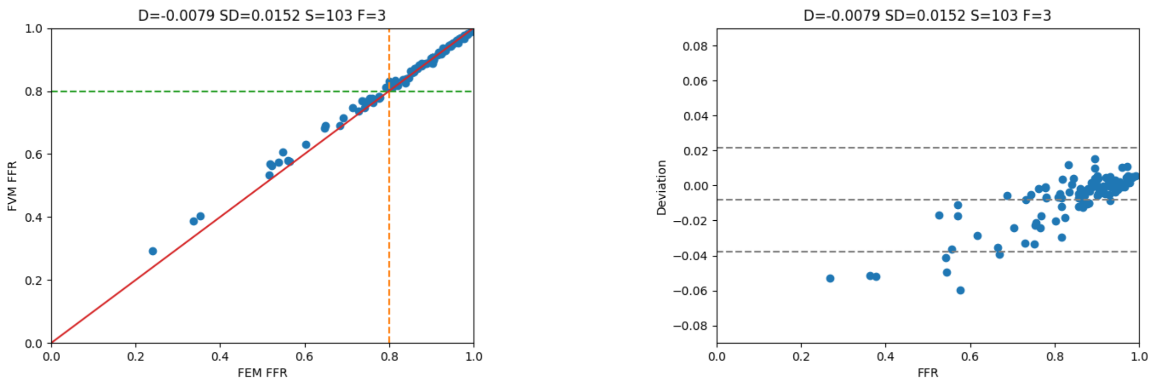


Figure 4.6: Comparing tetrahedral hyperemic simulations with FEM results (D=Mean difference, SD=Standard deviation, S=success, F=Failed).

Polyhedral cells

To increase the number of vessels that gave successful results, the polyhedral function of Fluent was used to convert the domain to polyhedral cells. Figure 4.7 shows the results with calculated resistances and a polyhedral mesh. Here one more mesh succeeded, but the final mesh was still not possible to complete[4]. Resulting in 104 FFR values. The $FFR_{FEM} - FFR_{FVM}$ bias is slightly lower than tetrahedral with -0.0075, but the standard deviation have increased to 0.0163. With 77 completed meshes and a bias which is very close to negligible the solver can be said to perform well. However, some work is needed on reducing the error when the FFR values are lower than 0.8. With a clear linear relationship this is a systemic error. This can be related to one of the mesh factors discussed earlier, or the way of prescribing the boundary conditions for flow in Fluent. To test the boundary conditions, it would be relevant to explore the opposite way of prescribing outlet conditions. Reading flow across the surface, and setting the pressure, could help the issues of negative pressures, as well as problems with underdeveloped flow.

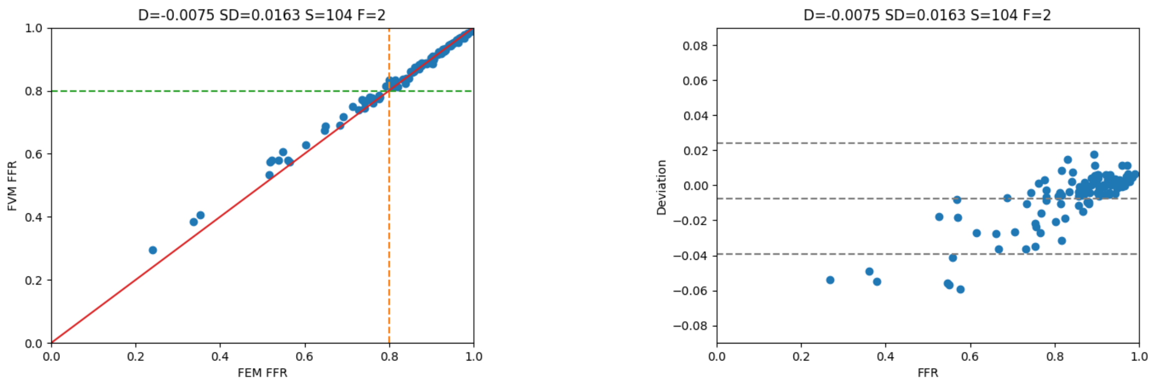


Figure 4.7: Comparing polyhedral hyperemic simulations with FEM results (D=Mean difference, SD=Standard deviation, S=success, F=Failed).

[4]This was CT_FFR_55

Turbulence

Another issue that was brought up in Section 1.3 is the presence of turbulence. When setting up the simulations it was assumed that the flow would be laminar based on the inlet flow. Looking at the results from the FEM solver, the Reynolds number is quite high, with a maximum Re ranging from 4357 to 10368 in the population. The border between laminar and turbulent is starting at about Re=2300, indicating that there is at least an intermittent turbulent area in the domain. With the complex shape of the domain, there might be several areas where the flow is turbulent, giving some unwanted effects during the simulation. This is something that should be explored further with different turbulence models to ensure that the assumption of laminar flow is still valid.

4.2.4 Diagnostic relevance

Comparing the results to the clinical values, the spread is much more visible. The $FFR_{Clinical} - FFR_{FVM}$ bias is -0.0198, and the standard deviation is 0.1154 which means that there will be many in the range 0.7-0.9 that can be misdiagnosed. As can be seen in Figure 4.8, there are many positions were the clinical values, and the calculated values disagree whether the stenosis is significant or not. Take extra notice of the scale in the Bland-Altman plot. The results differed with close to one order of magnitude compared to the FEM-FVM comparison and was therefore not possible to present both results with the same axes. The diagnostic accuracy with prediction sensitivity, prediction specificity, positive predictive value and negative predicted values they were 70, 92, 77 and 0.88%, respectively.

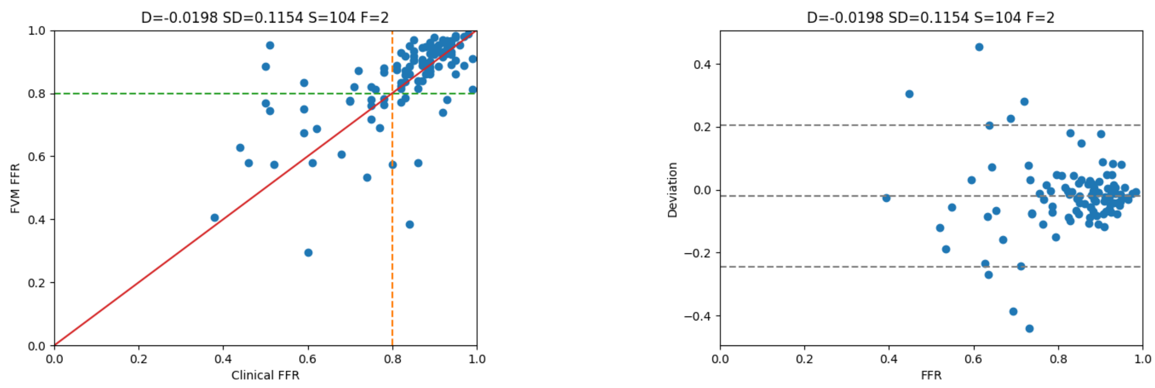


Figure 4.8: Comparing polyhedral hyperemic simulations with clinical results. Notice the scale is different on the Bland-Altman plot (D=Mean difference, SD=Standard deviation, S=success, F=Failed).

Clinical measurements on the FFR values have been gathered using pressure wire measurements, and are the basis of this comparison. The procedure and possibilities for failure have been presented in Section 1, and concludes on many problems related to the procedure. However, it does not discuss the validity of the actual measurements. With an intrusive measurement procedure, the introduction of a pressure wire into the artery may in itself produce deviating results. The wires in use have a diameter of 0.38mm, which is smaller than the most severe stenosis, but not that much. To exemplify, the diameter in the strongest stenosis in Pilot 1 is 1.07mm, which makes the pressure wire obstructing 12.6% of the area in the stenosis. This obstruction could change the results quite a lot. The effects of this can be tested by introducing pressure wires in the simulations in the future.

Chapter 5

Conclusion

A 3D test-case has been produced to develop knowledge of ANSYS and the programming tools of Python needed to automate the procedure. The simulations were successfully validated with the analytical solution for Hagen-Poiseuille flow. There were some minor discrepancies in the magnitude of the velocity profile, but the pressure drop approached the linear relation found in theory. The knowledge of batch language and large scale simulations in Fluent was developed.

A case for determining Computational Fractional Flow Reserve in human coronary arteries have been developed. A functional model to compare the results from FVM and FEM solvers has been performed in 77 out of 78 coronary trees available. With the chosen solver setup, the FEM-FVM bias was -0.0075 and a standard deviation of 0.0163. The model could produce accurate FFR results with an average simulation time of down to 320 seconds per case, running with one node and 20 cores of an HPC cluster.

The work did not succeed in producing individual meshes and had to utilise previously created meshes to be able to simulate the domain. The case showed some more sensitivity to mesh refinement, but possibly also the length of the extended areas at each boundary.

The case can be used as a basis for future work in the research on reduced order models and the improvements of diagnostic tools for stable CAD.

Chapter 6

Further work

6.0.1 Complete the model

The current model is useful in the intermediary state, but to be able to include the results in research, the final vessel should also be possible to simulate. The difficulty of negative pressures and instabilities when the pressure drop is approaching the level of pressure at the inlet needs to be addressed. Some simulation managed to bounce back, but the CT_FFR_55 coronary tree did not succeed in any of the simulations. This case can be used as a benchmark in further studies to finalise the model.

6.0.2 Reverse the method

There is now a difference in methodology between the FEM method and the FVM method. Where the FVM is reading pressure values and setting the flow values at the boundary. To increase the similarity between the methods, efforts should be made at producing a case that is reading flow values and returning a pressure value instead.

6.0.3 Turbulence

Checking the Reynolds numbers in the simulations shows that there can be intermediate turbulent regions. This is something that need to be tested for different turbulence models to ensure that the assumption of laminar flow is still valid.

6.0.4 Mesh improvement

The mesh independence study showed that there is some potential to reduce the error in FFR by refining the meshes further. This should be tested to see how the mesh refinement can effect the results together with increasing the extensions of of the outlets.

Bibliography

[1] World Health Organization. The top 10 causes of death. `http://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death`, accessed October 2018.

[2] Olivier Muller, Fabio Mangiacapra, Argyrios Ntalianis, Katia M.C. Verhamme, Catalina Trana, Michalis Hamilos, Jozef Bartunek, Marc Vanderheyden, Eric Wyffels, Guy R. Heyndrickx, Frank J.A. van Rooij, Jacqueline C.M. Witteman, Albert Hofman, William Wijns, Emanuele Barbato, and Bernard De Bruyne. Long-term follow-up after fractional flow reserve–guided treatment strategy in patients with an isolated proximal left anterior descending coronary artery stenosis. JACC: Cardiovascular Interventions, 4(11):1175 – 1182, 2011.

[3] Pijls et. al. NHJ. Experimental basis of determining maximum coronary, myocardial, and collateral blood flow by pressure measurements for assessing functional stenosis severity before and after percutaneous transluminal coronary angioplasty. Circulation, 1993.

[4] Bora Toklu MD Judah Rauch MD Jeffrey D. Lorin MD Iryna Lobach PhD Steven P. Sedlis MD Louis H. Miller, MD. Very long-term clinical follow-up after fractional flow reserve-guided coronary revascularization, 2012.

[5] Manesh R Patel, John H Calhoon, Gregory J Dehmer, James Aaron Grantham, Thomas M Maddox, David J Maron, and Peter K Smith. Acc/aats/aha/ase/asnc/scai/scct/sts 2017 appropriate use criteria for coronary revascularization in patients with stable ischemic heart disease: A report of the american college of cardiology appropriate use criteria task force, american association for thoracic surgery, american heart association, american society of echocardiography, american society of nuclear cardiology, society for cardiovascular angiography and interventions, society of cardiovascular computed tomography, and society of thoracic surgeons. Journal of the American College of Cardiology, 69(17), 2017.

[6] Carlos Alberto Bulant. Computational models for the geometric and functional assessment of the coronary circulation. PhD thesis, LNCC - National Laboratory for Scientific Computing, 2017.

[7] Dr. S. Venkatesan MD. `https://drsvenkatesan.com/tag/drawbacks-of-ffr-in-pci/`, accessed May 2019.

[8] Mark A. Hlatky, Akshay Saxena, Bon-Kwon Koo, Andrejs Erglis, Christopher K. Zarins, and James K. Min. Projected costs and consequences of computed tomography-determined fractional flow reserve. Clinical Cardiology, 36(12):743–748, 2013.

[9] Pim A.L. Tonino, Bernard De Bruyne, Nico H.J. Pijls, Uwe Siebert, Fumiaki Ikeno, Marcel vant Veer, Volker Klauss, Ganesh Manoharan, Thomas Engstrøm, Keith G. Oldroyd, Peter N. Ver Lee, Philip A. MacCarthy, and William F. Fearon. Fractional flow reserve versus angiography for guiding percutaneous coronary intervention. The New England Journal of Medicine, 360(3):213–224, 2009.

[10] Nico H.J. Pijls, William F. Fearon, Pim A.L. Tonino, Uwe Siebert, Fumiaki Ikeno, Bernhard Bornschein, van&Amp;Apos, Marcel T Veer, Volker Klauss, Ganesh Manoharan, Thomas Engstrøm, Keith G. Oldroyd, Peter N. Ver Lee, Philip A. Maccarthy, and Bernard De Bruyne. Fractional flow reserve versus angiography for guiding percutaneous coronary intervention in patients with multivessel coronary artery disease: 2-year follow-up of the fame (fractional flow reserve versus angiography for multivessel evaluation) study: 2-year follow-up of the fame (fractional flow reserve versus angiography for multivessel evaluation) study. Journal of the American College of Cardiology, 56(3):177–184, 2010.

[11] Bernard De Bruyne, William F. Fearon, Nico H.J. Pijls, Emanuele Barbato, Pim Tonino, Zsolt Piroth, Nikola Jagic, Sven Mobius-Winckler, Gilles Rioufol, Nils Witt, Petr Kala, Philip MacCarthy, Thomas Engström, Keith Oldroyd, Kreton Mavromatis, Ganesh Manoharan, Peter Verlee, Ole Frobert, Nick Curzen, Jane B. Johnson, Andreas Limacher, Eveline Nüesch, and Peter Jüni. Fractional flow reserve–guided pci for stable coronary artery disease. The New England Journal of Medicine, 371(13):1208–1217, 2014.

[12] William Fearon, F., Takeshi Nishi, B., Bernard De Bruyne, H.J., Derek Boothroyd, A., Emanuele Barbato, A., Pim Tonino, A., Peter Jüni, A., Nico Pijls, A., and Mark Hlatky, A. Clinical outcomes and cost-effectiveness of fractional flow reserve–guided percutaneous coronary intervention in patients with stable coronary artery disease: Three-year follow-up of the fame 2 trial (fractional flow reserve versus angiography for multivessel evaluation). Circulation, 137(5):480–487, 2018.

[13] Gilles Montalescot, Udo Sechtem, Stephan Achenbach, Felicita Andreotti, Chris Arden, Andrzej Budaj, Raffaele Bugiardini, Filippo Crea, Thomas Cuisset, Carlo Di Mario, J Rafael Ferreira, Bernard J Gersh, Anselm K Gitt, Jean-Sebastien Hulot, Nikolaus Marx, Lionel H Opie, Matthias Pfisterer, Eva Prescott, Frank Ruschitzka, and Manel Sabaté. 2013 esc guidelines on the management of stable coronary artery disease: the task force on the management of stable coronary artery disease of the european society of cardiology. European heart journal : the journal of the European Society of Cardiology, 34(38):2949–3003, October 2013.

[14] Nico H. J. Pijls, Morton J. Kern, Paul G. Yock, and Bernard De Bruyne. Practice and potential pitfalls of coronary pressure measurement. Catheterization and Cardiovascular Interventions, 49(1):1–16, 2000.

[15] B. Hannawi, W.W. Lam, S. Wang, and G.A. Younis. Current use of fractional flow reserve: A nationwide survey. Texas Heart Institute Journal, 41(6):579–584, 2014.

[16] Paul D. Morris, Frans N. van de Vosse, Patricia V. Lawford, D. Rodney Hose, and Julian P. Gunn. ”virtual” (computed) fractional flow reserve: Current challenges and limitations: Current challenges and limitations. JACC: Cardiovascular Interventions, 8(8):1009–1017, 2015.

[17] Strategic research area NTNU Health. Model based, noninvasive diagnosis of coronary artery disease with 3d ultrasound and ct. `https://www.ntnu.edu/health`, accessed December 2018.

[18] Joel Ferziger and Milovan Peric. Computational methods for fluid dynamics. Berlin: Springer-Verlag, 1996.

[19] Laura Dempere-Marco, Estanislao Oubel, Marcelo Castro, Christopher Putman, Alejandro Frangi, and Juan Cebral. Cfd analysis incorporating the influence of wall motion: Application to intracranial aneurysms. In Rasmus Larsen, Mads Nielsen, and Jon Sporring, editors, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2006, pages 438–445, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[20] Guus A de Waard, Christopher M Cook, Niels van Royen, and Justin E Davies. Coronary autoregulation and assessment of stenosis severity without pharmacological vasodilation. European Heart Journal, 39(46):4062–4071, 12 2017.

[21] Gim-Hooi Choo. Collateral circulation in chronic total occlusions – an interventional perspective. Current Cardiology Reviews, 11(4):277–284, 2015.

[22] C.A. Bulant, P.J. Blanco, G.D. Maso Talou, C. Guedes Bezerra, P.A. Lemos, and R.A. Feijóo. A head-to-head comparison between ct- and ivus-derived coronary blood flow models. Journal of Biomechanics, 51:65 – 76, 2017.

[23] Paul D. Morris, Daniel Alejandro Silva Soto, Jeroen F.A. Feher, Dan Rafiroiu, Angela Lungu, Susheel Varma, Patricia V. Lawford, D. Rodney Hose, and Julian P. Gunn. Fast virtual fractional flow reserve based upon steady-state computational fluid dynamics analysis: Results from the virtu-fast study. JACC: Basic to Translational Science, 2:434 – 446, 2017.

[24] J.T. Dodge Jr., B.G. Brown, E.L. Bolson, and H.T. Dodge. Lumen diameter of normal human coronary arteries: Influence of age, sex, anatomic variation, and left ventricular hypertrophy or dilation. Circulation, 86(1):232–246, 1992.

[25] Shingo Sakamoto, Saeko Takahashi, Ahmet U. Coskun, Michail I. Papafaklis, Akihiko Takahashi, Shigeru Saito, Peter H. Stone, and Charles L. Feldman. Relation of distribution of coronary blood flow volume to coronary artery dominance. The American Journal of Cardiology, 111(10):1420 – 1424, 2013.

[26] Frank M White. Fluid mechanics, 2009.

[27] Jos A. E Spaan. Coronary Blood Flow : Mechanics, Distribution, and Control, volume 124 of Developments in Cardiovascular Medicine. Dordrecht, 1991.

[28] Magnus Johannesen. Steady-state solver for blood flow simulations in patient-specific coronary arteries. Technical report, Norwegian University of Science and Technology, 2019.

[29] Dirk J Duncker, Akos Koller, Daphne Merkus, and John M Canty. Regulation of coronary blood flow in health and ischemic heart disease. Progress in cardiovascular diseases, 57(5):409–422, 2015.

[30] Heart anatomy introduction, website. `http://upload.wikimedia.org/wikipedia/commons/thumb/1/18/Coronary_arteries.svg/758px-Coronary_arteries.svg.png`, accessed October 2018.

[31] Cecil D. Murray. The physiological principle of minimum work. i. the vascular system and the cost of blood volume. Proceedings of the National Academy of Sciences of the United States of America, 12(3):207–214, 1926.

[32] Cecil D. Murray. The physiological principle of minimum work applied to the angle of branching of arteries. The Journal of General Physiology, 9(6):835–841, 1926.

[33] Fredrik E. Fossan, Jacob Sturdy, Lucas O. Müller, Andreas Strand, Anders T. Bråten, Arve Jørgensen, Rune Wiseth, and Leif R. Hellevik. Uncertainty quantification and sensitivity analysis for computational ffr estimation in stable coronary artery disease. Cardiovascular Engineering and Technology, Oct 2018.

[34] S. Kishi, A.A. Giannopoulos, A. Tang, N. Kato, Y.S. Chatzizisis, C. Dennie, Y. Horiuchi, K. Tanabe, J.A.C. Lima, F.J. Rybicki, and D. Mitsouras. Fractional flow reserve estimated at coronary ct angiography in intermediate lesions: Comparison of diagnostic accuracy of different methods to determine coronary flow distribution. Radiology, 287(1):76–84, 2018.

[35] Radi Medical Systems AB. `https://fccid.io/U4L01080410/User-Manual/Users-manual-909001`, accessed May 2019.

[36] Nils P. Johnson, Daniel T. Johnson, Richard L. Kirkeeide, Colin Berry, Bernard De Bruyne, William F. Fearon, Keith G. Oldroyd, Nico H.J. Pijls, and K. Lance Gould. Repeatability of fractional flow reserve despite variations in systemic and coronary hemodynamics. JACC: Cardiovascular Interventions, 8(8):1018–1027, 2015.

[37] ANSYS Inc. `https://www.sharcnet.ca/Software/Ansys/18.2.2/en-us/help/ai_sinfo/flu_intro.html`, accessed January 2019.

[38] S. V. Patankar and D. B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. International Journal of Heat and Mass Transfer, 1972.

[39] Yunus A Çengel. Fluid mechanics : fundamentals and applications. McGraw-Hill, Boston, 3rd ed. in si units. edition, 2014.

[40] Etienne Boileau, Sanjay Pant, Carl Roobottom, Igor Sazonov, Jingjing Deng, Xianghua Xie, and Perumal Nithiarasu. Estimating the accuracy of a reduced-order model for the calculation of fractional flow reserve (ffr. International Journal for Numerical Methods in Biomedical Engineering, 34(1):n/a–n/a, 2018.

[41] HPC Group NTNU. `https://www.hpc.ntnu.no/display/hpc/Hardware`, accessed May 2019.

[42] SchedMD. `https://slurm.schedmd.com/documentation.html`, accessed May 2019.

Appendix A

Mesh generation

During the work in this thesis one of the main things that was preferable to explore was the opportunity to create meshes using the ANSYS package. Mainly because it is natural that the meshing tools and solver setup have been optimised to work together, and therefore a better pipeline towards the solution could be achieved. Also to be able to perform more tests on mesh sensitivity and interaction, it would be preferable to perform this using only ANSYS software. This was tested with various setups and configurations, but ultimately ended up without any usable results. The work in setting up and preparing is however something that will be delivered from this thesis, and will be summed up in this Appendix.

A.1 Meshing pipeline

The overall goal is to create an automated tool that can read mesh generated using VMTK and prepare a new mesh with a set of meshing parameters that will be able to produce usable meshes.

First the mesh needs to be translated from Vascular Modelling Toolkit (VTK). From VTK the geometry is based represented by a triangular surface mesh with information regarding inlet, outlets and wall of the domain. It is then converted to a Fluent meshg (MSH) to be interpreted by the solver. There are a number of meshing tools in the ANSYS package. Here follows a rundown of strengths and weaknesses of these packages.

In the ANSYS package one can utilise the following tools for meshing: Workbench Meshing, TGrid, TurboGrid and ICEM CFD. The Workbench Meshing tool is only accepting CAD geometry, and was therefore not relevant for the current test. The TurboGrid mesher is optimalised for spinning geometries and mostly used alongside CFX, which is also optimised for turbo-machinery, and hence was not prioritised. Then TGrid mesher is in later version of ANSYS been incorporated as Fluent Meshing tool, and was therefore assumed to be the best choice ,as Fluent had already been chosen as the preferred solver. It is made to handle complex geometries which is essential when handling CT Imagery. TGrid also had the opportunity to read and re-mesh the surfaces with all the metadata intact from a test on the 3D testcase. As this is integrated in the Fluent module the programming language is also following the same syntax, which is preferable when adapting the simulation to batch based simulations.

Then followed the extensive testing of the software to obtain suitable meshes for the 3D test-case and coronary artery tree domains. The process was developed to be as automated as possible. The mesh was loaded as boundary mesh, skipping all the internal faces. Then assigned as both geometry and mesh to attempt different starting points. The patches were read in with allocated names from mesh file. As the input is only a triangulated surface and not a geometry representation, it was necessary to re-mesh all the patches at the same time. To have a reasonable account on how refined the domain should be fluent is utilising size functions min/max sizes and growth rate for the cell elements. With a bounding box from [0 0 0] to [0.004 0.1 0.004] the element sizes were in the range of [1e-6 , 1e-4].

After a long time trying to perfect this method to be able to automate the setup and execute the mesh generation in a batch-wise manner it was evident that there was no effective way to interpolate the initial geometry. It was not possible to re-mesh the boundary surfaces in a way that would be able to generate a finer mesh on the surfaces, which means that the highest point of refinement would always be the initial mesh.

With no successful method in TGrid efforts were made to utilise ICEM CFD instead. This module also supports importing of MSH files as a geometrical entity and regain the boundaries as labelled surfaces for re-meshing. The ICEM CFD had more geometry recognition abilities than TGrid, but did not support journal writing. As ICEM CFD is based on Tcl/Tk while Fluent is based on Lisp there was another setup language that needed to be understood in order to develop batch jobs. With the fact that there was limited time left of the thesis period, the attempt to produce independent meshes and perform mesh independence testing with ANSYS meshes was abandoned.

However a script based setup where the scoped functions and simulation generation was produced in Python to handle batch simulation for three different mesh parameters. This can easily be expanded to include the other parameters, or changed to handle different ways of running the simulation. The files created for this purpose can be found here:
`https://www.dropbox.com/s/ssx8sek3g5uhlaj/meshGeneration.zip?dl=0`

Appendix B

Simulation files

Here the files for a full regular setup will be presented. The journal file, C file and slurm file are all generated from the baselineFileGeneration.py. Also, the rest of the Python scripts used during the setup, simulation and post-processing for this thesis are included here:
`https://www.dropbox.com/s/of5gmod6u8jovij/simulationFiles.zip?dl=0`

B.1 File generation

Listing B.1: Python script for generating simulation files

.

```python
#Python script for generating mesh journal straight pipe
import os
import math
import re
import io
import numpy as np
import sys
import copyfilesFromffr_simulationDB as copyScript # import prepareFFRCases_list_ffr_simulationDB
import writeSolutionDataFromCTL as vtkToCSVScript

def readConfFile(filename, **options):
    #Procedure to read the configuration file and return the relevant simulation values
    with open(filename,'r')as file:
        listOfValues = []
        for line in file:
            if  "mu=" in line:
                mu = round(float(line.strip().split('=')[1])/10,4)
            if "rho=" in line:
                rho = float(line.strip().split('=')[1])*1000
            if "wall=" in line:
                wall = int(line.strip().split('=')[1])
            if "inlet=" in line:
                inlet = int(line.strip().split('=')[1])
            if "num_outlets=" in line:
                nOutlets = int(line.strip().split('=')[1])
            if "outletAverageTarget=" in line:
                flows = line.strip().split('=')[1].split('*')
                for d in range (len(flows)): flows[d] = float(flows[d])*1e-6*rho
            if "p_initial=" in line:
                #The pressure is read as static pressure, and from the calculated flows adding the dynamic pressure.
                inletPressure= float(line.strip().split('=')[1])/10+0.5*rho*(sum(flows)/(1.590431281e-5*rho))**2 #This number is from Pilot 1. If not set, need to use UDF to set inletTotalPressure
                # print(0.5*rho*(sum(flows)/(0.00001661902514*rho))**2)
            if "meshfile" in line:
                meshFile = line.strip().split()[2]
            if options.get('resistance'):
                #If it is preferred to run with resistances from the configuration file
                if "outletsResistance=" in line:
                    resistances = line.strip().split('=')[1].split(',')
                    for d in range (len(resistances)): resistances[d]= rho*1e-5/float(resistances[d])
            else:
                resistances = []
        listOfValues = [mu, rho, wall, inlet, nOutlets, inletPressure, flows, meshFile, resistances]
    if nOutlets > 10:
        #With more then 10 outlets the fluent solver will fail because the conversion makes 1-9, then a-f, then 10-N
        #This method should be developed further to change the numeration of the mesh file if it finds an error. Now it just checks whether it can be run or not.
        with open('../../'+re.sub(r'\/CT.*f','',filename.replace('../../',''))+'/'+meshFile.replace('.xml.gz', '.msh').replace('../../',''), 'rb') as f:
            f.seek(-2, os.SEEK_END)
            while f.read(1) != b'\n':
                f.seek(-2, os.SEEK_CUR)
            lastline = f.readline().strip('()').replace('(','').split()[1]
        if float(lastline)<=13:
            print('../../'+re.sub(r'\/CT.*f','',filename.replace('../../',''))+'/'+meshFile.replace('.xml.gz', '.msh').replace('../../',''))
            sys.exit('TOO MANY OUTLETS WILL FAIL ON READ')
        # print(listOfValues)
    return listOfValues

def makeBoundaryNames(wallID, inletID,nOutlets):
    #Making names to be used in the journal
    boundaryNames = []
    outletMarker = 1;
    for i in range (0,nOutlets+3):
        if i == wallID:
            boundaryNames.append("walls")
        elif i == inletID:
            boundaryNames.append("inlet")
        elif i == nOutlets+2:
            boundaryNames.append("interior")
        else:
            boundaryNames.append("outlet"+str(outletMarker))
            outletMarker+=1;
    return boundaryNames

def readCSVFile(filename, **options):
    #Reading the CSV file with nodes and result to create bounded planes or surfaces.
    #Printing out commands for journal file to create the preferred type.
    #The bounded planes are made from /surface/plane-bou plane{nodeID} x1 y1 z1 x2 y2 z2 x3 y3 z3 "samplePoints yes/no"
    #The sphere is made from  /surface/sphere-slice sphere{nodeID} x0 y0 z0 radius
    file = open(filename, "r")
    nodenames = []
    # n =[]
    # p =[]
    # f =[]
    # nodeprint = [0, 127, 256, 268, 322, 329, 356]
    index = 0;
    radiusMultiplier = 3.2
    if options.get('frequency') != None:
        frequency = options.get('frequency')
    else:
        frequency = 500

    journaloutput = """
"""
    firstline = file.readline()
    lineID = -1
    xyz = np.arange(9).reshape(3,3).astype(np.float)
    theta = math.radians(70);
    for line in file:
        nodenames= np.array(line.strip().split(',')).astype(np.float)
        #If just spheres are wanted everything can be made from:
        # spherelist = """ {0} {1} {2} {3}""".format(nodenames[2],nodenames[3],nodenames[4],nodenames[5])
        # journaloutput+="""/surface/sphere-slice sphere{nodeID} {list}""".format(nodeID=int(planeID), list = spherelist)
        # And the skip straight to report definitions, where you have to change to sphere* and not plane*

        # if nodenames[1] in nodeprint:
        #     n.append(nodenames[1])
        #     f.append(nodenames[6])
        #     p.append(nodenames[7])
        # print(nodenames)

        # Calculating three positions that can be used to generate bounded planes.
        # Checking whether one direction is negative when doing linalg operations.
        if lineID == int(nodenames[0]):
            oldpoint =newpoint
            newpoint = nodenames[2:5]
            direction=np.array(newpoint-oldpoint)/np.linalg.norm(np.array(newpoint-oldpoint))
            if direction[2]==0:
                if direction [1]==0:
                    if direction[0]==0:
                        print("all directions =0")
                    else:
                        avec = np.array([-(direction[2]*1 + direction[1]*1)/direction[0],1,1])
                else:
                    avec = np.array([1,-(direction[0]*1 + direction[2]*1)/direction[1],1])
            else:
                avec = np.array([1,1,-(direction[0]*1 + direction[1]*1)/direction[2]])
            avec = avec/np.linalg.norm(avec)
            bvec = np.cross(avec, direction)
            planeID = nodenames[1]-1
            radius = nodenames[5]*radiusMultiplier
        elif lineID ==-1:
            newpoint = np.array(file.next().strip().split(',')[2:5]).astype(np.float)
            radius = nodenames[5]*radiusMultiplier
            oldpoint = nodenames[2:5]
            direction=np.array(newpoint-oldpoint)/np.linalg.norm(np.array(newpoint-oldpoint))
            if direction[2]==0:
                if direction [1]==0:
                    if direction[0]==0:
                        print("all directions =0")
                    else:
                        avec = np.array([-(direction[2]*1 + direction[1]*1)/direction[0],1,1])
                else:
                    avec = np.array([1,-(direction[0]*1 + direction[2]*1)/direction[1],1])
            else:
                avec = np.array([1,1,-(direction[0]*1 + direction[1]*1)/direction[2]])
            avec = avec/np.linalg.norm(avec)
            bvec = np.cross(avec, direction)
            planeID = nodenames[1]
            lineID = nodenames[0]
        elif lineID != int(nodenames[0]):
            direction = -direction;
            oldpoint = newpoint
            if direction[2]==0:
                if direction [1]==0:
                    if direction[0]==0:
                        print("all directions =0")
                    else:
                        avec = np.array([-(direction[2]*1 + direction[1]*1)/direction[0],1,1])
                else:
                    avec = np.array([1,-(direction[0]*1 + direction[2]*1)/direction[1],1])
            else:
                avec = np.array([1,1,-(direction[0]*1 + direction[1]*1)/direction[2]])
            avec = avec/np.linalg.norm(avec)
            bvec = np.cross(avec, direction)
            planeID +=1
            lineID = nodenames[0]
            newpoint=nodenames[2:5]
        else:
            print("something wierd happened")

        journaloutput+="""/surface/plane-bou plane{nodeID} """.format(nodeID=int(planeID))
        for i in range (3):
            for j in range(3):
                xyz[i][j] = oldpoint[j] + radius*avec[j]*np.cos(theta*(i+1)) + radius*bvec[j]*np.sin(theta*(i+1))
                journaloutput+=""" {coord}""".format(coord=xyz[i][j])
        journaloutput+=""" no
"""


    journaloutput+="""/solve/report-definitions/add pressurePlanes surface-facetavg field pressure surface-names plane* () per-surface  yes /
/solve/report-definitions/add flowPlanes surface-volumeflowrate surface-names plane* () per-surface  yes /
/solve/report-files/add pressurePlanes-rfile file-name "pressurePlanes.out" frequency {freq} report-defs pressurePlanes () print? no /
/solve/report-files/add flowPlanes-rfile file-name "flowPlanes.out" frequency {freq} report-defs flowPlanes () print? no /
""".format(freq = frequency)
    # l = [n,p,f]
    # for i in range (3):
    #     for j in range (len(p)):
    #         print(l[i][j])
    return journaloutput

def generateJournalFile(floats, flows, boundaryNames,nodedata, hyperemicChanges,meshFile, simtype,**options):
    # Gerenating the journalfile with all paramters in the right place.
    if simtype <1:
        conv = 0;
        iterations = 5000;
        convergenceLevel = simtype
    elif (type(simtype) is int):
        iterations = simtype;
        conv = 3;
        convergenceLevel = "1e-06"
    journal = """/file/set-tui-version "19.1"
/file/read-case {meshName}
/mesh/scale 0.01 0.01 0.01
/define/materials/change-create air blood yes constant {rhoValue} no no yes constant {muValue} no no no yes
/define/operating-conditions/operating-pressure 0
""".format(rhoValue=floats[1], muValue=floats[0], meshName=meshFile)
    # The mesh for CT_FFR_40 was wierd and had to be converted through openfoam. Therefore some extra had to be done.
    if meshFile == '../../CT_FFR_40_Mesh/CT_FFR_40_Mesh_0000/CT_FFR_40_Mesh_0000_RCA_vol.msh':
        for k in range (0,floats[4]+2):
            journal+="""/define/boundary-conditions/zone-name surface{surfID} {newName}
""".format(surfID=k+3, newName=boundaryNames[k])
        journal+="""/define/boundary-conditions/zone-name int* interior
/define/boundary-conditions/zone-name fl* blood
```

```
"""
    else:
        for k in range (0,floats[4]+3):
            journal+="""/define/boundary-conditions/zone-name {surfID} {newName}
""".format(surfID=k+3, newName=boundaryNames[k])
    journal+="""/define/boundary-conditions/modify-zones/zone-type inlet pressure-inlet
"""
    for k in range(floats[4]):
        journal+="""/define/boundary-conditions/modify-zones/zone-type outlet{num} mass-flow-outlet
""".format(num=k+1);
    journal+="""
/solve/set/p-v-coupling 24
/solve/set/p-v-control 100 0.2 0.2"""
    #P-v coupling is 24 coupled, 20 SIMPLE, 21 SIMPLEC
    if options.get('polySim'):
        journal+="""
/mesh/poly/convert-domain yes"""
    journal +="""
/define/boundary-conditions/pressure-inlet inlet yes no {pres} no 0. no yes
""".format(pres = floats[5])
    for k in range(floats[4]):
        journal+="""/define/boundary-conditions/mass-flow-outlet outlet{num} yes yes no {flow} no yes
""".format(num=k+1, flow= flows[k]);
    journal+="""
/solve/report-definitions/add volumeflow surface-volumeflowrate surface-names inlet """
    for k in range(floats[4]):
        journal+="""outlet{num} """.format(num=k+1);
    journal+=""", average-over 1 per-surface yes
/add pressurerep surface-facetavg surface-names inlet """
    for k in range(floats[4]):
        journal+="""outlet{num} """.format(num=k+1);
    journal+=""" () field pressure per-surface yes
/add velocitymax volume-max zone-names blood () field velocity-magnitude /

/solve/report-files/add volumeflow-rfile file-name "volumeFlows.out" frequency 1 report-defs volumeflow () print? yes /
/solve/report-files/add pressurerep-rfile file-name "surfacepressureFile.out" frequency 1  report-defs pressurerep () print? yes /"""
    journal+=nodedata
    # Adding the surface definitions and sample positions
# /solve/report-files/add ffrValues-rfile file-name "ffrValues.out" frequency 1 report-defs ffrvalues () frequency 1 print? no /
    journal +="""/solve/report-files/add velocitymax-rfile file-name "velocitymax.out" frequency 1 report-defs velocitymax () frequency 1 print? yes /
/solve/monitor/res/crit-typ 3
/solve/initialize/set-hyb-initialization gen-se 10 1 1 relative no no no
/solve/initialize/hyb-initialization
"""
    #Adding the UDF compilation
    journal+=hyperemicChanges

    # /solve/init/hyb-init yes
    journal+="""

/solve/iter 10
/solve/monitor/residual/crit-typ {convergence}
/solve/monitors/residual/convergence-criteria {convergenceLevel} {
    convergenceLevel} {convergenceLevel} {convergenceLevel}

/solve/iterate {iter}

/report/system/time-sta""".format(convergence = conv, iter = iterations,
    convergenceLevel=convergenceLevel)
    # journal +=""" /file/write-ca-da  simResults""" # Can be added if you
    want to save the simulation data in addition to the pressure/flow values
    journal+= """

/exit yes
"""

    return (journal)

def slurmSimulationGeneration(workingDirs, arrayLength, simtype,output):
    # Generating the batchfile for running on cluster with slurm queue
    slurm="""#!/bin/bash
#SBATCH --partition=WORKQ
#SBATCH --time=20:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=20
#SBATCH --array=0-{length}%5
#SBATCH --mem=25G

module load FLUENT/19.2

A=({listOfDirs})

cd $""".format(length = arrayLength, listOfDirs=workingDirs)
    # A will hold al the folders that will be simulated in.
    slurm+="""{A[${SLURM_ARRAY_TASK_ID}]}
rm -r libudf
rm *.out log
b=($(ls -d */))
echo "${b}"
"""
    slurm+="""
c={outputFolder}
fluent 3ddp -i {simulation}.jou""".format(simulation=simtype, outputFolder=
    output)
    slurm+=""" -pinfiniband -t${SLURM_NTASKS} -g >stdout.out 2>error.out
    """
    #In regular baseline the output should be "" in poly it should be poly/
    if simtype == 'baseline':
        slurm+="""

mkdir -p ${b}fluentResults/${c}
mv *.out *.cas *.dat *.xy *.sh ${b}fluentResults/${c}
```

```
"""
    else:
        slurm+="""
mkdir -p ${b}fluentResults/${c}
mv *.out log libudf *.cas *.dat *.xy *.sh ${b}fluentResults/${c}
cp *.c ${b}fluentResults/${c}

"""
    return slurm

def getCaseIndexes(haystack, needle):
    # To check that all cases also have a directory to work in this method
    is checking folders against database
    if not needle:
        return
    # just optimization
    lengthneedle = len(needle)    # print(needle[0])
    list = []
    for i in range(len(needle)):
        firstneedle = needle[i]
        for idx, item in enumerate(haystack):
            # print (haystack[1])
            if item['patientName'] == firstneedle:
                # print("haystack")
                # print(haystack[idx:idx+lengthneedle][1])
                # if haystack[idx]['patientName'] == needle:
                list.append(idx);
            # print(item['patientName'])
                # print(tuple(range(idx,idx+lengthneedle)))
    return list

def writeSlurm(slurmText, filename):
    # Was prepared to write slurm files on windows computer with UNIX
    endings
    with io.open (filename, 'w', newline = '\n') as file:
        file.write(slurmText);

def writeFile(journalText, filename):
    print("Writing file {0}".format(filename))
    with open (filename, 'w') as file:
        file.write(journalText);

def readFFRFiles(filename):
    with open(filename, 'r') as f:
        ffrValues = f.read().splitlines()
        for i in range (len(ffrValues)): ffrValues[i] = ffrValues[i].split(
    ' ')
    return ffrValues

def readResults(resultFolder, **options):
    # Reading the flow and pressure outlets
    # When used to postprocess, also reading the flowPlanes and
    pressurePlanes files
    with open(resultFolder+'/volumeFlows.out', 'rb') as f:
        f.seek(-2, os.SEEK_END)
        while f.read(1) != b'\n':
            f.seek(-2, os.SEEK_CUR)
        flows =f.readline().decode("utf-8").strip().split()[1:];
    # print (iteration)
    for d in range(0,len(flows)): flows[d] = abs(float(flows[d]))
    # print (massflowstrings)
    # iteration = int(iteration[0])
    with open(resultFolder+'/surfacepressureFile.out', 'rb') as f:
        f.seek(-2, os.SEEK_END)
        while f.read(1) != b'\n':
            f.seek(-2, os.SEEK_CUR)
        pressures =f.readline().decode("utf-8").strip('\n').split()[1:];
        # print(pressures)
    for d in range(0,len(pressures)): pressures[d] = float(pressures[d])
    # Addition for postProcessing
    if options.get("postProcess")!= None:
        resultFiles = options.get('postProcess')
        results = []
        for i in range(len(resultFiles)):
            floatValues = []
            with open(resultFolder+'/'+resultFiles[i], 'r') as f:
                name = resultFiles[i].replace('Planes.out','Values')
                linefile = f.read().splitlines()
                linefile[-1] =linefile[-1].split(' ')
                # print(linefile[2].replace(resultFiles[i].lower().strip('.
out'),'').replace('\"(plane', 'ID').replace(')\"','').split())
                for d in range(len(linefile[-1])): floatValues.append(abs(
float(linefile[-1][d])))
                results.append({'filename' : resultFiles[i],
                                'planeIDs' : linefile[2].replace(
resultFiles[i].lower().strip('.out'),'').replace('\"(plane', 'ID').
replace(')\"','').split()[1:],
                                name : floatValues[1:]})

        if os.path.isfile(resultFolder+'/velocitymax.out')== True:
            with open(resultFolder+'/velocitymax.out', 'rb') as f:
                f.seek(-2, os.SEEK_END)
                while f.read(1) != b'\n':
                    f.seek(-2, os.SEEK_CUR)
                maxvelocity =float(f.readline().decode("utf-8").strip().
split()[1])
        if os.path.isfile(resultFolder+'/stdout.out')== True:
            with open(resultFolder+'/stdout.out', "r") as file:
                lines = file.read().splitlines()
            linecount = 0
            for i in range(len(lines)):
                if "Total wall-clock" in lines[i]:
                    time=lines[i].strip().split()[3]
                    break

                if "/report/system/time-sta" in lines[i]:
                    finalResiduals = lines[i-9].split()[1:5]
                    # print(finalResiduals)

        pressures = [flows, pressures,maxvelocity, time, finalResiduals] #
Sending it as extravalues to the postprocessing
```

```
        flows = results  #Sending the results from the sampled planes to postprocessing
    return [flows, pressures]

def resistanceCalculation (flows, pressures, density):
    # Calculating the four different types of resistances that was tested
    # pVAdjustedHyp is the one that is using the venous pressure and dividing it by four
    pressureVenous = 666.61
    regular = []
    pVAdjusted = []
    regularHyp = []
    pVAdjustedHyp = []
    for i in range(len(flows)):
        regular.append(density/(pressures[i]/abs(flows[i])))
        pVAdjusted.append(density/((pressures[i]-pressureVenous)/abs(flows[i])))
        regularHyp.append(regular[i]*4)
        pVAdjustedHyp.append(pVAdjusted[i]*4)
    return [regular, pVAdjusted,regularHyp, pVAdjustedHyp]

def udfGeneration (nOutlets, resistances):
    # Generating the UDF file that is controlling the simulations during runtime
    UDFfile = """/***********************************************************************
UDF for setting resistive boundary conditions at all outlets
hyperemic conditions
************************************************************************/
#include "udf.h"
real pressureVenous = 666.61;
"""
    for i in range(nOutlets):
        UDFfile+="""real resistance{numOut} = {resistance};
""".format(numOut=i+1, resistance=resistances[i])
# Fix for hindering negative pressure values part 1
#         UDFfile+="""
# real presval{numOut};
# """.format(numOut=i+1)
    for i in range (nOutlets):
        UDFfile +="""
DEFINE_PROFILE(mass_flow_{numOut},t,i)""".format(numOut=i+1)
        UDFfile +="""
{
  face_t f;
  begin_f_loop(f,t)
    { """
# Fix for hindering negative pressure values part 2
#         UDFfile+="""
#         presval{numOut} = F_P(f,t);
#          if (presval{numOut} < 0) """.format(numOut=i+1)
#         UDFfile+="""
#         {"""
#         UDFfile+="""
#             presval{numOut} = 100; """.format(numOut=i+1)
#         UDFfile+= """
#          }
        # UDFfile+="""
        # F_PROFILE(f,t,i) =presval{numOut}*resistance{numOut}; """.format(numOut=i+1)
#  """
        UDFfile+="""
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance{numOut}; """.format(numOut=i+1)
        UDFfile +="""
    }
   end_f_loop(f,t);"""
        UDFfile +="""
}
"""
    return UDFfile

def generateHyperemicPart(nOutlets, filename):
    # Adding the part to introduce the UDF to the calculation
    journal ="""
/define/user-defined/compiled-functions compile "libudf" yes "{name}" "" ""
/define/user-defined/compiled-functions load "libudf"
""".format(name = filename)
    for k in range(nOutlets):
        journal+="""/define/boundary-conditions/mass-flow-outlet outlet{num} yes yes yes yes "udf" "mass_flow_{num}::libudf" no yes
""".format(num=k+1);
    return journal

def simulationPrep(patient,arrayLength, **options):
    # Full method for preparing the simulations
    # Most of the changes can be made in the main part, but choosing which resistance is done manually in this method


    # folderPath = '../database/'+patient['patientName']+'/'
    folderPath = '../../'+patient['patientName']+'/' # added by Fredrik
    simulationPath = folderPath+patient['patientName']+'_Simulation/'+patient['simuName']+'/'
    #simulationPath = folderPath+patient['patientName']+ '/'+patient['patientName']+ '_Simulation/'+patient['simuName']+'/' # added by Fredrik
    CSVPath =simulationPath+patient['simuName']+'_out/ctlResults/'
    meshPath = patient['patientName']+'_Mesh/'+patient['patientName']+'_Mesh_'+patient['meshNumber']+'/'
    #meshPath = folderPath+patient['patientName']+ '/' + patient['patientName']+'_Mesh/'+patient['patientName']+'_Mesh_'+patient['meshNumber']+'/' # added by Fredrik
    [mu, rho, wall, inlet, nOutlets, inletPressure, flows, meshFile, res] = readConfFile(simulationPath+patient['simuName']+'.conf', resistance=options.get('getconf'))
    if options.get('onlyResistance'):
        resultPath = folderPath+patient['patientName']+'_Simulation/'+options.get("baselinePath")+'/'+options.get("baselinePath")+'_out/fluentResults/'
        [flow_results, pressure_results] = readResults(resultPath)
        [regular, pVAdjusted,regularHyp, pVAdjustedHyp] = resistanceCalculation(flow_results[1:],pressure_results[1:], rho)
        journal = [pVAdjustedHyp, res]
        for i in range(len(pVAdjustedHyp)):
            diff = (pVAdjustedHyp[i]-res[i])/res[i]
            if diff>=0.01:
                print(diff)
                print(simulationPath)
    else:
        if "baseline" in  options.get("simtype"):
            hyperemicExtra = ""
        elif options.get("simtype")=="hyperemic" or options.get('simtype')=='hyperemicConf':
            resultPath = folderPath+patient['patientName']+'_Simulation/'+options.get("baselinePath")+'/'+options.get("baselinePath")+'_out/fluentResults/'
            if options.get('simtype')=='hyperemicConf':
                udfString = udfGeneration(nOutlets,res)
                [flows, pressure_results] = readResults(resultPath)

            else:
                if options.get('polySim'):
                    resultPath+='poly/'
                [flow_results, pressure_results] = readResults(resultPath)
                [regular, pVAdjusted,regularHyp, pVAdjustedHyp] = resistanceCalculation(flow_results[1:], pressure_results[1:], rho)
                udfString = udfGeneration(nOutlets,pVAdjustedHyp)
                if len(flows) != len(flow_results)-1:
                    print('Not equal lengths')
                    print(len(flows))
                    print(len(flow_results))
                else:
                    for i in range(len(flows)):
                        flows[i] = flow_results[i+1]*rho
            hyperemicExtra = generateHyperemicPart(nOutlets, options.get("udfName"))
            writeFile(udfString, simulationPath+options.get("udfName"))
    #             print (patient)
        if os.path.isfile(CSVPath+'ctlSol_Average.csv'):
            nodedata =readCSVFile(CSVPath+'ctlSol_Average.csv', frequency=options.get('iterations'))
        else:
            vtkToCSVScript.variableDefinitionAndWrite(CSVPath+'ctlSol_Average.vtk',CSVPath+'ctlSol_Average.csv')
            nodedata =readCSVFile(CSVPath+'ctlSol_Average.csv')
        if os.path.isfile(folderPath+meshPath+patient['meshNameVTK'].replace('.vtk','.msh')):
            meshFile = ('../../'+meshPath+patient['meshNameVTK'].replace('.vtk','.msh')).strip()
        else:
            print("did not find meshfile {0}".format(patient['simuName']))
            commandstring = """vmtk vmtkmeshwriter -f fluent -mode ascii -ifile {VTKFile} -entityidsarray CellEntityIds -ofile {MSHFile}""".format(VTKFile=(meshPath+patient['meshNameVTK']), MSHFile=meshPath+patient['meshNameVTK'].replace('.vtk','.msh') )
            # os.system(commandstring)
            meshFile = meshPath+patient['meshNameVTK'].replace('.vtk','.msh')
        boundaries = makeBoundaryNames(wall, inlet, nOutlets)
        journal = generateJournalFile([mu, rho, wall, inlet, nOutlets, inletPressure], flows, boundaries, nodedata, hyperemicExtra,meshFile, options.get('iterations'), polySim = options.get('polySim'))
    return [journal, str(simulationPath)]

if __name__=='__main__':
    listofcases = copyScript.passVar('1D_3D_TAG_BLN.xlsx')
    listofHypCases = copyScript.passVar('1D_3D_TAG_HYP.xlsx')
    # Generating all the cases
    slurmfolder=''
    #target = '../database/'
    target = '../../' # added by Fredrik
    directoryList = os.listdir(target)
    iter = 0
    indexes = getCaseIndexes(listofcases,directoryList)
    # Finding the indexes. This could preferable be a sorted list
    # print indexes, directoryList
    list = []
    resList = []
    simulationType = 'hyperemic'
    # Simulationtype can be baseline, hyperemic or hyperemicConf. Choosing three different simulation setups
    udfName = 'resistanceBaseline.c'
    outputDirectory = 'poly/resistanceBaseline/'
    # outputDirectory being sent to the slurmfile ensuring that the simulation is ending up in the right place
    polySim = True
    #  polysim to choose where the results are calculated from and or produced with
    onlyResistance = False
    # Small if to check the resistances that will be used
    simLength = 2000
    # print(indexes)
    # print(listofHypCases[12])

    for i in range(len(indexes)): #(len(indexes)):
        if onlyResistance == False:
            if simulationType=='baseline' or simulationType=='baselinePoly' :
                [journal, folderPath] = simulationPrep(listofcases[indexes[i]],len(indexes),simtype = simulationType, iterations = simLength, polySim = polySim)
                slurmfolder += folderPath+ ' '
                writeFile(journal, folderPath+simulationType+'.jou')
                list.append(indexes[i])
                iter+=1
            elif simulationType=='hyperemic' or simulationType =='hyperemicConf':
                if simulationType == 'hyperemicConf':
                    getconf= True
                else:
                    getconf = False
                print('Connecting ', listofcases[indexes[i]]['simuName'], ' with ', listofHypCases[indexes[i]]['simuName'])
                if os.path.isfile(target+listofcases[indexes[i]]['patientName']+'/'+listofcases[indexes[i]]['patientName']+'_Simulation/'+listofcases[indexes[i]]['simuName']+'/'+listofcases[indexes[i]]['simuName']+'_out/fluentResults/pressurePlanes.out') or simulationType == 'hyperemicConf':
                    [journal, folderPath] = simulationPrep(listofHypCases[indexes[i]],len(indexes),simtype = simulationType,baselinePath=listofcases[indexes[i]]['simuName'], udfName =udfName, iterations = simLength, getconf= getconf, polySim = polySim )
                    # print(listofcases[indexes[i]])
                    list.append(indexes[i])
                    # Slurmfolder is the list of folders that will be added to the slurm file and iterated over in the arraysim
                    slurmfolder+=folderPath+' '
                    writeFile(journal, folderPath+simulationType+'.jou')
                    iter+=1
                else:
                    print(target+listofcases[indexes[i]]['patientName']+'/'+listofcases[indexes[i]]['patientName']+'_Simulation/'+listofcases[indexes[i]]['simuName'], 'Failed')
        else:
            # Checking for resistances
            [journal, folderPath] = simulationPrep(listofHypCases[indexes[i]],len(indexes),simtype = simulationType,baselinePath=listofcases[indexes[i]]['simuName'], udfName =udfName, iterations = simLength, getconf= True, onlyResistance = onlyResistance)
            resList.append(journal)
    slurm = slurmSimulationGeneration(slurmfolder,len(list)-1,simulationType,outputDirectory)
    print (slurm)
    # print(resList)
    writeFile(slurm,simulationType+'Queue.slurm' )
    print("Done")
```

B.2 Simulation journal

Listing B.2: Ansys Fluent journal for simulation setup

```
.
/file/set-tui-version "19.1"
/file/read-case ../../CT_FFR_Pilot_1_Mesh/CT_FFR_Pilot_1_Mesh_0001/
    CT_FFR_Pilot_1_Mesh_0001_LM_vol.msh
/mesh/scale 0.01 0.01 0.01
/define/materials/change-create air blood yes constant 1050.0 no no yes
    constant 0.0035 no no no yes
/define/operating-conditions/operating-pressure 0
/define/boundary-conditions/zone-name 3 walls
/define/boundary-conditions/zone-name 4 outlet1
/define/boundary-conditions/zone-name 5 outlet2
/define/boundary-conditions/zone-name 6 inlet
/define/boundary-conditions/zone-name 7 outlet3
/define/boundary-conditions/zone-name 8 outlet4
/define/boundary-conditions/zone-name 9 outlet5
/define/boundary-conditions/zone-name 10 outlet6
/define/boundary-conditions/zone-name 11 interior
/define/boundary-conditions/modify-zones/zone-type inlet pressure-inlet
/define/boundary-conditions/modify-zones/zone-type outlet1 mass-flow-outlet
/define/boundary-conditions/modify-zones/zone-type outlet2 mass-flow-outlet
/define/boundary-conditions/modify-zones/zone-type outlet3 mass-flow-outlet
/define/boundary-conditions/modify-zones/zone-type outlet4 mass-flow-outlet
/define/boundary-conditions/modify-zones/zone-type outlet5 mass-flow-outlet
/define/boundary-conditions/modify-zones/zone-type outlet6 mass-flow-outlet

/solve/set/p-v-coupling 24
/solve/set/p-v-control 100 0.2 0.2
/define/boundary-conditions/pressure-inlet inlet yes no 12829.1904942 no 0.
     no yes
/define/boundary-conditions/mass-flow-outlet outlet1 yes yes no
    0.000267997387085 no yes
/define/boundary-conditions/mass-flow-outlet outlet2 yes yes no
    0.000464699635851 no yes
/define/boundary-conditions/mass-flow-outlet outlet3 yes yes no
    0.000649859235945 no yes
/define/boundary-conditions/mass-flow-outlet outlet4 yes yes no
    0.000149463407609 no yes
/define/boundary-conditions/mass-flow-outlet outlet5 yes yes no
    7.72099875164e-05 no yes
/define/boundary-conditions/mass-flow-outlet outlet6 yes yes no
    0.000126197855286 no yes

/solve/report-definitions/add volumeflow surface-volumeflowrate
    surface-names inlet outlet1 outlet2 outlet3 outlet4 outlet5 outlet6 ,
    average-over 1 per-surface yes
/add pressurerep surface-facetavg surface-names inlet outlet1 outlet2
    outlet3 outlet4 outlet5 outlet6  () field pressure per-surface yes
/add velocitymax volume-max zone-names blood () field velocity-magnitude /

/solve/report-files/add volumeflow-rfile file-name "volumeFlows.out"
    frequency 1 report-defs volumeflow () print? yes /
```

```
/solve/report-files/add pressurerep-rfile file-name "surfacepressureFile.
    out" frequency 1  report-defs pressurerep () print? yes /


/surface/plane-bou plane0  0.00363295891004 0.173736718459 -0.183161921588
    -0.00335325075306 0.170493443533 -0.190023216298 -0.0104009805418
    0.177852272346 -0.191629599233 no
/surface/plane-bou plane1  0.00388886579701 0.173959536228 -0.183484448195
    -0.00348165074655 0.170351888041 -0.189734542853 -0.0108631545713
    0.177453667335 -0.190951693195 no
#Repeats for N number of planes

/solve/report-definitions/add pressurePlanes surface-facetavg field
    pressure surface-names plane* () per-surface  yes /
/solve/report-definitions/add flowPlanes surface-volumeflowrate
    surface-names plane* () per-surface  yes /
/solve/report-files/add pressurePlanes-rfile file-name "pressurePlanes.out"
     frequency 2000 report-defs pressurePlanes () print? no /
/solve/report-files/add flowPlanes-rfile file-name "flowPlanes.out"
    frequency 2000 report-defs flowPlanes () print? no /
/solve/report-files/add velocitymax-rfile file-name "velocitymax.out"
    frequency 1 report-defs velocitymax () frequency 1 print? yes /
/solve/monitor/res/crit-typ 3
/solve/initialize/set-hyb-initialization gen-se 10 1 1 relative no no no
/solve/initialize/hyb-initialization

/define/user-defined/compiled-functions compile "libudf" yes "
    resistancePVExtraUDF.c" "" ""
/define/user-defined/compiled-functions load "libudf"
/define/boundary-conditions/mass-flow-outlet outlet1 yes yes yes yes "udf"
    "mass_flow_1::libudf" no yes
/define/boundary-conditions/mass-flow-outlet outlet2 yes yes yes yes "udf"
    "mass_flow_2::libudf" no yes
/define/boundary-conditions/mass-flow-outlet outlet3 yes yes yes yes "udf"
    "mass_flow_3::libudf" no yes
/define/boundary-conditions/mass-flow-outlet outlet4 yes yes yes yes "udf"
    "mass_flow_4::libudf" no yes
/define/boundary-conditions/mass-flow-outlet outlet5 yes yes yes yes "udf"
    "mass_flow_5::libudf" no yes
/define/boundary-conditions/mass-flow-outlet outlet6 yes yes yes yes "udf"
    "mass_flow_6::libudf" no yes


/solve/iter 10
/solve/monitor/residual/crit-typ 3
/solve/monitors/residual/convergence-criteria 1e-06 1e-06 1e-06 1e-06

/solve/iterate 2000

/report/system/time-sta

/exit yes
```

B.3 User Defined Function

Listing B.3: User Defined Function

.

```
/**********************************************************************
UDF for setting resistive boundary conditions at all outlets
hyperemic conditions
**********************************************************************/
#include "udf.h"
real pressureVenous = 666.61;
real resistance1 = 8.97469967043e-08;
real resistance2 = 1.73564631367e-07;
real resistance3 = 2.47415051215e-07;
real resistance4 = 5.00588852099e-08;
real resistance5 = 2.85452564402e-08;
real resistance6 = 4.20362641325e-08;

DEFINE_PROFILE(mass_flow_1,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance1;
    }
   end_f_loop(f,t);
}

DEFINE_PROFILE(mass_flow_2,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance2;
    }
   end_f_loop(f,t);
}

DEFINE_PROFILE(mass_flow_3,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance3;
    }
   end_f_loop(f,t);
}

DEFINE_PROFILE(mass_flow_4,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance4;
    }
   end_f_loop(f,t);
}

DEFINE_PROFILE(mass_flow_5,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance5;
    }
   end_f_loop(f,t);
}

DEFINE_PROFILE(mass_flow_6,t,i)
{
  face_t f;
  begin_f_loop(f,t)
    {
        F_PROFILE(f,t,i) =(F_P(f,t)-pressureVenous)*resistance6;
    }
   end_f_loop(f,t);
}
```

B.4 Slurm queue file

Listing B.4: Slurm file for running on cluster

.
```
#!/bin/bash
#SBATCH --partition=WORKQ
#SBATCH --time=20:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=20
#SBATCH --array=0-78%5
#SBATCH --mem=25G

module load FLUENT/19.2

A=(../../CT_FFR_55/CT_FFR_55_Simulation/CT_FFR_55_Simulation_0010/ ../../
    CT_FFR_Pilot_1/CT_FFR_Pilot_1_Simulation/CT_FFR_Pilot_1_Simulation_0014/
     ../../CT_FFR_31/CT_FFR_31_Simulation/CT_FFR_31_Simulation_0010/ (and so
     on))

cd ${A[${SLURM_ARRAY_TASK_ID}]}
rm -r log libudf *.out
b=($(ls -d */))
echo "${b}"

c=noInitPVExtra/
fluent 3ddp -i hyperemic.jou -pinfiniband -t${SLURM_NTASKS} -g >stdout.out
    2>error.out
mkdir -p ${b}fluentResults/${c}
mv *.out log libudf *.cas *.dat *.xy *.sh ${b}fluentResults/${c}
cp *.c ${b}fluentResults/${c}
```

B.5 Post-processing

Listing B.5: Post processing script

.
```python
#Python script for generating mesh journal straight pipe
import os
import math
import re
import io
import openpyxl
import numpy as np
import matplotlib.pyplot as plt
from baselineFileGeneration import *

def bland_altman_plot(data1, data2, *args, **kwargs):
    data1     = np.asarray(data1)
    data2     = np.asarray(data2)
    mean      = np.mean([data1, data2], axis=0)
    diff      = data1 - data2                   # Difference between data1 and data2
    md        = np.mean(diff)                   # Mean of the difference
    sd        = np.std(diff, axis=0)            # Standard deviation of the difference

    plt.scatter(mean, diff, *args, **kwargs)
    plt.axhline(md,           color=’gray’, linestyle=’--’)
    plt.axhline(md + 1.95*sd, color=’gray’, linestyle=’--’)
    plt.axhline(md - 1.95*sd, color=’gray’, linestyle=’--’)
    plt.xlim(0,1)
    plt.ylim(-0.09 , 0.09)
    plt.xlabel(’FFR’)
    plt.ylabel(’Deviation’)
    return [md, sd]

def getFinalResults(patient, ffr, planes, **options):
    # Calculating the FFR values of the simulation
    # print(planes[’filename’])
    values = []
    aorticPressure = float(planes[’pressureValues’][-2])
    # print(ffr)
    # print(planes[’planeIDs’])
    for j in range(len(planes[’planeIDs’])):
        if planes[’planeIDs’][j].replace(’ID’,’’)==ffr[2]:
            val = float(planes[’pressureValues’][j])
            index = j
            break
            # print(index)
            # print(planes[’planeIDs’][j])
            # print(planes[’values’][j])
        # print(planes)
    # print (ffr[5])
    # print(aorticPressure)
    if abs(val/aorticPressure) >10 :
        print("THIS IS NOT OK")
    # print(val/aorticPressure-float(ffr[5]))
    # print(val/aorticPressure-float(ffr[4]))
    # print([float(ffr[5]), float(ffr[4]),val/aorticPressure,val/aorticPressure-float(ffr[5]), val/aorticPressure-float(ffr[4]) ])
    # print (ffr[2], planes['planeIDs'][j], planes['pressureValues'][j], options.get('indexpoint'))

    return ([float(ffr[5]), float(ffr[4]),val/aorticPressure,val/aorticPressure-float(ffr[5]), val/aorticPressure-float(ffr[4]) , aorticPressure])

def findCSVPressure(ffrID, ffrPath):
    # Finding the pressure in the FFR position of the FEM resultfile
    pout = []
    with open(ffrPath, 'r') as f:
        values = f.read().splitlines()
        # print (values)
    pin = float(values[1].split(',')[7])
    # print(pin)
    for i in range (len(values)):
        if values[i].split(',')[1] == ffrID:
            # print(ffrID, ' found at ', values[i].split(',')[1])
            # print(ffrID, ' at ', values[i])
            pout = float(values[i].split(',')[7])
            break
    if pout == []:
        print(ffrID , ffrPath)
    # print(pin, pout)
    return [pin, pout]

def getSimFilesAndWriteFFRResults(xlsxFile, xlsxFileSheetName,ffr, patients,outputName, **options):
    # Method to write out results in an EXcel file, currently not in use
    newSet = True
    if os.path.isfile(outputName):
        newSet = False
        xlsxFile = outputName
        # print(xlsxFile)
        wb = openpyxl.load_workbook(xlsxFile)
        sheetData = wb.get_sheet_by_name(xlsxFileSheetName)
        max_colum = sheetData.max_column -5
    else:
        wb = openpyxl.load_workbook(xlsxFile)
        sheetData = wb.get_sheet_by_name(xlsxFileSheetName)
        max_colum = sheetData.max_column


    max_row = sheetData.max_row
    caseList = []
    caseCount = 0

    names =  ['Clinical','FEM values', 'FVM Values', 'Clinical diff', 'Solver diff']
    # print(ffrValues[0][0][1])
    z = [""]*len(ffr[0])
    for i in range(len(ffr[0])):
        for j in range(len(ffr)):
```

```python
            z[i]= z[i]+str(ffr[j][i])+';'

    for row in range(max_row):
        if newSet == True:
            print ("newset true")
            for i in range(len(names)):
                sheetData.insert_cols(sheetData.max_column+1)
                sheetData.cell(row=1, column = sheetData.max_column+1, value = names[i])

            newSet = False

        if str(sheetData.cell(row=row+1, column=1).value) == patients['simuName']:
            # print(row)
            # print(sheetData.cell(row=row, column=1).value)
            # print(sheetData.cell(row=row + 1, column=1).value)
            # print('sheetData == simuname = true')
            for i in range (len(names)):
                a = sheetData.cell(row = row+1, column=max_colum+i+1,value = str(z[i]))
        #
    caseCount += 1
    wb.save(outputName)
    # print(outputName)
    return caseList

def plotFVMFEM (ffrListing, dirout, nFailed, **options):
    # Method to plot all relevant figures for each simulation
    if os.path.isdir(dirout)==False:
        os.system("mkdir -p {dir}".format(dir = dirout))
    fontsize = 12
    #plt.rcParams["svg.fonttype"] = "none"
    # plt.rc('text',usetex=True)
    if options.get('compare'):
        # simtype =  '$P_i-P_v$ vs $P_i$'
        simtype = '$R_{FEM}$ vs $R_{FVM}$'
        adj = [0]*len(ffrListing)
        conf = [0]*len(ffrListing)
        res1 = [0]*len(ffrListing)
        res2 = [0]*len(ffrListing)
        for i in range(len(ffrListing)):
            if ffrListing[i][0]['patientName'] != ffrListing[i][1]['patientName']:
                print(ffrListing[i][0]['patientName'], "misaligned")
                print(ffrListing[i][1]['patientName'], "misaligned")
                break
            else:
                adj[i] = ffrListing[i][0]['printValues'][2]
                conf[i] = ffrListing[i][1]['printValues'][2]
                res1[i] = ffrListing[i][0]['residuals'][0]
                res2[i] = ffrListing[i][1]['residuals'][0]
        plt.figure(4)
        plt.rcParams["axes.titlesize"] = fontsize
        [md , sd ] = bland_altman_plot(conf,adj)
        print('md', md, 'sd', sd)
        plt.title('D={0}, SD={1} S={2} F={3}'.format(np.round(md,4), np.round(sd,4), len(adj), nFailed))
        plt.savefig(dirout+'blandAltmanConfvsPV.png')

        plt.figure(1)
        plt.rcParams["axes.titlesize"] = fontsize
        plt.plot(conf,adj,'o')
        plt.plot([0.8, 0.8], [0,1],'--')
        plt.plot([0,1],[0.8, 0.8],'--')
        plt.plot([0,1],[0,1],'-')
        plt.axis([0,1,0,1])
        plt.ylabel('FFR FVM')
        plt.xlabel('FFR FVM ($R_{FEM}$)')
        plt.title ('D={0} SD={1} S={2} F={3}'.format(np.round(md,4), np.round(sd,4), len(adj), nFailed))
        plt.savefig(dirout+'ConfvsPV.png')
    else:
        simtype = dirout.replace('figures/','').replace('/','').capitalize()
        fem = [0]*len(ffrListing)
        fvm = [0]*len(ffrListing)
        clin = [0]*len(ffrListing)
        res = [0]*len(ffrListing)
        if options.get('residuals'):
            resid = options.get('residuals');
            # print(resid)
        else:
            resid = [0]*len(ffrListing)
        for i in range(len(ffrListing)):
            fem[i]=ffrListing[i][1]
            fvm[i]= ffrListing[i][2]
            clin[i] = ffrListing[i][0]
            res[i] = float(resid[i][0])
        print ('Clin > 0.8', sum((np.asarray(clin)>0.8)==True))
        print ('FEM > 0.8', sum((np.asarray(fem)>0.8)==True))
        print ('FVM > 0.8', sum((np.asarray(fvm)>0.8)==True))
        print('Diff = ',(sum((np.asarray(fvm)>0.8)==True)-sum((np.asarray(fem)>0.8)==True)) )
        plt.figure(4)
        plt.rcParams["axes.titlesize"] = fontsize
        [md , sd ] = bland_altman_plot(fem,fvm)
        plt.title('D={0} SD={1} S={2} F={3}'.format(np.round(md,4), np.round(sd,4), len(ffrListing), nFailed))
        plt.savefig(dirout+'blandAltmanFEMvsFVMfloat.png')


        if options.get('residuals'):
            diff = np.asarray(fem)-np.asarray(fvm)
            plt.figure(7)
            plt.plot(res,diff, 'o')
            plt.ylabel('FFR difference')
            plt.xlabel('Final residual')
            plt.title('Diff and residual  ')
            plt.xscale('log')
            plt.axhline(md,            color='gray', linestyle='--')
            plt.axhline(md + 1.645*sd, color='gray', linestyle='--')
            plt.axhline(md - 1.645*sd, color='gray', linestyle='--')
            plt.savefig(dirout+'residual.png')
        plt.figure(1)
        plt.rcParams["axes.titlesize"] = fontsize
        plt.plot(fem,fvm,'o')
        plt.plot([0.8, 0.8], [0,1],'--')
        plt.plot([0,1],[0.8, 0.8],'--')
        plt.plot([0,1],[0,1],'-')
        plt.axis([0,1,0,1])
        plt.ylabel('FVM FFR')
        plt.xlabel('FEM FFR')
        plt.title ('D={0} SD={1} S={2} F={3}'.format(np.round(md,4), np.
    round(sd,4), len(ffrListing), nFailed))
        plt.savefig(dirout+'FVMvsFEM.png')

        plt.figure(5)
        plt.rcParams["axes.titlesize"] = fontsize
        [md , sd ] = bland_altman_plot(clin,fvm)
        plt.title('D={0} SD={1} S={2} F={3}'.format(np.round(md,4), np.
    round(sd,4), len(ffrListing), nFailed))
        plt.savefig(dirout+'blandAltmanClinvsFVMfloat.png')

        plt.figure(2)
        plt.rcParams["axes.titlesize"] = fontsize
        plt.plot(clin,fvm,'o')
        plt.plot([0.8, 0.8], [0,1],'--')
        plt.plot([0,1],[0.8, 0.8],'--')
        plt.plot([0,1],[0,1],'-')
        plt.axis([0,1,0,1])
        plt.ylabel('FVM FFR')
        plt.xlabel('Clinical FFR')
        plt.title ('D={0} SD={1} S={2} F={3}'.format(np.round(md,4), np.
    round(sd,4), len(ffrListing), nFailed))
        plt.savefig(dirout+'FVMvsClinical.png')

        plt.figure(6)
        [md , sd ] = bland_altman_plot(clin,fem)
        plt.title('D={0}, SD ={1}'.format(np.round(md,4), np.round(sd,4)))
        plt.savefig(dirout+'blandAltmanClinvsFEMfloat.png')

        plt.figure(3)
        plt.plot(clin, fem,'o')
        plt.plot([0.8, 0.8], [0,1],'--')
        plt.plot([0,1],[0.8, 0.8],'--')
        plt.plot([0,1],[0,1],'-')
        plt.axis([0,1,0,1])
        plt.ylabel('FEM FFR')
        plt.xlabel('Clinical FFR')
        plt.title('D={0} SD={1}'.format(np.round(md,4), np.round(sd,4)))
        plt.savefig(dirout+'FEMvsClinical.png')

    plt.close('all')
    return("Plotted "+simtype)


def prepCSVresults (oldCSV, results):
    # Method to write out a CSV with the results from the FLUENT simulations
    # This method is currently loosing a lot of data. Either because the lines are to long, or that there are a lot of planes that are not generated
    # Using spheres would mitigate the last part, but something with the method needs to be fixed to handle the first part
    q = 0
    count = 0
    with open(oldCSV, 'r') as f:
        csv = f.read().splitlines()
    for i in range (len(csv)):
        if i ==0:
            CSVText = csv[i]
        else:
            line = csv[i].split(',')
            if i ==1:
                CSVText+="""
{0},{1},{2},{3},{4},{5},{6},{7}
""".format(line[0], line[1], line[2], line[3], line[4], line[5], results[0]['flowValues'][-1], results[1]['pressureValues'][-1])
            else:
                for j in range(q,len(results[0]['flowValues'])):
                    if results[0]['planeIDs'][-j].replace('ID','')== line[1]:
                        CSVText+="""{0},{1},{2},{3},{4},{5},{6},{7}
""".format(line[0], line[1], line[2], line[3], line[4], line[5], results[0]['flowValues'][j], results[1]['pressureValues'][j])
                        q=j
                        break
                    if j ==len(results[0]['flowValues'])-1:
                        count+=1
                        print('Lost ID ', line[1])
    if count > 20:
        print(results[0]['planeIDs'])
        print(csv)
    return CSVText

def checkReNumbers():
    # method to read Re numbers in FEM results
    listofHypCases = copyScript.passVar('1D_3D_TAG_HYP.xlsx')
    rmin = [[10000] for d in range(len(listofHypCases))]
    rmax = [0 for d in range(len(listofHypCases))]
    # print(rmax)
    for i in range (len(listofHypCases)):
        csvPath = '../../'+listofHypCases[i]['patientName']+'/'+listofHypCases[i]['patientName']+'_Simulation/'+listofHypCases[i]['simuName']+'/'+listofHypCases[i]['simuName']+'_out/ctlResults/ctlSol_Average.csv'
        with open(csvPath,'r') as f:
            values= f.read().splitlines()
        for j in range(1,len(values)):
            # print(values[j].split(','))
            if values[j].split(',')[5]==0:
                continue
            rad =float(values[j].split(',')[5])
            u = (float(values[j].split(',')[6])*1e-5)/(np.pi*(float(values[j].split(',')[5]))**2)
            re = 2*rad*1050*u/0.0035
            # print(rmax[i])
            if re>=rmax[i]:
                rmax[i]= re
            elif re<=rmin[i]:
                rmin[i]=re
    # print (rmin[0],rmax[0])
    return [rmin, rmax]
def getDiagnosticRelevance(results):
    # Interpreting resuls and printing out medically relevant diagnostic tools
    TP = 0
    FP = 0
    FN = 0
    TN = 0
    for i in range(len(results)):
        # print(results[i][0])
        # print(results[i][2])
        if results[i][0]>0.8 and results[i][2]> 0.8:
            TN+=1
        elif results[i][0]<0.8 and results[i][2]> 0.8:
            FN+=1
        elif results[i][0]<0.8 and results[i][2]< 0.8:
            TP+=1
        elif results[i][0]>0.8 and results[i][2]< 0.8:
            FP+=1
    print(TP,FP, FN, TN)
    TPR=float(TP/float(TP+FN))
    TNR=float(TN/float(TN+FP))
    PPV=float(TP/float(TP+FP))
    NPV=float(TN/float(TN+FN))
    print('TPR, TNR, PPV, NPV')
    return [TPR, TNR, PPV, NPV]

def processHypCases(sims,**options):
    # Processing the hyperemic cases based on inputfactors
    ffrValues = [[]]
    listofcases = copyScript.passVar('1D_3D_TAG_BLN.xlsx')
    listofHypCases = copyScript.passVar('1D_3D_TAG_HYP.xlsx')
    slurmfolder=''
    target = '../../'
    directoryList = os.listdir(target)
    indexes = getCaseIndexes(listofHypCases,directoryList)

    printValues=[]

    # indexes.append(27)
    print(indexes)
    # indexes= [69, 0, 41 , 42]
    filesToProcess = ['flowPlanes.out', 'pressurePlanes.out']
    iter = 0
    oldit = 0
    count = 0
    maxvel = 0
    res = []
    time = []
    ffrdiff = [0,10000,0,0]
    pinDiff = [0,10000, [],[]]
    simVersion = sims
    printDict = []
    caselist = []
    for i in range (len(indexes)): #(len(indexes)):
    #     # resultPath = target+listofcases[indexes[i]]['patientName']+'/'+listofcases[indexes[i]]['patientName']+'_Simulation/'+listofcases[indexes[i]]['simuName']+'/'+listofcases[indexes[i]]['simuName']+'_out/fluentResults'
        resultPath = target+listofHypCases[indexes[i]]['patientName']+'/'+listofHypCases[indexes[i]]['patientName']+'_Simulation/'+listofHypCases[indexes[i]]['simuName']+'/'+listofHypCases[indexes[i]]['simuName']+'_out/fluentResults/'+simVersion
        meshPath = target+listofHypCases[indexes[i]]['patientName']+'/'+listofHypCases[indexes[i]]['patientName']+'_Mesh/'+listofHypCases[indexes[i]]['patientName']+'_Mesh_'+listofHypCases[indexes[i]]['meshNumber']+'/'
        ffrPath = meshPath+listofHypCases[indexes[i]]['patientName']+'_Mesh_'+listofHypCases[indexes[i]]['meshNumber']+'_FFR'
        if (os.path.isfile(resultPath+'simResults.dat') and os.path.isfile(resultPath+'flowPlanes.out') ) or (('poly' in resultPath or 'Init' in resultPath) and os.path.isfile(resultPath+'flowPlanes.out')):
            # print(listofHypCases[indexes[i]]['simuName'], "with mesh" , listofHypCases[indexes[i]]['meshNameVTK'], " Succeeded")
            # print (listofHypCases[indexes[i]])
            [planeValues, extraValues] = readResults(resultPath, postProcess=filesToProcess)
            if options.get('writeCSV'):
                csvPath = '../../'+listofHypCases[i]['patientName']+'/'+listofHypCases[i]['patientName']+'_Simulation/'+listofHypCases[i]['simuName']+'/'+listofHypCases[i]['simuName']+'_out/ctlResults/ctlSol_Average.csv'
                outvalue = prepCSVresults(csvPath, planeValues)
                # print(outvalue)
                # writeFile(outvalue, resultPath+'solution.csv')
            if extraValues[2]> maxvel:
                maxvel = extraValues[2]
            time.append(float(extraValues[3]))
            ffrValues[0]= readFFRFiles(ffrPath)
            indexrange = listofHypCases[indexes[i]]['FFR_Num']
            if listofHypCases[indexes[i]]['simuName']=='CT_FFR_48_Simulation_0010':
                indexrange = indexrange[1:]
                print(indexrange)
            # if simVersion == 'regular':
            #     print(indexrange)
            for j in range(len(indexrange)):
                if listofHypCases[indexes[i]]['patientName'] == 'CT_FFR_38' or listofHypCases[indexes[i]]['patientName'] == 'CT_FFR_14' or listofcases[indexes[i]]['patientName'] == 'CT_FFR_40':
                    indexrange[j] = int(indexrange[j])-1
                # print(ffrValues[0][int(indexrange[j])-1])
                csvPressure = findCSVPressure(ffrValues[0][int(indexrange[j])-1][2], resultPath.replace('fluentResults/'+simVersion, 'ctlResults/ctlSol_Average.csv'))
                # print (csvPressure)
                printValues.append(getFinalResults(listofHypCases[indexes[i]], ffrValues[0][int(indexrange[j])-1], planeValues[1], indexpoint = int(indexrange[j]), floatFFR = csvPressure))
                printValues[iter][1] = csvPressure[1]/csvPressure[0]
                if abs(csvPressure[0]-printValues[iter][5]) > abs(pinDiff[0]):
                    pinDiff[0] =csvPressure[0]-printValues[iter][5]
                    pinDiff[2] = indexes[i]
                elif abs(csvPressure[0]-printValues[iter][5]) < pinDiff[1]:
                    pinDiff[1] = abs(csvPressure[0]-printValues[iter][5])
                    pinDiff[3] = indexes[i]
                if abs(printValues[iter][1]-printValues[iter][2])>0.1:
                    print(listofHypCases[indexes[i]]['simuName'], "with mesh" , listofHypCases[indexes[i]]['meshNameVTK'], " gave diff = ", printValues[iter][1]-printValues[iter][2], "Residual =", extraValues[4])
                    if abs(printValues[iter][1]-printValues[iter][2])>10:
                        print(listofHypCases[indexes[i]]['simuName'], "was scrapped")
                        printValues[iter]=[0,0,0,0,0]
                printDict.append({
                            'patientName' : listofHypCases[indexes[i]]['patientName']+'_'+str(indexrange[j]),
                            'printValues' : printValues[iter],
                            'residuals' : extraValues[4],
                            'simulationTime' : extraValues[3]
                            })
                if abs(printValues[iter][1]-printValues[iter][2])>ffrdiff[0]:
                    ffrdiff[0]= abs(printValues[iter][1]-printValues[iter][2])
                    ffrdiff[2]= abs(printValues[iter][1]-printValues[iter][2])/printValues[iter][1]
                if abs(printValues[iter][1]-printValues[iter][2])<ffrdiff[0]:
                    ffrdiff[1]= (printValues[iter][1]-printValues[iter][2])
                    ffrdiff[3]= abs(printValues[iter][1]-printValues[iter][2])/printValues[iter][1]
                res.append(extraValues[4])
                iter+=1
            # getSimFilesAndWriteFFRResults('1D_3D_TAG_HYP.xlsx', "Sheet", printValues[oldit:iter], listofHypCases[indexes[i]], 'output.xlsx', iter = i)
            oldit = iter
        else:
            print(listofHypCases[indexes[i]]['simuName'], "did not succeed")
            count +=1
        # print(ffrValues)
    # print(['Clinical','FEM values', 'FVM Values', 'Clinical diff', 'Solver diff'])
    # print(['Vessel', 'CTL ID', 'PointID', 'Stenosis ID','FFR FEM', 'FFR Measured'])
```

```
    # print(printValues)

    print(len(printValues), "functional FFR")
    print(count, "non-functional Sims")
    print(np.mean(time))
    print('Max velocity = ' , maxvel   )
    print(plotFVMFEM(printValues,'figures/'+simVersion, 106-len(printValues),residuals=res))
    print(caselist)
    print(pinDiff)
    print(listofHypCases[pinDiff[2]])
    print(listofHypCases[pinDiff[3]])
    print(getDiagnosticRelevance(printValues))
    print(ffrdiff)
    # ffrValues = ffrValues[0]
    # print(planeValues[1]['filename'])
    # print(listofHypCases[27]['simuName'])
    # print(ffrValues)
    #     if simulationType=='baseline':
    #         [journal, folderPath] = simulationPrep(listofcases[indexes[i]],len(indexes),simtype = simulationType, iterations = simLength)
    #     elif simulationType =='hyperemic':
    #         [journal, folderPath] = simulationPrep(listofHypCases[27],len(indexes),simtype = simulationType,baselinePath=listofcases[27]['simuName'], udfName =udfName, iterations = simLength)
    #     print(listofcases[indexes[i]])
    #     writeFile(journal, folderPath+'/simulation.jou')
    #     iter+=1
    #     slurmfolder+=folderPath+' '
    # slurm = slurmSimulationGeneration(slurmfolder,len(indexes)-1,listofcases[indexes[i]])
    # # print (slurm)
    # writeFile(slurm,'../slurm/'+simulationType+'Queue.slurm' )
    return printDict

def processBaselineResults(output,**options):
    # Processing baseline cases with some modifications to not follow subfolders and print somewhat different
    ffrValues = [[]]
    listofcases = copyScript.passVar('1D_3D_TAG_BLN.xlsx')
    slurmfolder=''
    target = '../../'
    directoryList = os.listdir(target)
    indexes = getCaseIndexes(listofcases,directoryList)
    printValues=[]

    # indexes.append(27)
    # indexes= [69, 0, 41 , 42]
    filesToProcess = ['flowPlanes.out', 'pressurePlanes.out']
    iter = 0
    oldit = 0
    count = 0
    maxvel = 0
    res = []
    time = []
    simVersion = output
```

```python
    printDict = []
    for i in range (len(indexes)): #(len(indexes)):
    #     # resultPath = target+listofcases[indexes[i]]['patientName']+'/'+listofcases[indexes[i]]['patientName']+'_Simulation/'+listofcases[indexes[i]]['simuName']+'/'+listofcases[indexes[i]]['simuName']+'_out/fluentResults'
        resultPath = target+listofcases[indexes[i]]['patientName']+'/'+listofcases[indexes[i]]['patientName']+'_Simulation/'+listofcases[indexes[i]]['simuName']+'/'+listofcases[indexes[i]]['simuName']+'_out/fluentResults/'+simVersion
        meshPath = target+listofcases[indexes[i]]['patientName']+'/'+listofcases[indexes[i]]['patientName']+'_Mesh/'+listofcases[indexes[i]]['patientName']+'_Mesh_'+listofcases[indexes[i]]['meshNumber']+'/'
        ffrPath = meshPath+listofcases[indexes[i]]['patientName']+'_Mesh_'+listofcases[indexes[i]]['meshNumber']+'_FFR'
        if (os.path.isfile(resultPath+'simResults.dat') and os.path.isfile(resultPath+'flowPlanes.out') ) or (('poly' in resultPath or 'Init' in resultPath) and os.path.isfile(resultPath+'flowPlanes.out')):
            # print(listofcases[indexes[i]]['simuName'], "with mesh" , listofcases[indexes[i]]['meshNameVTK'], " Succeeded")
            print (listofcases[indexes[i]]['simuName'])
            [planeValues, extraValues] = readResults(resultPath, postProcess=filesToProcess)
            if extraValues[2]> maxvel:
                maxvel = extraValues[2]
            time.append(float(extraValues[3]))
            ffrValues[0]= readFFRFiles(ffrPath)
            indexrange = listofcases[indexes[i]]['FFR_Num']
            if listofcases[indexes[i]]['simuName']=='CT_FFR_48_Simulation_0008':
                indexrange = indexrange[1:]
                print(indexrange)
            for j in range(len(indexrange)):
                if listofcases[indexes[i]]['patientName'] == 'CT_FFR_38' or listofcases[indexes[i]]['patientName'] == 'CT_FFR_14' or listofcases[indexes[i]]['patientName'] == 'CT_FFR_40' :
                    indexrange[j] = int(indexrange[j])-1
                # print(ffrValues[0][int(indexrange[j])-1])
                csvPressure = findCSVPressure(ffrValues[0][int(indexrange[j])-1][2], resultPath.replace('fluentResults/'+simVersion, 'ctlResults/ctlSol_Average.csv'))
                # print (csvPressure)
                printValues.append(getFinalResults(listofcases[indexes[i]], ffrValues[0][int(indexrange[j])-1], planeValues[1], indexpoint = int(indexrange[j]), floatFFR = csvPressure))
                printValues[iter][1] = csvPressure[1]/csvPressure[0]
                if abs(printValues[iter][1]-printValues[iter][2])>0.01:
                    print(listofcases[indexes[i]]['simuName'], "with mesh" , listofcases[indexes[i]]['meshNameVTK'], " gave diff = ", printValues[iter][1]-printValues[iter][2], "Residual =", extraValues[4])
                    if abs(printValues[iter][1]-printValues[iter][2])>10:
                        print(listofcases[indexes[i]]['simuName'], "was scrapped")
                        printValues[iter]=[0,0,0,0,0]
                printDict.append({
```

```
                            'patientName' : listofcases[indexes[i]]['patientName']+'_'+str(indexrange[j]),
                            'printValues' : printValues[iter],
                            'residuals' : extraValues[4],
                            'simulationTime' : extraValues[3]
                            })
                res.append(extraValues[4])
                iter+=1
            # getSimFilesAndWriteFFRResults('1D_3D_TAG_HYP.xlsx', "Sheet", printValues[oldit:iter], listofcases[indexes[i]], 'output.xlsx', iter = i)
            oldit = iter
        else:
            print(listofcases[indexes[i]]['simuName'], "did not succeed")
            count +=1

    print(len(printValues), "functional FFR")
    print(count, "non-functional Sims")
    print(np.mean(time))
    print('Maxvel = ', maxvel)
    print(plotFVMFEM(printValues,'figures/baseline/'+output, 106-len(printValues),residuals=res))
    return "Processed baseline cases"

if __name__=='__main__':
    checkRe = False
    # Only check the Reynolds numbers
    writeCSV = False
    # turn on writing of CSV from results
    simVersions = ''
    subpath = ''
    subpath = 'poly/'
    # simVersions = ['pVAdjusted/' , 'confFile/','regular/', 'lowInit/']
    # simVersions = ['noInit250/','noInit500/', 'noInit1000/', 'noInit2000/', 'noInit/']
    # simVersions = ['pVAdjusted/' , 'confFile/', 'noInit/', 'noInitConf/']
    # simVersions = ['lowInit/', 'noInitConf/']
    simVersions = ['noInitExtra/','noInitConfExtra/']
    # simVersions = ['noInitExtra/', 'noInit/']
    results = []
    if simVersions == '':
        print(processBaselineResults(subpath))
    elif checkRe == True:
        [rmin, rmax] = checkReNumbers()
        plt.figure(1)
        plt.hist(rmin)
        plt.xlabel('Reynold\'s number')
        plt.ylabel('Instances')
        plt.title('Minimum domain value \n Mean = {0}'.format(int(np.mean(rmin))))
        print('Saving figures/RE/rmin.png')
        plt.savefig('figures/RE/rmin.png')
        plt.clf()
        plt.hist(rmax)
        plt.xlabel('Reynold\'s number')
        plt.ylabel('Instances')
        print(min(rmax))
        plt.title('Maximum domain value \n Mean = {0} Max = {1}'.format(int(np.mean(rmax)),int(max(rmax))))
        print('Saving figures/RE/rmax.png')
        plt.savefig('figures/RE/rmax.png')
    else:
        for i in range (len(simVersions)):
            results.append(processHypCases(subpath+simVersions[i], writeCSV=writeCSV))
        ffrs = []
        i1 = 0
        i2 = 1
        if len(results)>1:
            for i in range (len(results[i2])):
                for j in range (len(results[i1])):
                    if results[i2][i]['patientName'] == results[i1][j]['patientName']:
                        ffrs.append([ results[i2][i],  results[i1][j]])
                        # print(results[i2][i]['patientName'] )
                        # print(results[i1][j]['patientName'])
            # print(results)
            plotFVMFEM(ffrs, "figures/"+subpath, 106-len(ffrs), compare=True)
            print(len(ffrs))
        # print(processBaselineResults())
```