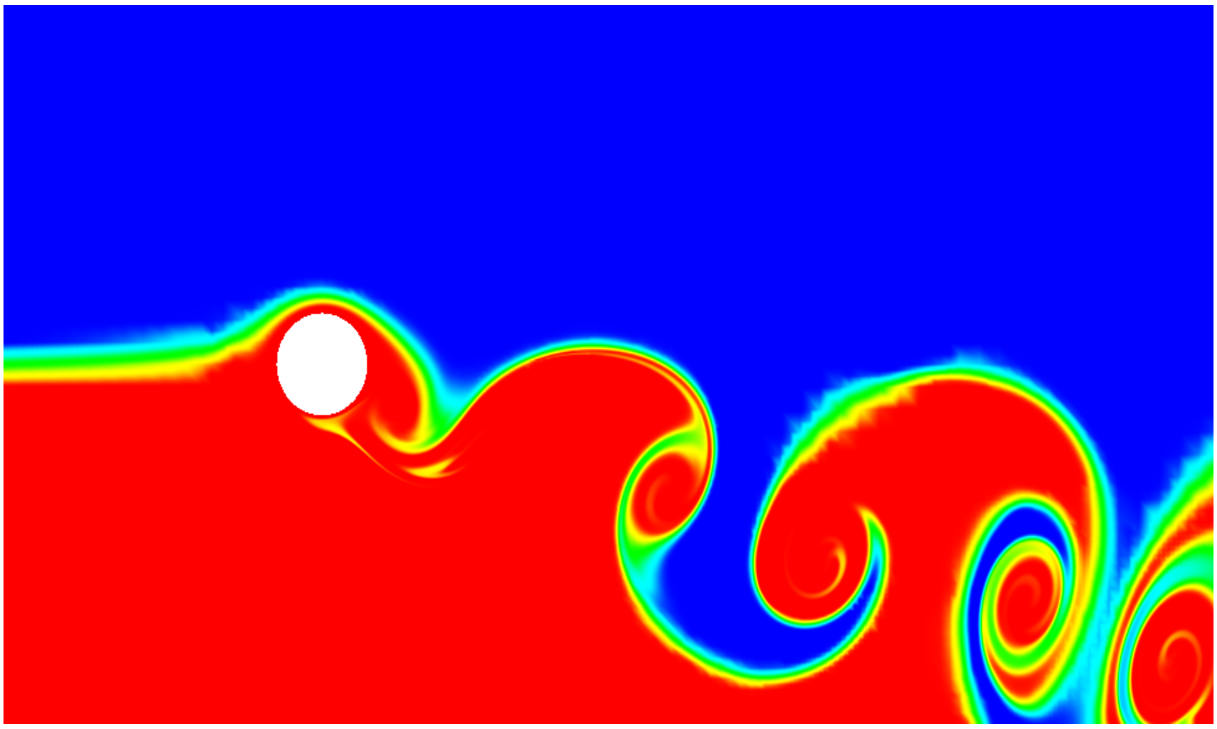
CHALMERS



Benchmark and validation of Open Source CFD codes, with focus on compressible and rotating capabilities, for integration on the SimScale platform.

Master's Thesis in Engineering Mathematics & Computational Sciences

Magnus Winter

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2013

Master's Thesis 2013:81

MASTER'S THESIS IN ENGINEERING MATHEMATICS

Benchmark and validation of Open Source CFD codes, with focus on compressible and rotating capabilities, for integration on the SimScale platform.

MAGNUS WINTER

Department of Applied Mechanics
Division of Fluid Dynamics
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2013

Benchmark and validation of Open Source CFD codes, with focus on compressible and rotating capabilities, for integration on the SimScale platform.
MAGNUS WINTER

Master's Thesis no 2013:81
ISSN 1652-8557
Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: Rotating cylinder ($\omega = 1.2733$ [rad/s]) in a free stream flow with Reynolds number $Re = 333.33$ and at the time $t = 17.5$ [s], calculated using Gerris.

Chalmers Reproservice
Göteborg, Sweden 2013

Benchmark and validation of Open Source CFD codes, with focus on compressible and rotating capabilities, for integration on the SimScale platform.
Magnus Winter
Department of Applied Mechanics
Division of Fluid Dynamics
Chalmers University of Technology

Abstract

The use of CFD is widely spread in industry today. However, smaller business have difficulty to make use of this tool, due to high costs for commercial licenses and hardware. As SimScale provides the combination of cloud computing with open source CFD software, the prerequisites to use CFD are lowered. To provide more diverse solvers for the platform, different CFD software are integrated onto the platform.

As OpenFOAM was the sole CFD software on the platform at the start of this thesis, two other open source software were researched and compared to OpenFOAM. OpenFOAM was compared to Gerris and SU^2 in three different validation cases.

OpenFOAM and Gerris rotating body capabilities are investigated in an analytical solvable case of laminar Taylor-Couette flow. Two different approaches were used in OpenFOAM, where AMI (Adaptive Mesh Interface) and MRF (Multiple Reference Frame) were used to solve the problem.

OpenFOAM and SU^2's turbulent capabilities are compared using a turbulent flat plate case from Wieghardt [1]. Compressible transonic flow over a RAE2822 airfoil was set up for OpenFOAM and SU^2. The results were compared to data from NASA [2] It was found that the rotating body capabilities (MRF and AMI) of OpenFOAM were suitable for integration onto the SimScale platform. Gerris as a software was deemed to be easy to work with, but would require a new work flow on the platform to be integrated. SU^2 was integrated onto the platform during this thesis work, but was deemed to still suffer from some early development issues as its first release was done in January 2013.

Keywords: OpenFOAM, SU^2, Gerris, Turbulent flat plate, Taylor-Couette, RAE2822, RANS, DNS, AMI, MRF

Acknowledgements

The conducted thesis was done at SimScale GmbH in Garching bei München. First and foremost I would like to express my gratitude to my supervisor at SimScale, Johannes Probst, for always making time to help me when I encountered problems. Also I would like to express my thanks to David Heiny for giving me the opportunity to conduct my thesis at SimScale and to my colleagues at SimScale for keeping an uplifting and fun atmosphere at the workplace.

I also like to thank my supervisor at Chalmers, Lars Davidsson, for his support and help during this thesis.

To all the developers of the open source software used in this thesis including but not limited to, SU^2, OpenFOAM, Gerris, Salome, Eclipse, SVN, Kile, LaTeX and Ubuntu. Finally to my friends in Gothenburg and Munich for being helpful distractions from CFD.

Magnus Winter, Munich 7/12/13

Nomenclature

Abbreviation

AMI	Arbitrary Mesh Interface
BiCGSTAB	Biconjugate Gradient Stabilized
BC	Boundary Condition
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrich-Lewy number
CGS	Conjugate Gradient Squared
DNS	Direct Numerical Simulation
GAMG	Generalized Geometric-Algebraic Multigrid
GMRES	Generalized Minimum Residual Method
ILU	Incomplete Lower Upper factorization
LES	Large Eddy Simulation
LUDS	Linear Upwind Scheme
MAC	Marker and Cell
MRF	Multiple Reference Frame
OpenFOAM	Open source Field Operation And Manipulation
PISO	Pressure-Implicit Split-Operator
QUICK	Quadratic Upwind Scheme
RANS	Reynolds Averaged Navier-Stokes
SA	Spalart-Allmaras
SIMPLE	Semi-Implicit Method for Pressure Linked Equations
SU^2	Stanford University Unstructured
TVD	Total Variation Diminishing

Dimensionless quantities

C_f	Skin Friction Coefficient
C_P	Pressure Coefficient
M	Mach number
Pr	Prandtl number

Greek symbols

δ	Kronecker delta
ρ	Density
γ	Isentropic Expansion Factor
μ	Dynamic viscosity
ν	Kinematic viscosity
∂	Partial derivative
ω	Specific turbulent dissipation
τ_{ij}	Viscous stress tensor
ϵ	Turbulent dissipation

Roman symbols

C_D	Coefficient of Drag
C_L	Coefficient of Lift
E	Energy
R	Specific gas constant
C_V	Heat capacity at constant volume
q	Heat flux
n_i	Surface normal gradient
p	Pressure
T	Temperature
t	Time
k	Turbulent kinetic energy
u_i	Velocity vector
x	Space variable

Subscripts

∞	Free stream property
i,j,k	Tensor indices
t	Turbulence

Contents

1 Introduction

The usage of Computational Fluid Dynamics (CFD) is widely spread today. As CPU's become more powerful and affordable, most larger companies in industry are using it today. However, investing in the required hardware and commercial licenses is still a hurdle for smaller businesses to use CFD.

Open source softwares provide a cheap approach to simulations, compared to commercial software. However, the open source softwares are dependent on a more knowledgeable user than for the commercial softwares, as more freedom is provided with the software and documentation can be limited. Also, as mentioned previously, another limitation for small businesses is the need for computing power to perform simulations without having to invest in the hardware. The start-up SimScale GmbH are currently developing a web platform where users can use open source CFD software with an user interface and computer clusters. This cloud computing approach provides small businesses with the required computational power needed with the user friendliness of commercial softwares.

As different open source CFD softwares have their strengths and weaknesses, providing a variety of solvers to the customer is of importance. A short introduction of a subset of the large variety of open source CFD softwares can be seen below.

SU^2 is developed by Stanford University and is a acronym for Stanford University Unstructured. The first release of SU^2 was done in January of 2013. Thus SU^2 still undergoes development and is set to release version 3.0 in January 2014. It is developed in C++ [3].

Gerris is developed by Stéphane Popinet and supported by the National Institute of Water and Atmospheric research in New Zealand. Its first release was in 2003. It is developed in C [4].

OpenFOAM is produced by OpenCFD Ltd. It contains over 80 solver applications to simulate specific problems in engineering. It has been released as open source since 2004. The code is developed in C++ [5].

1.1 Objective

At the starting point of this thesis, OpenFOAM was the main fluid solver on the SimScale platform. To provide more capabilities and different solvers on the platform an evaluation of the open source CFD softwares SU^2 and Gerris was tasked. The main focus was to investigate new compressible capabilities and dynamic mesh operations, mainly rotational motion, to the platform. Also a possible integration onto the SimScale platform was to be investigated and, if deemed possible, to integrate the software onto the platform.

The versions of the softwares used were OpenFOAM version 2.2.0, SU^2 version 2.0.8 and Gerris version 1.3.2.

1.2 Limitations

Only one compressible solver in OpenFOAM is compared to SU^2. There are several other compressible solvers including steady solvers. The *rhoCentralFoam* solver was chosen due to its similar set up as the compressible solver in SU^2, as it is also a density based solver. Comparing SU^2 with other compressible solvers from OpenFOAM such as *rhoSimpleFoam* is left out from this thesis.

The incompressible solver of SU^2 was found to be difficult to find convergence with. In the end, only the compressible solver from SU^2 was used.

A part of the SU^2 code was integrated on the SimScale platform in the course of this thesis. Due to trade secrets of SimScale only the outline of the integration will be published in this paper.

1.3 Outline of Thesis

The thesis was conducted as a combined literature study of the capabilities of the different softwares and testing the capabilities of the solver against suitable benchmark cases. Theory, covering the fundamentals of the numerics of CFD softwares, is given in section 2 and referenced in the subsequent secctions when comparing the capabilities of the different softwares. An overview of capabilities and strengths and weaknesses of the softwares are done in section 3 of the thesis. The benchmark cases conducted with the softwares are displayed in section 4. Recommendations, conclusions and future work is discussed in section 5.

2

Theory

2.1 Governing equations

The governing equations for fluids consists of a continuity equation (2.1), a momentum equation (2.2), an energy balance (2.3) and a state equation, connecting density to pressure. If the fluid is assumed to be Newtonian the equations reduce to,

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0 \tag{2.1}$$

$$\left(\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j}\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij}\right) + \rho f_i \tag{2.2}$$

$$\left(\frac{\partial \rho E}{\partial t} + \frac{\partial \rho u_j E}{\partial x_j}\right) = -\frac{\partial p u_j}{\partial x_j} + \frac{\partial u_i \tau_{ji}}{\partial x_j} + \frac{\partial}{\partial x_j}\left(k\frac{\partial T}{\partial x_j}\right) + S_E \tag{2.3}$$

In the above equations, f_i is an force applied on the fluid, S_E is an energy source term and the tensor $\tau_{ij} = \mu(\partial u_i/\partial x_j + \partial u_j/\partial x_i) - \frac{2}{3}\mu\partial u_k/\partial x_k$. Equations of state relate pressure $p = p(\rho,T)$ and internal energy $i = i(\rho,T)$ to the variables ρ and T. Thus allowing the above equations to be solved. An example of this relation is the equations of state for an ideal gas, [6]

$$p = \rho R T \;\text{ and }\; i = C_v T. \tag{2.4}$$

2.1.1 Incompressible fluids

At low Mach numbers, compressible effects can be neglected. As the continuity equation (2.1) reduces to

$$\frac{\partial u_j}{\partial x_j} = 0, \tag{2.5}$$

there is no need to solve the energy equation as there is no link between the energy equation and momentum and continuity equations. As a result the calculated pressure field is not unique. After simplifying the governing equations using the incompressible condition (2.5), the following system of equations is received, [6]

$$\frac{\partial \rho u_j}{\partial x_j} = 0 \tag{2.6}$$

$$\left(\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j}\right) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\mu \frac{\partial u_i}{\partial x_j}\right) + \rho f_i. \tag{2.7}$$

2.2 Modeling

To solve the governing equations some simplifications have to be made. To remove dependence on small fluctuations usually the Reynolds Averaged Navier-Stokes assumption is used. There are other models, more advanced (and computationaly heavy) as LES and DNS, but they will not be covered in this thesis.

2.2.1 RANS

To model the Reynolds Averaged Navier Stokes equation, all variables are split into a time-averaged part and a fluctuating part, $\phi = \bar{\phi} + \phi'$. The time-averaged part is calculated as, $\bar{\phi} = \frac{1}{T}\int_T \phi(x,t)dt$. For compressible flows often another form of decomposition is done using Favre-averaging. Here variables are decomposed as $\theta = \tilde{\theta} + \theta''$, where $\tilde{\theta} = \overline{\rho\theta}/\bar{\rho}$. Thus θ'' not only includes the turbulent fluctuations but also the density fluctuations. After Favre averaging the velocity and energy and performing a standard time-averaging for ρ and p, the following equations are derived.

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_i}{\partial x_i} = 0 \tag{2.8}$$

$$\left(\frac{\partial \bar{\rho}\tilde{u}_i}{\partial t} + \frac{\partial \bar{\rho}\tilde{u}_j\tilde{u}_i}{\partial x_j}\right) = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\overline{\tau_{ij}} - \overline{\rho u_i'' u_j''}\right) \tag{2.9}$$

Here $\tau_{ij} = \left(\mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\mu\frac{\partial u_k}{\partial x_k}\delta_{ij}\right)$ and $\overline{\tau_{ij}} = \widetilde{\tau_{ij}} + \overline{\tau_{ij}''}$. For the energy equation a similar tranformation is done with the resulting Favre averaged equation,

$$\left(\frac{\partial \overline{\rho}\tilde{E}}{\partial t} + \frac{\partial \overline{\rho}\tilde{u}_j\tilde{E}}{\partial x_j}\right) = -\frac{\partial \overline{p}\tilde{u}_j}{\partial x_j} + \frac{\partial \overline{u_i\tau_{ji}}}{\partial x_j} - \frac{\partial}{\partial x_j}\left(\overline{q_j}\right) - \frac{\partial}{\partial x_j}\overline{u_j'' p} - \frac{\partial}{\partial x_j}\overline{\rho u_j'' E''}. \tag{2.10}$$

This is the decomposition made in compressible codes. For incompressible codes the Favre decomposition is not used and the RANS equations are derived from the standard time-averaging. For solving the RANS equations assumptions have to be made regarding the turbulent terms. The equations solved can be found in section 3 of the respective softwares (SU2 and OpenFoam) [6].

2.2.2 Turbulence models

When modeling the governing equations with RANS, the need to model the turbulent scales is apparent. There are several models which deal with how to model turbulence. Here only the Spalart-Allmaras and the $k-\omega$ SST model will be reviewed.

2.2.2.1 Spalart-Allmaras

The turbulence model Spalart-Allmaras is an one equation model, where the kinematic eddy viscosity is calculated through a transport equation and a length scale is found from an algebraic expression. The model is a cheap way of calculating the boundary layers in aerodynamics. In the Spalart-Allmaras model an eddy viscosity parameter $\widetilde{\nu}$ is calculated. It is related to the eddy viscosity through,

$$\nu_t = \widetilde{\nu} f_{\nu_1}. \tag{2.11}$$

The Reynolds stresses are calculated using the following assumption,

$$-\rho\overline{u_i'u_j'} = \rho\widetilde{\nu} f_{\nu_1}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right). \tag{2.12}$$

A transport equation is set up for $\widetilde{\nu}$ to find the eddy viscosity,

$$\frac{\partial \rho\widetilde{\nu}}{\partial t} + \frac{\partial \rho\widetilde{\nu}\bar{u}_k}{\partial x_k} = \frac{1}{\sigma_\nu}\left[\frac{\partial}{\partial x_k}\left((\mu+\rho\widetilde{\nu})\frac{\partial \widetilde{\nu}}{\partial x_k}\right) + C_{b_2}\rho\frac{\partial \widetilde{\nu}}{\partial x_k}\frac{\partial \widetilde{\nu}}{\partial x_k}\right] + C_{b_1}\rho\widetilde{\nu}\widetilde{\Omega} - C_{w_1}\rho\left(\frac{\widetilde{\nu}}{\kappa y}\right)^2 f_w \tag{2.13}$$

Where $\widetilde{\Omega} = \Omega + \frac{\widetilde{\nu}}{(\kappa y)^2} f_{\nu_2}$, and Ω is the mean vorticity. The wall functions are dependent on the following variables $f_{\nu_2} = f_{\nu_2}(\widetilde{\nu}/\nu)$ and $f_w = f_w(\widetilde{\nu}/(\widetilde{\Omega}\kappa^2 y^2))$. The turbulent length scale can be found from κy, where y is the distance from the wall.

The model constants are set as $\sigma_\nu = 2/3$, $\kappa = 0.4187$, $C_{b_1} = 0.1355$, $C_{b_2} = 0.622$ and $C_{w_1} = 0.56203$ [6]. There are further coefficients in the wall functions, not given here. For all the model details, see [7].

2.2.2.2 $k-\omega$ SST

The $k-\omega$ SST model is a mix of the $k-\omega$ and $k-\epsilon$ models. In the near-wall region the $k-\omega$ model is used and further away from the wall in the fully turbulent regions the $k-\epsilon$ method is used. The eddy viscosity is

$$\mu_t = \rho k/\omega,$$

and the Reynolds stresses are derived from the Boussinesq assumption.

$$\tau_{ij} = -\rho\overline{u_i'u_j'} = \mu_t\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) - \frac{2}{3}\rho k\delta_{ij} \tag{2.14}$$

The transport equation for k is,

$$\frac{\partial(\rho k)}{\partial t} + \frac{\partial}{\partial x_i}(\rho k \bar{u}_i) = \frac{\partial}{\partial x_i}\left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_i}\right] + 2\mu_t S_{ij} S_{ij} - \frac{2}{3}\rho k \frac{\partial \bar{u}_i}{\partial x_j}\delta_{ij} - \beta^* \rho k \omega \tag{2.15}$$

where

$$S_{ij} = \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}.$$

The ω equation is,

$$\begin{aligned}\frac{\partial(\rho\omega)}{\partial t} + \frac{\partial}{\partial x_i}(\rho\omega\bar{u}_i) =& \frac{\partial}{\partial x_i}\left[\left(\mu + \frac{\mu_t}{\sigma_{\omega,1}}\right)\frac{\partial \omega}{\partial x_i}\right] + \gamma_2\left(2\rho S_{ij}S_{ij} - \frac{2}{3}\rho\omega\frac{\partial \bar{u}_i}{\partial x_j}\delta_{ij}\right) + \\ &- \beta_2\rho\omega^2 + 2\frac{\rho}{\sigma_{\omega,2}\omega}\frac{\partial k}{\partial x_k}\frac{\partial \omega}{\partial x_k}.\end{aligned} \tag{2.16}$$

The model constants are $\sigma_k = 1.0$, $\sigma_{\omega,1} = 2.0$, $\sigma_{\omega,2} = 1.17$, $\gamma_2 = 0.44$, $\beta_2 = 0.083$ and $\beta^* = 0.09$ [6]. Blending functions are implemented to assure a smooth transition between the $k-\omega$ model and the $k-\epsilon$ model is done, see more in [8].

2.2.3 Artificial compressibility

Instead of implementing a pure incompressible solver a compressible solver can be modified with an artificial compressibility to satisfy the incompressible conditions. The compressible equations are hyperbolic and the incompressible have a mixed parabolic/elliptic character. This difference stems from the lack of time derivative in the continuity equation for the incompressible flow. To amend for this difference in behavior, a time derivative for pressure ($\frac{\partial p}{\partial t}$) is added to the continuity equation.

$$\frac{1}{\beta}\frac{\partial p}{\partial t} + \frac{\partial \rho u_i}{\partial x_i} = 0 \tag{2.17}$$

For steady flows the solution is run until convergence, and thus satisfies the continuity equation. The parameter β is here the artificial compressibility, the larger the value of β the more incompressible the equations are. The value of β determines the performance of the method, usually it is chosen between 0.1 and 10 [9].

The artificial compressibility method has also been applied to unsteady flows. Either the equations are solved as in the steady case with a sufficiently large β such that the contribution of the added pressure time derivative is negligible, or a dual time τ is introduced to the equation, as seen in (2.18).

$$\begin{gathered}\frac{1}{\beta}\frac{\partial p}{\partial \tau} + \frac{\partial \rho u_i}{\partial x_i} = 0 \\ \rho\frac{\partial u_i}{\partial \tau} + \rho\frac{\partial u_i}{\partial t} + \rho\frac{\partial (u_i u_j)}{\partial x_j} + \frac{\partial p}{\partial x_i} - \mu\frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) = 0\end{gathered} \tag{2.18}$$

The dual time method is computationally more demanding than the single time method, as an iterative process is run for every time step t^n in the dual time τ until steady state is achieved in τ. In return it gives the advantage that the continuity equation is fulfilled at every time step.

From numerical simulations it has also been verified that reasonably good results can be achieved with this method of simulation, especially for the dual time approach [10].

2.3 SU^2 modeled equations

The governing equations (2.1), (2.2) and (2.3) are coupled. Thus a system of equations is built as to correlate the variables naturally. As this system of equations is linearized, the corresponding linear system will be of a block-banded structure [9].
In the open source CFD software SU^2, the equations are modeled as coupled. All equations solved are of the form,

$$\frac{\partial U}{\partial t} + \nabla \cdot \vec{F}^c - \nabla \cdot \vec{F}^\nu = Q. \tag{2.19}$$

U is a vector containing the state variables, $\vec{F}^c(U)$ represents the convective fluxes, $\vec{F}^\nu(U)$ are the viscous fluxes and $Q(U)$ is the source term.

2.3.1 Compressible solver

For compressible flow the state vector is defined as $U = (\rho,\ \rho v_1,\ \rho v_2,\ \rho v_3,\ \rho E\ )^T$. The convective and viscous fluxes are given as,

$$\vec{F}_i^c = \begin{pmatrix} \rho v_i \\ \rho v_i v_1 + P\delta_{i1} \\ \rho v_i v_2 + P\delta_{i2} \\ \rho v_i v_3 + P\delta_{i3} \\ \rho v_i H \end{pmatrix},\ \vec{F}_i^\nu = \begin{pmatrix} \cdot \\ \tau_{i1} \\ \tau_{i2} \\ \tau_{i3} \\ \rho v_j \tau_{ij} + \mu_{tot}^* C_p \partial_i T \end{pmatrix},\ Q = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \rho g \\ \cdot \end{pmatrix},\ i = 1,2,3$$

Here P is the static pressure, H is the fluid enthalpy $H = E + p/\rho$ and the viscous stresses are given as $\tau_{ij} = \mu_{tot}\left(\partial_j v_i + \partial_i v_j - \frac{2}{3}\delta_{ij}\partial_k v_k\right)$. The source term Q can contain a gravity term, with $g = 9.81$ if the user chooses so. The gravity term can at this point (SU^2 version 2.0.8) only be chosen in the z-direction in 3D-calculations or y-direction for 2D-calculations. The ideal gas law is used to relate temperature, pressure and density $T = P/(R\rho)$. Also from the ideal gas law the specific heat is given as $C_p = \gamma R/(\gamma - 1)$ To take into accordance the temperature dependence of the dynamic viscosity μ_{dyn}, the Sutherland's law is applied,

$$\mu_{dyn} = A_s \frac{T^{3/2}}{T + C}. \tag{2.20}$$

In the source code of SU^2, the following constants are chosen $A_s = 1.46 \cdot 10^{-6}$ and $C = 110.3$ [3]. The total viscosity μ_{tot} is calculated by summing the contribution of the dynamic viscosity and the turbulent viscosity, thus $\mu_{tot} = \mu_{dyn} + \mu_{turb}$. Furthermore for the heat transfer the viscosity is calculated as $\mu^*_{tot} = \mu_{dyn}/Pr_{dyn} + \mu_{turb}/Pr_{turb}$. Here Pr_{dyn} is the laminar Prandtl number and Pr_{turb} is the turbulent Prandtl number. For air the values are approximately $Pr_{dyn} = 0.9$ and $Pr_{turb} = 0.72$ [3].

The turbulent viscosity is calculated by a chosen turbulence model. At the moment available turbulence models, in SU^2 version 2.0.8, are the Spalart-Allmaras (SA) and the $k - \omega$ SST models. The turbulence model is solved seperately from the governing equations, eventhough these are coupled.

2.3.2 Incompressible solver

The incompressible solver is based on the artificial compressibility formulation by Chorin, which is only valid for steady-state calculations. The state variable is set to $U = (P,\rho v_1, \rho v_2, \rho v_3)^T$, P is the pressure and v_i the velocity in a Cartesian coordinate system. The convective and viscous fluxes are given by,

$$\vec{F}^c_i = \begin{pmatrix} \beta^2 v_i \\ \rho v_i v_1 + P\delta_{i1} \\ \rho v_i v_2 + P\delta_{i2} \\ \rho v_i v_3 + P\delta_{i3} \end{pmatrix}, \ \vec{F}^\nu_i = \begin{pmatrix} \cdot \\ \mu_{tot}\partial_i v_1 \\ \mu_{tot}\partial_i v_2 \\ \mu_{tot}\partial_i v_3 \end{pmatrix}, \ Q = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ -\frac{\rho}{Fr^2} \end{pmatrix}, i = 1,2,3.$$

The artificial compressibility parameter is here denoted by β^2, also the Froude number Fr is included in the equations to provide a source term for gravity. The gravity source term can be chosen to be included or not in the same way as in the compressible case [3].

2.4 OpenFOAM modeled equations

In OpenFOAM the governing equations are not solved as a coupled system of equations as in SU^2 but rather the continuity, momentum and energy equations are all solved seperately. A short description of what equations and algorithms the solvers in this paper use is provided in this section.

2.4.1 simpleFoam

This solver is a steady-state incompressible solver. Due to the incompressibility, the energy equation does not need to be solved, as there is no link between the momentum and continuity equations. Instead the momentum equation is solved and a pressure correction is performed, thus a physical pressure is not solved but rather pressure differences are found [6].

If the divergence is taken of the momentum equation and the continuity equation is used to expand the term $\frac{\partial}{\partial x_k}(\frac{\partial}{\partial t}\rho u_k)$, a Poisson equation for the pressure is found.

$$\frac{\partial}{\partial x_i}\left(\frac{\partial p}{\partial x_i}\right) = -\frac{\partial}{\partial x_i}\left[\frac{\partial}{\partial x_j}\left(\rho u_i u_j - \tau_{ij}\right)\right] + \frac{\partial(\rho g_i)}{\partial x_i} + \frac{\partial^2 \rho}{\partial t^2} \tag{2.21}$$

In the case of constant density and viscosity the equation reduces to,

$$\frac{\partial}{\partial x_i}\left(\frac{\partial p}{\partial x_i}\right) = -\frac{\partial}{\partial x_i}\left[\frac{\partial}{\partial x_j}(\rho u_i u_j)\right]. \tag{2.22}$$

The SIMPLE algorithm, which the simpleFoam solver is based upon, is solving the momentum equation (2.7) and the Poisson pressure equation (2.22). As OpenFOAM utilizes a collocated grid Rhie-Chow interpolation is used for the pressure-velocity coupling. The SIMPLE algorithm can be found in [9].

2.4.2 pimpleFoam

The solver is a transient solver for incompressible fluids and is a merged solver consisting of the PISO and SIMPLE algorithms. The solver is using SIMPLE algorithm in an inner loop with an outer PISO loop. The SIMPLE algorithm neglects the velocity corrections as they are unknown, which results in slow convergence. In the PISO loop these are taken into consideration, and thus faster convergence [9].

2.4.3 rhoCentralFoam

This solver is solving each of the governing compressible equations seperately. First the continuity equation is solved, providing a new value for ρ. Thereafter the momentum equation is solved in two steps where first the inviscid part is calculated and the values updated and afterwards the viscid part is added. The energy equation is first solved without the diffusive flux of heat, which is later added when the updated temperature is calculated.

The continuity equation,

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_i}(u_i \rho) = 0$$

is solved with the previous time step velocity values.

For convention purposes $\widehat{u_i} = \rho u_i$ and $\widehat{E} = \rho E$. After the continuity equation is solved the inviscid momentum equation is solved,

$$\left(\frac{\partial \widehat{u_i}}{\partial t}\right)_I + \frac{\partial}{\partial x_j}(u_i \widehat{u_j}) + \frac{\partial p}{\partial x_i} = 0 \tag{2.23}$$

Equation (2.23) explicitly calculates $\widehat{u_i}$ and thus no linear solver is needed. The time derivative $(\partial/\partial t)_I$ is only the inviscid contributions. The new velocity u_i is updated by using the updated density ρ, thus $u_i = \widehat{u_i}/\rho$. The diffusion correction equation is now solved (2.24) for u_i, where the viscous terms are added.

$$\left(\frac{\partial(\rho u_i)}{\partial t}\right)_V - \frac{\partial}{\partial x_j}\left(\mu\frac{\partial u_i}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\mu\left(\frac{\partial u_j^{exp}}{\partial x_i} - \frac{2}{3}\frac{\partial u_k^{exp}}{\partial x_k}\delta_{ij}\right) = 0 \tag{2.24}$$

In the equation the time derivative $(\partial/\partial t)_V$ is the contribution of diffusion and viscous forces. The velocities u_i^{exp} are taken from the solution of the inviscid equation (2.23).

The laplacian term is added implicitly in u_i and solved for with a linear solver available in OpenFOAM.

A similar procedure is done for the energy equation. First $\widehat{E}$ at the new time step is found explicitly from the equation,

$$\left(\frac{\partial \widehat{E}}{\partial t}\right)_I + \frac{\partial}{\partial x_k}\left[u_k\left(\widehat{E}+p\right)\right] - \frac{\partial}{\partial x_i}\mu u_j\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\right) = 0. \tag{2.25}$$

From $\widehat{E}$ the temperature T is calculated through,

$$T = \frac{1}{C_v}\left(\frac{\widehat{E}}{\rho} - \frac{u_k u_k}{2}\right). \tag{2.26}$$

Then a diffusion correction equation for T is solved to include the diffusive terms,

$$\left(\frac{\partial(\rho C_v T)}{\partial t}\right)_V - \frac{\partial}{\partial x_k}\left(k\frac{\partial T}{\partial x_k}\right) = 0 \tag{2.27}$$

The diffusion correction equation (2.27) for T is carried out implicitly, thus the laplacian of T is taken at the new time step. An iterative solver of choice is used to solve this system of equations. After the temperature is calculated the temperature dependent quantities k and μ are evaluated at the new temperature T. Also the pressure $p = \rho RT$ is updated. The variables k, μ and p are held constant through each iteration and only updated at the end of it [11].

2.5 Gerris modeled equations

For the time being Gerris does not include steady state, compressible or turbulent capabilities. Thus incompressible DNS calculations is what is available at the moment. To counter this a powerful adaptive mesh refinement is provided in Gerris in combination with an octree-mesh structure. Calculations with Gerris are done through a projection method. This method utilizes a pressure correction with the equation (2.22) [4]. As Gerris utilizes a collocated grid an approximate projection method is used to calculate the divergence free velocity field. The intermediate velocity field is calculated at the cell faces by interpolation and the created staggered (MAC) field is then projected. From this the cell centered pressure gradients are constructed and thus an approximate divergence free velocity can be calculated at the new time step [12].

For solving the discretized equations a Godunov solver is implemented in Gerris for the convective terms.

2.6 Finite volume method

The finite volume method utilizes an integration over a control volume. The control volume is constructed from a given mesh. Depending on whether the mesh is unstructured or structured the control volumes for every cell node is created differently. The Gaussian theorem (2.28) is used to simplify terms in the governing equations,

$$\int_{CV} \frac{\partial \phi_i}{\partial x_i} dV = \int_A n_i \phi_i dA. \tag{2.28}$$

where CV is the control volume, A is the area of the control volume and n_i is the normal vector to the surface area integrated.

2.6.1 CFL number

The Courant-Friedrich-Lewy condition is a numerical constraint which determines the allowed time step for a specific grid size. This constraint determines that information can only propagate no further than one cell away from the original cell. In explicit schemes this constraint is necessary for convergence. If information propagates with the speed $\tilde{u}$, then the CFL number is given in equation (2.29) for a one dimensional case.

$$\tilde{u}\frac{\Delta t}{\Delta x} < C \tag{2.29}$$

where C is a number which determines the CFL condition. For explicit schemes $C < 1$ is required, but it can be larger for implicit schemes [13].

2.6.2 y^+

y^+ is a non-dimensional wall distance used to determine the boundary layer at which a control volume is located. It is calculated from (2.30).

$$y^+ = \frac{u_\tau y}{\nu} \tag{2.30}$$

Where $u_\tau = \sqrt{\tau_w/\rho}$ is the friction velocity. For $y^+ < 5$ the cell resides in the viscous sublayer, where $u/u_\tau = y^+$ and for $y^+ > 30$ the flow is in the log-law region [8].

2.6.3 Wall functions

Wall functions are used to lessen the amount of cells needed to resolve a domain, as the viscous sublayer is not fully resolved close to walls. Instead the first computational node is put at around $30 < y^+ < 100$ in the log-law region. Problems using this approach may arise in separation and attachment regions [9].

2.7 Discretization of Modeled Equations

To solve the modeled equations, a discretization has to be made of the modeled equations. Different discretization schemes are used for different components of the modeled equations.

2.7.1 Time discretization

The time discretization determines the way the algorithm updates the solution in time. The term $\partial/\partial t$ can be discretized a various ways. Let (2.31) be a partial differential equation.

$$\frac{\partial \phi}{\partial t} = f(\phi, \psi) \tag{2.31}$$

The explicit Euler, implicit Euler and Crank-Nicholson methods can be explained with the following discretization,

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = f(\theta\phi^{n+1} + (1-\theta)\phi^n, \psi^n). \tag{2.32}$$

The notation ϕ^{n+1} indicates the variable ϕ at the time step $n+1$ and $\Delta t = t^{n+1} - t^n$. The variable ψ is set as explicit as it is not solved for in the equation. If $\theta = 0$ the discretization is called explicit Euler and is first order accurate. For $\theta = 1$ the discretization is called implicit Euler and it is also first order accurate. For $\theta = 1/2$ it is the Crank-Nicholson method, which is second order accurate.

More advanced schemes like Runge-Kutta are also available for time discretization, these schemes can reach high order of accuracy, but are more computationally expensive. The time discretization term in the finite volume method is taken as piecewise constants over the control volumes. Thus, [6]

$$\int_{CV} \frac{\partial \phi}{\partial t} dV = V_{CV} \left(\frac{\partial \phi}{\partial t}\right)_P . \tag{2.33}$$

2.7.2 Convective discretization

If the modeled equations contain divergence terms $\partial/\partial x_j(\rho u_j \phi_i)$, the finite volume method requires the evaluation of ϕ_i at the control volume faces in accordance with the Gaussian theorem (2.28),

$$\int_{CV} \frac{\partial \rho u_j \phi_i}{\partial x_j} dV = \sum_k \int_{S_k} n_j \rho u_j \phi_i dS_k. \tag{2.34}$$

The integer k is the number of faces of the control volume. For a Cartesian grid in three dimensions, a control volume is a hexahedron and has six faces. As the value of ϕ is not known at the faces, an interpolation has to be made between the cell nodes.

2.7.2.1 Centered schemes

The convective terms can be discretized using one of the common interpolation schemes such as the central difference scheme. This approach is unlikely to be used in CFD application as the interpolation schemes are not bounded and do not fulfill the transportivness requirement [6].

2.7.2.2 Upwind schemes

A simple solution is to use an upwind scheme to find the values at the faces. The upwind scheme is enforced as the following for ϕ at a face between two cell nodes P and A,

$$\phi_{PA} = \phi_P \quad \text{if } F_k > 0 \tag{2.35}$$

$$\phi_{PA} = \phi_A \quad \text{if } F_k < 0. \tag{2.36}$$

The flux F_k is defined as $F_k = n_i \rho u_i \Delta S_k$, where ΔS_k is the surface of the k control volume face. This method is easily implemented, conservative, bounded and first order accurate. The issue with this scheme lies in the fact that when the flow is not aligned with the grid a false diffusion error is introduced to the solution. The linear upwind scheme (LUDS) and the quadratic upwind scheme (QUICK) use the same principle as the upwind scheme but uses interpolation between nodes to increase the accuracy. The linear upwind scheme is second order accurate and the QUICK scheme is third order accurate. The drawback of the QUICK scheme is that it suffers from oscillatory behavior for high Peclet number flows.

2.7.2.3 TVD schemes

The TVD (Total Variation Diminishing) schemes were developed to counter the oscillatory behavior of higher order upwind schemes while still achieving a high order of accuracy. This is done by introducing a monotonicity criterion. This criterion states that a scheme must not create local extrema and that the value of an existing local minimum/maximum be non-decreasing respectively non-increasing [6].

In a structured grid, a TVD scheme is constructed from a flux limiter function $\Psi(r)$ and a nonlinear function r. The non-linear function r is defined as, if the velocity at the face $v_{i+1/2} > 0$,

$$r = \frac{\phi_i - \phi_{i-1}}{\phi_{i+1} - \phi_i}. \tag{2.37}$$

The index i determines what node points are used, for instance if the flux is calculated at a face in the x direction, $i+1$ would correspond to the eastward node and $i-1$ to the westward node. Flux limited schemes are described from the following equation (still assuming $v_{i+1/2} > 0$),

$$\phi_{i+1/2} = \phi_i + \frac{1}{2}\Psi(r)\left(\phi_{i+1} - \phi_i\right). \tag{2.38}$$

A TVD scheme is created by imposing requirements on the flux limiter function $\Psi(r)$. The upwind schemes in section 2.7.2.2 can be written in this format. The upwind scheme is attained from $\Psi(r) = 0$, LUDS from $\Psi(r) = r$ and QUICK from $\Psi(r) = (3 + r)/4$. However, only the upwind scheme is TVD.

Necessary and sufficient conditions for a scheme to be TVD are,

$$\begin{cases} \Psi(r) \leq & 2r \quad 0 < r < 1 \\ \Psi(r) \leq & 2 \quad\quad r \geq 1. \end{cases} \tag{2.39}$$

A few flux limiters for TVD schemes are,

$$\begin{array}{rl} \text{Van Leer} & \dfrac{r+|r|}{1+r} \\ \text{MUSCL} & \dfrac{r+|r|}{1+|r|} \\ \text{SUPERBEE} & \max(0,\min(1,2r),\min(2,r)). \end{array}$$

For unstructured grids equation (2.37), does not always hold. A modification is proposed in (2.40). Let the indexes D denote a neighboring and C denote the current cells node.

$$\begin{aligned} r &= \left[\frac{(2\nabla\phi_C\cdot\vec{r}_{CD})}{\phi_D-\phi_C}-1\right] \\ \phi(r) &= \phi_C+\frac{1}{2}\Psi(r)(\phi_D-\phi_C) \end{aligned} \tag{2.40}$$

The vector $\vec{r}_{CD}$ is the vector between the nodes C and D [14].

Another way to impose the monotonicity criterion is to impose the condition that the values of the reconstructed polynomial in the control volume is not to exceed the maximum and minimum values at the neighbors. This condition was slightly modified to,

$$\min(\phi_N,\phi_P)\leq(\phi_P+\Psi_j\nabla\phi_P\cdot\vec{r}_{jP})\leq\max(\phi_N,\phi_P)\ \forall N\in\text{Neighbors(P)} \tag{2.41}$$

where Ψ is the flux limiter, r_{jP} is the distance between the points j and P (cell center) and j is an arbitrary point in the control volume of P. The reconstruction polynomial $(R_P(r_{jP})=(\phi_P+\Psi_j\nabla\phi_P\cdot\vec{r}_{jP}))$ is evaluated at each cell face. The flux limiter function reduces to,

$$\Psi_f=\begin{cases} \Phi\left(\frac{\max(\phi_N,\phi_P)-\phi_P}{\nabla\phi_P\cdot\vec{r}_{fP}}\right) & \phi_f>\phi_P \\ \Phi\left(\frac{\phi_P-\min(\phi_N,\phi_P)}{\nabla\phi_P\cdot\vec{r}_{fP}}\right) & \phi_f<\phi_P \\ 1 & \phi_f=\phi_P \end{cases} \tag{2.42}$$

where $\Phi(x)=\min(x,1)$. For the flux limiter the minimal value over all cell faces are chosen, $\Psi_P=\min(\Psi_f)$.

Venkatakrishnan [15] improved this approach by changing the function Φ to,

$$\Phi(x)=\frac{x^2+2x+\varepsilon^2}{x^2+x+2+\varepsilon^2} \tag{2.43}$$

where $\varepsilon=(Kh)^3$. K is a user-specified constant and h a local mesh size [14].

2.7.2.4 Compressible flow

For a compressible flow fluid properties are not only transported by the flow, but also by waves. This can give arise to shocks which are discontinuous. Different types of discretization are needed for these flows. The methods can be divided into either central schemes such as the Lax-Friedrich method and the Kurganov-Tadmor scheme [16] (used in *rhoCentralFoam*) or approximate Riemann solvers such as Godunov's and Roe's method [13].

2.7.3 Gradient discretization

From the modeled equations terms of the form $\partial/\partial x_i(\phi)$ appear. These terms are discretized in the finite volume method as,

$$\int_{CV} \partial/\partial x_i \phi dV = \sum_k \int_{S_k} \phi n_i dS_k \tag{2.44}$$

The values of ϕ needs to be calculated at the surfaces of the control volumes. This is done by an interpolation scheme for the value of ϕ at the cell centers.

Alternatively a least squares method can be used to calculate the gradient. Instead of calculating the values of ϕ at the cell faces the gradient $\partial/\partial x_i(\phi)$ is calculated at the cell point P. This is done by a least squares method where the gradients between the cell and its neighbors are calculated [9].

2.7.4 Viscous discretization

The modeled equations contain terms of the form $\partial/\partial x_i(\Gamma\partial\phi_k/\partial x_k)$. When these terms are integrated over a control volume the following is found,

$$\int_{CV} \frac{\partial}{\partial x_k}\left(\Gamma\frac{\partial}{\partial x_k}\phi\right) dV = \sum_k \int_{S_k} n_j \Gamma \frac{\partial \phi}{\partial x_j} dS_k \tag{2.45}$$

This requires the calculation of gradients at the surfaces of the control volumes. A way to do this is to calculate the gradients of the cell centers and then use interpolation to find the value on cell faces. To find the value of the gradient at the cell center P, it is approximated as the average value over the cell as proposed in equation (2.46).

$$\left(\frac{\partial \phi}{\partial x_i}\right)_P \approx \frac{\int_\Omega \frac{\partial \phi}{\partial x_i} d\Omega}{\Delta\Omega} \tag{2.46}$$

Through the finite volume approximation this is simplified to,

$$\left(\frac{\partial \phi}{\partial x_i}\right)_P \approx \frac{\sum_k \phi_k \Delta S_k n_i^{(k)}}{\Delta\Omega} \tag{2.47}$$

where $n_i^{(k)}$ is the outward normal of the surface k in the control volume. The gradients can then be interpolated to the faces after they have been calculated. The gradient at the cell center can be derived using a least squares method as well [9].

2.8 Linear Solvers

After discretization of the Navier-Stokes equations linear solvers are needed to solve linear equations of the form $Ax = b$. As these systems of equations are too large for direct solvers, iterative methods are used to find a solution. The most commonly used method for solving linear systems iteratively, make use of Krylov subspace methods. In the open source CFD codes OpenFoam, SU^2 and Gerris, mainly Krylov solvers are used to solve the linear systems. Though in some solvers of OpenFOAM more basic iterative methods like Jacobi and Gauss-Seidel are used. To increase performance, preconditioning and multigrid methods are also used [17].

2.8.1 Krylov subspace methods

Let x_0 be an initial guess to the problem $Ax = b$ and $r_0 = b - Ax_0$. An approximate solution x_m is found in the affine space,

$$x_m \in x_0 + \mathcal{K}_m(A,r_0). \tag{2.48}$$

where the space $\mathcal{K}$ is defined as,

$$\mathcal{K}_m(A,r_0) = span\left\{r_0,Ar_0,A^2r_0,...,A^{m-1}r_0\right\}. \tag{2.49}$$

Furthermore a Petrov-Galerkin condition is applied on x_m such that

$$b - Ax_m \perp \mathcal{L}_m, \tag{2.50}$$

where $\mathcal{L}_m$ is a subspace of dimension m. The choice of $\mathcal{L}_m$ and the preconditioning of the initial linear system defines the Krylov subspace method used. Thus the approximation of the solution x^* will always be of the form $A^{-1}b \approx x_0 + q_{m-1}(A)r_0$, where q_{m-1} is a polynomial of degree $m - 1$.

2.8.1.1 GMRES

The Generalized Minimum Residual Method (GMRES) is a Krylov subspace method with $\mathcal{K} = \mathcal{K}_m$ and $\mathcal{L} = A\mathcal{K}_m$. Using a version of the Arnoldi's method, which creates an orthonormal matrix that spans $\mathcal{K}_m$, a solution is found. This solver can handle both symmetric and asymmetric matrices.

2.8.1.2 Conjugate Gradient Method

This method also utilizes a form of the Arnoldi's method, but in the case when A is symmetric positive definite. For symmetric positive definite matrices the Anorldi's method reduces to the Lanczos method. It requires that $\mathcal{L}_m = \mathcal{K}_m$, thus making the solution an orthogonal projection onto the Krylov subspace $\mathcal{K}_m(r_0,A)$.

2.8.1.3 Biconjugate Gradient Method

This method uses an extension to Lanczos method used in the Conjugate Gradient Method. The solution is a projection onto $\mathcal{K}_m = span\left\{v_1,Av_1,...,A^{m-1}v_1\right\}$, orthogonally to $\mathcal{L}_m = span\left\{w_1,Aw_1,...,A^{m-1}w_1\right\}$. Usually $v_1 = r_0/\left\|r_0\right\|_2$, and w_1 can be chosen arbitrarily as long as $(v_1,w_1) \neq 0$ but is often chosen as v_1 [17].

2.8.1.4 BiCGSTAB

The Biconjugate Gradient Stabilized is a method where the transpose A^T is not needed to be computed explicitly. This reduces the computational time. It is an improved version of the Conjugate Gradient Squared (CGS) algorithm. For further reading [17] is recommended. The advantages of this method compared to the Biconjugate Gradient Method is a smoother and faster convergence properties [3].

2.8.2 LU-SGS

In SU^2 another solver for the linear system is implemented. The Lower-Upper Symmetric-Gauss-Seidel (LU-SGS) method, [18] which is a stationary iterative method.

2.9 Acceleration techniques

To accelerate the convergence of linear solvers a few different techniques are used in CFD codes. A brief description of some of the more common techniques are given in this section. Mainly preconditioning of the linear system and the multigrid method will be covered.

2.9.1 Multigrid method

Multigrid methods were created from the observation of iterative methods dependency on the spectral radius of the iteration matrix ($e^{\eta+1} = Me^{\eta}$). M is the iteration matrix and e^{η} is the error $e^{\eta} = u - u^{\eta}$ where u is the exact solution. The eigenvectors with the largest eigenvalues determine the rate of convergence for the method.[9] The Gauss-Seidel method is able to damp out high-frequency modes after only a few iterations, whereas for low-frequency modes a longer time is required to achieve convergence. However many of the low-frequency modes can be mapped onto a coarser grid as high-frequency modes. Thus the idea of the multigrid method was formed [17].

The Geometric multigrid, performs transforms between different coarse sized grids. Iterations with some iterative scheme are performed to smoothen the solution on each grid level, to remove the high-frequency errors. There are different ways of implementing the Multigrid method, some of the most common algorithms used are either the V-cycle ,W-cycle or Full Multigrid Method (FMG). If the multigrid method is applied without using any knowledge of the grid and only from the initial linear system $Ax = b$, the Algebraic Multigrid Method (AMG) is a viable method. It has the advantage over the Geometric Multigrid Method that it can better solve problems with anisotropic or discontinuous coefficients. For further reading on the topics of multigrid methods [17] is recommended.

2.9.2 Preconditioners

The efficiency and robustness of iterative solvers can be improved by the use of preconditioners. The reliability of of iterative techniques is often determined by what preconditioner is used, rather than what Krylov subspace method is chosen. After determining the preconditioned matrix M for the linear system $Ax = b$, the preconditioner either is applied from the left

$$M^{-1}Ax = M^{-1}b$$

or from the right

$$AM^{-1}u = b, \quad x = M^{-1}u.$$

One of the simplest ways to construct a preconditioner is to perform an incomplete factorization of the original matrix A. A decomposition of the form $A = LU - R$ where L and U have the same nonzero structure as the lower and upper parts of A respectively, and R is the residual error of the factorization. This incomplete factorization is called ILU. It is an inexpensive way to compute a preconditioner, but often requires the Krylov subspace method to perform many iterations before convergence. For a symmetric matrix the Incomplete Cholesky factorization can be used as a preconditioner instead to preserve symmetry.

Another simple preconditioner is the Jacobi preconditioner. Which is also known as a diagonal preconditioner, as the preconditioner is chosen to be the diagonal of the matrix A [17].

2.9.3 Linelet preconditioning

In SU^2 a linelet preconditioner has been implemented to improve the convergence of the Krylov subspace solvers. Lines are constructed in the mesh which lie in the direction normal to the grid stretching. By assembling the system matrix and the non-diagonal entries of the edges belonging to the linelets the preconditioned matrix is received. With appropriate numbering the preconditioned matrix will be a block tridiagonal matrix. For zones where linelets are not defined a Jacobi preconditioner is used. The preconditioned system is now block diagonal and can be inverted by using the Thomas algorithm [3].

2.9.3.1 Roe-Turkel preconditioner

The numerical discretization of the governing compressible equations often result in excess artificial viscosity at low Mach numbers. The Roe-Turkel preconditioning allows the solving of nearly incompressible problems using the same numerical methods as for compressible problems. This preconditioning is, in particular, useful in flows where parts of the flow is incompressible.

A correction of the convective fluxes discretization is performed and involves modifying the artificial viscosity, such that correct scaling of it can be achieved at low Mach numbers. This preconditioning does not affect the time accuracy of unsteady problems. Furthermore it is important to realize that this preconditioner is a modification of the Roe method of discretizing the convective fluxes [3].

2.10 Rotating motion

Different approaches can be chosen for rotating geometries. For simple geometries, such as a sphere, a velocity can be prescribed at the boundary to induce a rotating motion. For more complex geometries more advanced methods have to be used.

2.10.1 MRF

If a rotational frame of reference is chosen, a rotating object can be viewed as stationary through a transformation of the Navier-Stokes equation.

The properties of the flow between stationary and rotating zones are translated at the specified interfaces between them. For OpenFOAM the MRF (Multiple Reference Frame) capability is only able to handle steady-state simulations [5].

2.10.2 AMI

In contrary to MRF there occurs mesh motion in AMI (Arbitrary Mesh Interface). There needs to be two meshes and an interface which is geometrically identically joining the two meshes. One of the meshes is rotated whereas the other is kept stationary.

Standard consistent interpolation is unfortunately not a viable option for connecting the AMI surfaces. Interpolating a field q_D from the donor mesh (denoted T_D) to a target mesh (denoted T_T) receiving q_T suffers from some critical drawbacks, such as not being conservative, minima and maxima are eroded and have issues handling discontinuous fields. Instead if a Galerkin projection is used, the projection from the donor mesh to the target mesh is optimally accurate in the L_2 norm and from it, conservation follows. To project from the donor mesh an intermediate supermesh is created.

The Galerkin projection is defined as,

$$\|q_D - q_T\|_2 = \min_{q \in V_T} \|q_D - q\|_2 \tag{2.51}$$

here $V_T = \text{span}\left\{\phi_T^{(i)}\right\}$ and $\phi_T^{(i)}$ are the basis functions of the target mesh. Equation (2.51) can be rewritten as,

$$\int_\Omega 2\phi_T^{(i)}(q_D - q)dV = 0, \text{ for all } i \in 1,...,N_T. \tag{2.52}$$

The constant N_T is the number of basis functions in the target mesh. Noticing that $q = q_T$ it is easily verfied that the projection is conservative if the constant function

1 is contained in $\phi_T^{(i)}$ for $i \in 1,...,N_T$. If all components are written out the equations become,

$$\int_\Omega \sum_{i=1}^{N_D} q_D^{(i)} \phi_D^{(i)} \phi_T^{(k)} dV = \int_\Omega \sum_{j=1}^{N_T} q_D^{(j)} \phi_D^{(j)} \phi_T^{(k)} dV \tag{2.53}$$

or in matrix form,

$$M_T q_T = M_{TD} q_D. \tag{2.54}$$

The matrix M_{TD} can be viewed as a mixed mass matrix between the meshes T_D and T_T, and problems can arise while assembling the matrix. To assemble M_{TD} the products of T_D and T_T have to be integrated, over each element T_T the basis functions of T_D can be discontinuous. If the basis functions are evaluated at each of the quadrature points of T_T, the integral will not be exact as these methods are not specified for piecewise polynomials. To solve this issue a supermesh is introduced, where the intersection of the target and donor mesh elements define the mesh. On each of the new elements in the supermesh, the donor and target basis functions are polynomials and quadrature integration can be performed [19].

3

Software

In the observed open source CFD softwares, there are different ways of specifying a simulation. In this section a view on the different ways of initializing a simulation is made. Main focus will be put on SU^2 which was integrated on the SimScale platform. For the other solvers a brief overview will be given of the input structure.

3.1 OpenFOAM

In OpenFOAM a certain structure of the input files is expected. A case has to be set up in a predestined manner which contains a minimum of three directories. A *constant* folder and a *system* folder are needed. Also a time folder is needed, this is usually named *0*, but can be named differently if 0 is not the starting time. There are also subfolders and files that are contained in the mentioned folders, a few options are reviewed here. The structure of an OpenFOAM case can be seen in figure 3.1.

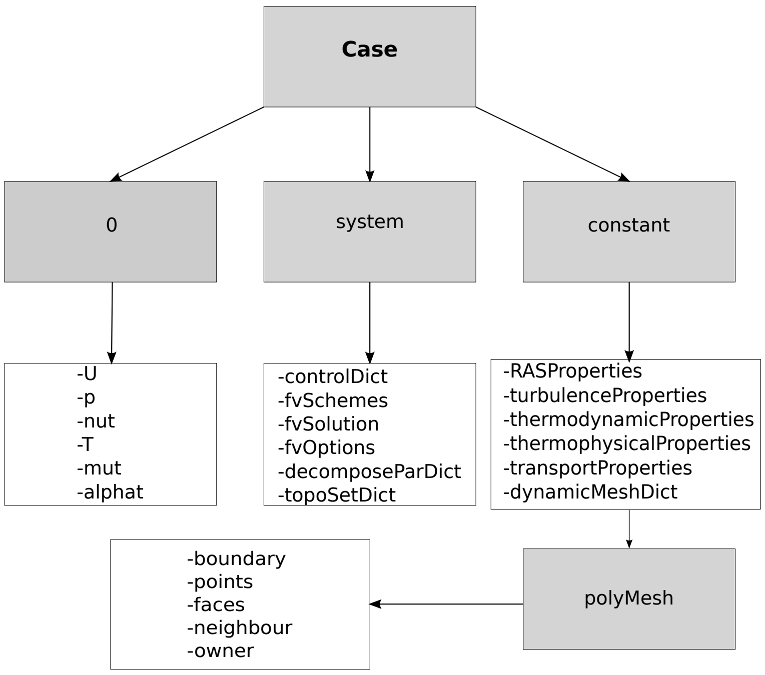


Figure 3.1: Case structure in OpenFOAM

3.1.1 0

This folder contains files with the initial conditions of the used variables. For laminar incompressible Navier-Stokes equation only files containing the initial conditions of U and p are needed. For turbulence models or compressible flow other variables will need to be added as well.

Three entries have to be done for each variable file. The dimension of the variable is assigned through *dimensions* in the file (for instance m/s for the velocity). The internal field is assigned through *internalField* and the boundary field is given through *boundaryField* [5].

3.1.2 system

This folder contains the specifications for the simulation. In the *decomposeParDict* file the mesh is decomposed into an assigned number of parts for parallel simulations. The *topoSetDict* creates sets in the mesh, which can be used to define areas with extra source terms. For the *fvOptions* file extra source terms can be assigned to sets, such as done for MRF in section 4.1.1. In *controlDict* the frequency of solution file outputs, run time, time steps and Courant number are assigned [5].

3.1.2.1 fvSchemes

In the *fvSchemes* file the numerical discretization schemes for the different components in the modeled equations are assigned. For time discretization the schemes shown in table 3.1 are available. They are assigned under *ddtSchemes* in the *fvSchemes* file.
In the CrankNicholson case a blending function parameter $\psi \in [0,1]$ can be chosen.

Euler	First order, bounded, implicit
localEuler	Local-time step, first order, bounded, implicit
CrankNicholson ψ	Second order, bounded, implicit
backward	Second order, implicit
steadyState	No solving of time derivatives

Table 3.1: Time schemes

For $\psi = 1$, the normal Crank-Nicholson scheme is used, whereas if $\psi = 0$ is chosen it corresponds to an Euler scheme [5].

Available interpolation schemes are given in table 3.2. These are assigned under *interpolationSchemes*

linear	Linear interpolation
cubicCorrection	Cubic scheme
midPoint	Linear interpolation with symmetric weighting

Table 3.2: Interpolation schemes

There are also upwind schemes that can be used for interpolation, but these are seldom used except for convective terms.

Gradients can be discretized as seen in table 3.3 and are assigned under *gradSchemes*. Interpolation schemes used for gradient calculations, are generally chosen to be a centered

Gauss <interpolationScheme>	Second order, Gaussian integration
leastSquares	Second order, least squares
fourth	Fourth order, least squares
cellLimited <gradScheme>	Cell limited version of one of the above schemes
faceLimited <gradScheme>	Face limited version of one of the above schemes

Table 3.3: Gradient schemes

scheme such as linear or cubic interpolation. Upwind based interpolation is available but seldom used for this purpose.

For discretization of divergence terms the schemes are presented in table 3.4 and assigned under *divSchemes*. Other interpolation schemes than upwind based schemes can

upwind	First order bounded
linearUpwind	First/second order linear upwind scheme, bounded
QUICK	First/second order bounded
TVD schemes	First/second order bounded
SFCD	Second order bounded
NVD schemes	First/Second order bounded

Table 3.4: Divergence schemes

be used. For instance the centered schemes of table 3.2 are available for discretization, but seldom used.

For laplacian terms an interpolation scheme is required for interpolation of diffusion coefficients and also a surface normal gradient scheme for the surface gradient. As the possible interpolation schemes already have been mentioned in table 3.2 only the surface normal schemes are presented in table 3.5. The laplacian terms are assigned under *laplacianSchemes*. The limited correction has to be assigned a value to $\phi \in [0,1]$, which determines how much non-orthogonal correction is made.

corrected	Explicit non-orthogonal correction
uncorrected	No non-orthogonal correction
limited ϕ	Limited non-orthogonal correction
bounded	Bounded correction
fourth	Fourth order

Table 3.5: Surface normal gradients for laplacian terms calculations

3.1.2.2 fvSolution

In this file the linear solvers for the sequential equations are assigned. Also the linear solver tolerance and maximum number of iterations are assigned here. In OpenFOAM the available linear solvers are preconditioned conjugate or bi-conjugate gradient solvers, smooth solvers, generalized geometric-algebraic multi-grid (GAMG) solvers and also diagonal solvers (for explicit systems).

The conjugate gradient solvers can use the preconditioners diagonal-incomplete LU (DILU for asymmetric systems), diagonal-incomplete Cholesky (DIC for symmetric systems), diagonal preconditioning and GAMG preconditioning.

Available smoothers are Gauss-Seidel and Diagonal incomplete-Cholesky (for symmetric matrices).

Under relaxation of the sequential equations can be set in this file, and if SIMPLE or PISO algorithms are used settings for these can also be specified.

3.1.3 constant

This folder contains specifications for turbulence and fluid properties. Depending on the solver chosen, different files need to be specified. For all solvers which calculate the RANS equations, the file *RASProperties* determines the turbulence model used. The type of turbulence model applied is determined in *turbulenceProperties* where either LES, RAS or laminar model can be chosen. For incompressible solvers the file *transport-Properties* determines the behavior of the kinematic viscosity ν. For the compressible solver *rhoCentralFoam* temperature dependency is determined in the files *thermodynam-icProperties* and *thermophysicalProperties*.

For dynamic mesh operations, the file *dynamicMeshDict* prescribes the conditions. To use dynamic mesh operations specific solvers have to be used such as pimpleDyM-Foam.

3.1.3.1 polyMesh

OpenFOAM uses a cell-centered control volume for its calculations. In the *polyMesh* folder files are contained describing the mesh. These files include *points*, which contain the points of the mesh. *faces*, which contain the faces of the cells. *owner*, that contains what faces belong to a cell and *neighbour* which contains the information about the connectivity between cells. Also the boundaries are given in the file *boundary*, here the boundaries are assigned names and also of what type they are, such as empty, wall or patch for instance [5].

3.2 SU^2

3.2.1 Mesh file

SU^2 uses its own mesh format *.su2*. This format is similar to the VTK format, and uses the same notation for cells. It is also utilizes vertex-based control volumes. As SU^2 has the capability to either have a two dimensional or three dimensional mesh as input the first thing specified in this mesh is **NDIME**, which can be set to either 2 or 3, depending on the dimension of the specified problem.

To demonstrate this a unit square with two triangular elements is created with the SU^2 native format, as seen in figure 3.2.

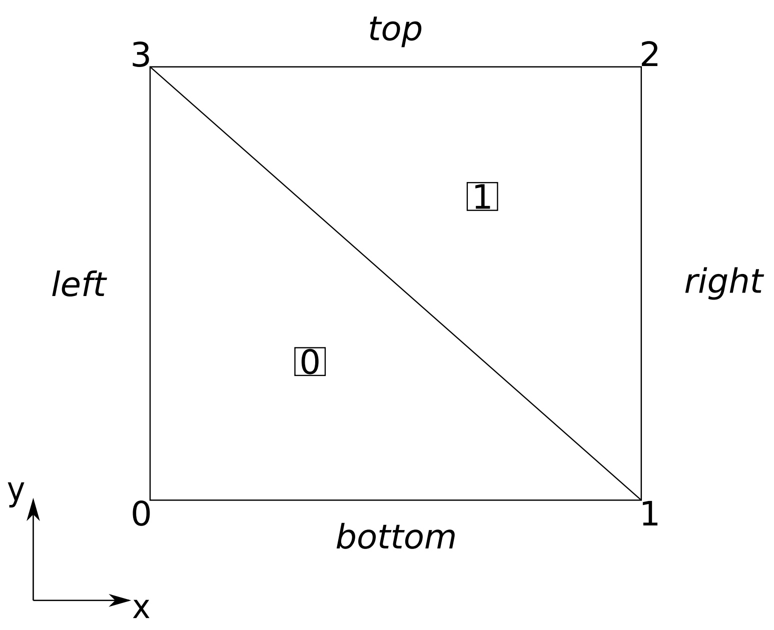


Figure 3.2: Example mesh of SU^2

Thus first line of the input file is,

```
%Problem dimension
NDIME= 2
```

Next the number of points and their coordinates are specified. First the total number of points are specified in **NPOIN**, then the point coordinates are assigned in the following fashion,

$$\begin{matrix} & \dots & \dots & \\ x_i & y_i & z_i & i \\ & \dots & \dots & \end{matrix}$$

The index i indexes the points, the indexing starts from 0 and goes to **NPOIN** -1. If the simulation is two dimensional the z_i coordinate is left out. For our unit square this would be the input.

```
%Number of points
NPOIN= 4
0.0 0.0 0
1.0 0.0 1
1.0 1.0 2
0.0 1.0 3
```

To create the mesh, the points have to be connected together in elements. This is done by following the VTK format, the following elements and their identifiers are shown here,

Element Type	Identifier
Line	3
Triangle	5
Rectangle	9
Tetrahedral	10
Hexahedral	12
Wedge	13
Pyramid	14

In the example mesh, the square is divided into two triangles. Thus two elements are present in this mesh, which is assigned to the variable **NELEM**. This is demonstrated in the example mesh by,

```
%Elements
NELEM= 2
5 0 1 3
5 1 2 3
```

As the mesh also consists of boundaries, these have to be defined as well. The number of boundaries in the mesh are prescribed to **NMARK**. The boundaries are defined by tag names in **MARKER_TAG** and the number of elements in a boundary are prescribed by **MARKER_ELEM**. In the example case the squares boundaries are named left, right, top and bottom.

```
%Boundaries
NMARK= 4
MARKER_TAG= top
MARKER_ELEM= 1
3 2 3
MARKER_TAG= bottom
MARKER_ELEM= 1
3 0 1
MARKER_TAG= left
MARKER_ELEM= 1
3 0 3
MARKER_TAG= right
MARKER_ELEM= 1
3 1 2
```

3.2.2 Configuration file

To assign solvers, specify boundary conditions and other conditions necessary for a simulation, the correct options have to be set in SU^2's configuration file *.cfg*. In SU^2 most specifications are already assigned by default. In the configuration file these default values can be changed. As the integration of SU^2 on the SimScale platform is intended for Compressible Navier-Stokes equation only the settings which cover this subject will be touched upon here.

3.2.2.1 Time discretization

Discretization in time is either done implicitly or explicitly. In SU^2 time is discretized in one of the following ways [20].

Euler	First order, explicit
Euler implicit	First order, bounded, implicit
Runge-Kutta	Up to 4th order, either explicit or implicit

To set the time discretization the command **TIME_DISCRE_FLOW** is changed in the configuration file, by default a Runge-Kutta scheme is applied for the flow. Also relevant for the time stepping is the Courant-Friedrichs-Lewy (CFL) number, this time stepping constraint is easily changed in SU^2, when changing the command **CFL_NUMBER**. For steady simulations the CFL number is set to 1.2 by default. Furthermore for steady simulations local time stepping is implemented and regulated by the CFL number.

3.2.2.2 Viscous terms discretization

The gradients of the flow variables can be calculated using a Green-Gauss method or a least squares method, at the cell nodes. By default a Green-Gauss method is chosen. The variable for determining the way gradients are calculated is **NUM_METHOD_GRAD** [20].

The gradients are then calculated to the cell faces by either averaging or corrected averaging of the gradients at the nodes. The gradient at the cell faces can also be calculated by a Galerkin method. The way this is done is determined by the command **VISC_NUM_METHOD_FLOW** in the configuration file.

3.2.2.3 Convective terms discretization

For the convective terms a lot of options are available in SU^2. The command **CONV_NUM_METHOD_FLOW** determines the way the convective terms are discretized. The available schemes for the convective terms are,

Roe-1st order	First order in space, upwind scheme
Roe-2nd order	Second order in space, MUSCL scheme with slope limiter
Lax-Friedrich	First order in space, centered scheme
Jameson-Schmidt-Turkel (JST)	First order in space, centered scheme
HLLC 1st and 2nd order	First/Second order in space, approximate Riemann solver with slope limiter
AUSM 1st and 2nd order	First/Second order in space, approximate Riemann solver with slope limiter

Available slope limiters for for the schemes are Venkatakrishnan, Bartha and Minmod limiters.

3.2.2.4 Linear solvers

As the system of equations are not symmetric in SU^2, a non-symmetric linear solver has to be chosen. The available linear solvers in SU^2 for flow calculations are LU-SGS, BiCGSTAB and GMRES. Also Newton, quasi-Newton and steepest decent methods are available but not recommended for flow calculations. For acceleration techniques multigrid is available, and preconditioners like Jacobi or Linelet preconditioners. By default LU-SGS is chosen as the linear solver and Jacobi as the preconditioner [20].

3.2.2.5 Convergence criterion

Either a residual reduction criterion is enforced, where the maximum residual in the domain is reduced by a set order of magnitude. Or a Cauchy convergence criterion can be enforced, where a chosen calculated value (such as C_D or C_L) is summed up over an assigned number of time steps and its change evaluated against the assigned convergence tolerance [20].

3.3 Gerris

This software uses an octree-grid mesh for its calculations. As Gerris produces its own mesh either an analytical expression of a geometry or a geometry file needs to be provided. The provided geometry file needs to be of the type *.gts*. There are tools such as *stl2gts* which can convert standard geometry files into this format. The input file is denoted as a *.gfs* file.

First the type of simulation is determined in the input file. For incompressible Navier-Stokes two possible simulation types are available. For non moving meshes the *GfsSimulation* is used and for moving meshes the *GfsMovingSimulation* is used. As these simulation types by default are solving the incompressible Euler equations (inviscid flow), a kinematic viscosity has to be added by using *GfsSourceViscosity*. If need be, the viscosity can be dependent on a scalar tracer quantity (such as temperature).

The calculated domain is bounded within boxes of size unity, which can be scaled but have to remain cubical in shape. The syntax for initializing the computational domain for *GfsSimulation* or *GfsMovingSimulation* is to prescribe the number of *GfsBox* the domain consists of and the number of *GfsEdge* which connect the boxes. Boundary conditions are applied on the faces of the boxes or on surfaces specified analytically through *GfsSurfaceBc* [4].

In appendix A.1.3 a rotating cylinder case can be viewed as an example.

4

Results

4.1 Rotating Taylor-Couette

Two dimensional Taylor-Couette flow is chosen as a validating flow as there exists an analytical solution for this flow. For a short derivation of the analytical flow see appendix A.1. For the validation cases the inner radius of the two cylinders was chosen to $r_1 = 0.35$ [m] and the outer cylinder was chosen to $r_2 = 1.0$ [m]. The angular velocity of the inner cylinder was chosen to $\omega = 0.001$ [rad/s], which translates to a velocity of $u_\theta = 0.00035$ [m/s]. The angular velocity of the outer cylinder is set to $\omega = 0$ [rad/s]. These values are purposely set low as to make certain that the flow stays laminar.

This validation is carried out to find the work flow needed for integration to the SimScale platform and also validating the effectiveness of different rotational methods. Only methods that are applicable for arbitrary rotating bodies are used. Three different ways of calculating this simple case are proposed. Two ways in OpenFOAM are provided, one using simpleFoam with MRF and one using pimpleDyMFoam with AMI. This case was also done with Gerris to provide a comparison with OpenFOAM. As a possible integration to the SimScale platform is investigated of the rotating capabilities, the creation and setup of the simulations are displayed more thoroughly than for the other validations.

4.1.1 OpenFOAM - MRF

Using the Multiple Reference Frame capability from OpenFOAM, the case was set up. A cylinder of radius 0.4 [m] was specified using OpenFOAM's topoSetDict, inside which the rotating reference frame will be calculated.

4.1.1.1 Case creation

A mesh is created using Salome (version 7.2.0), with Netgen-2D settings *Max. Size*: 0.01, *Min. Size*: 0.001, *Growth Rate*: 0.1. After performing an extrusion on the mesh, a fine mesh as seen in figure 4.1 is generated. Faces are named in Salome in accordance with the boundaries. The mesh is exported from Salome into UNV format. To create an OpenFOAM case the *ideasUnvToFoam* tool provided in OpenFOAM can be used. If the OpenFOAM case name is MRF-OF-case and the produced mesh from Salome is MRF-Salome.unv a way to initialize the mesh in OpenFOAM would be,

```
 $ ideasUnvToFoam -case MRF-OF-case MRF-Salome.unv
```

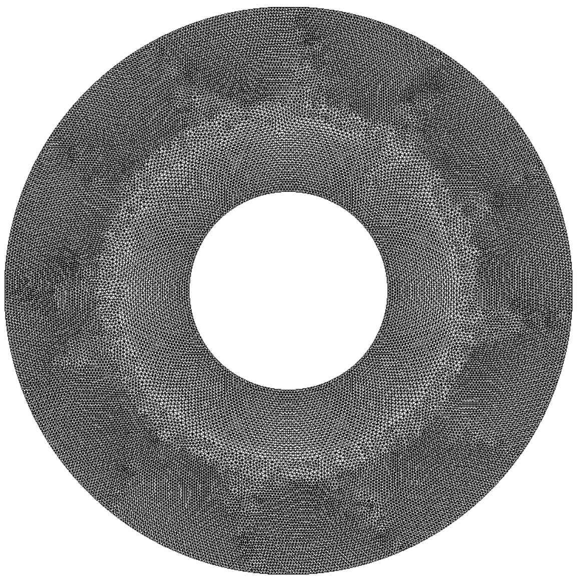


Figure 4.1: Mesh created by Salome

To use MRF in OpenFOAM *cellZones* have to be created to distinguish where different reference frames are used. In this case a cylinder around the smaller cylinder is defined to have a rotating reference frame. This is done using *system/topoSetDict* in the following way,

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
actions
(
    {
        name    rotorCellSet;
        type    cellSet;
        action  new;
        source  cylinderToCell;
        sourceInfo
        {
p1 (0 0 -1);
p2 (0 0 1);
radius 0.4;
        }
    }
{
name rotor;
type cellZoneSet;
action new;
source setToCellZone;
sourceInfo
{
set rotorCellSet;
}
}
);
```

To assign what reference frame is used inside this *cellZone* (called rotor) the file *system/fvOptions* is used. In the demonstrated case, it is the following.

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvOptions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
MRF1
{
    type            MRFSource;
    active          yes;
    selectionMode   cellZone;
    cellZone        rotor;

    MRFSourceCoeffs
    {
        nonRotatingPatches ();
        origin    (0 0 0);
        axis      (0 0 1);
        omega     constant 1E-3;
    }
}
```

Wall boundary conditions are assigned to the cylinders. As this is an incompressible simulation, only U and p have to be initialized and given boundary conditions. No slip boundary conditions are assigned for the velocity on the walls and zero gradient boundary conditions for the pressure on the walls. The viscosity is set to $\nu = 10^{-5}$ [m^2/s], which gives a Reynolds number of $Re = u_{\theta 1}(r_2 - r_1)/\nu = 22.75$. In appendix A.1.1, the files *system/fvSchemes* and *system/fvSolution* can be viewed, to see the schemes and discretizations used for the case. The case is ready to be run by typing the following into the terminal.

```
 $ topoSet
 $ simpleFoam
```

4.1.1.2 Results

Results are sampled along the line $y = 0$, $x \in [0.35,1]$ and the y-component of the velocity is plotted as u_θ. After sufficient convergence has been reached the velocity profile is found to be a good match as seen in figure 4.2. A more detailed look is given by the relative error of the velocity. The relative velocity error is given in equation (4.1) as

$$u_{\theta,rel} = \left| \frac{u_{\theta,simulation} - u_{\theta,analytical}}{u_{\theta,analytical}} \right| . \tag{4.1}$$

This is seen in figure 4.3.

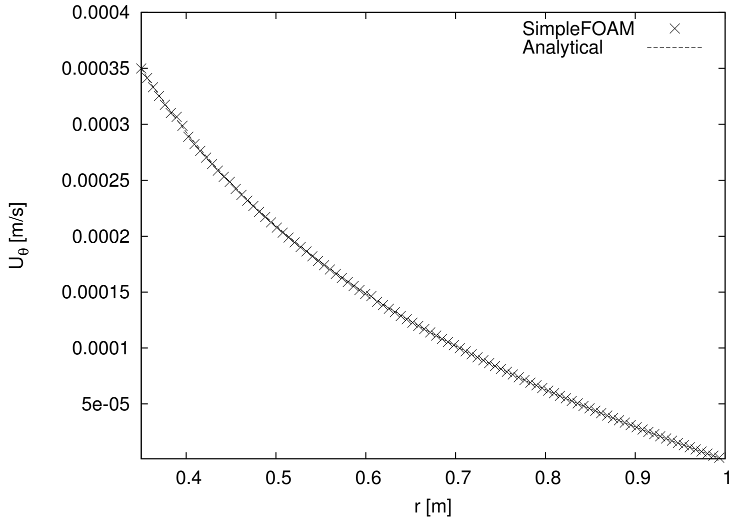


Figure 4.2: Velocity plot compared to analytical solution

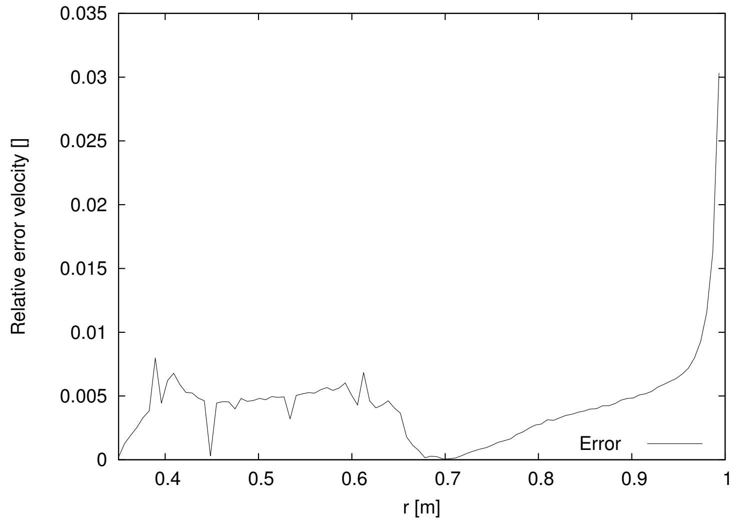


Figure 4.3: Relative error of the velocity

As can be seen in figure 4.4, the pressure profile is in good agreement but with a small offset close to the outer cylinder. The residuals are given in figure 4.5, which are of the order of magnitude 10^{-11} and are deemed as sufficiently small.

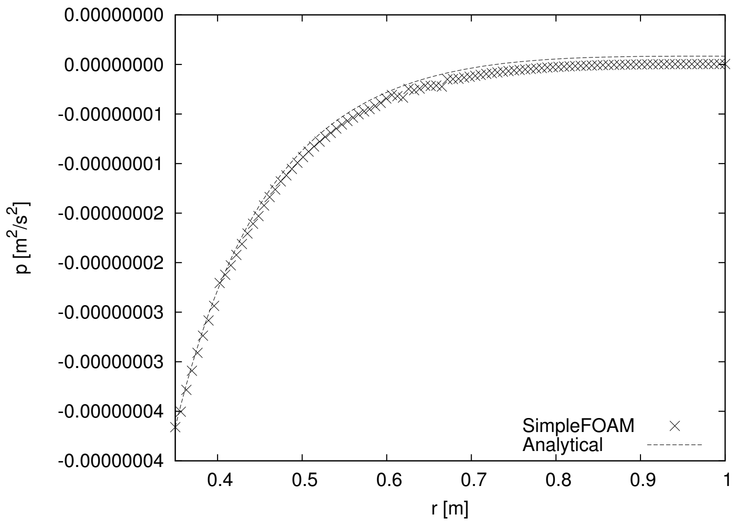


Figure 4.4: Relative error of the velocity

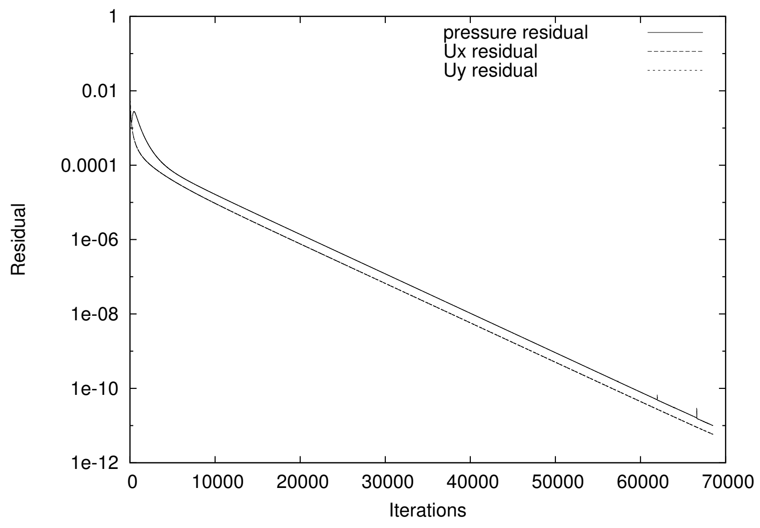


Figure 4.5: Residual plot

4.1.2 OpenFOAM - AMI

An arbitrary mesh interface approach was also done in OpenFOAM. This method is less straight forward than the MRF approach, as two seperate meshes are created with an AMI surface connecting them. It is advantageous as the mesh is moving with time and is the only choice for unsteady simulations.

4.1.2.1 Case creation

As in section 4.1.1.1 the meshes are created in Salome with the same mesh creation settings. However for AMI, two meshes are needed as output from Salome. An inner mesh from the AMI surface to the inner rotating cylinder, and an outer mesh from the AMI surface to the stationary outer cylinder.

The AMI surface was set at two different radii to compare the effects on the solution. The two radii chosen to place the AMI surface were set at $r = 0.4$ [m] and $r = 0.5$ [m]. The mesh files are exported from Salome as UNV and will be denoted InnerAMI.unv for the inner mesh and OuterAMI.unv for the outer mesh.

First the UNV-meshes are transformed into OpenFOAM cases by the *ideasUnvToFoam* tool.

```
 $ ideasUnvToFoam -case  InnerAMI-OF-Case InnerAMI.unv
 $ ideasUnvToFoam -case  OuterAMI-OF-Case OuterAMI.unv
```

Rename the boundaries so that none of the boundaries in the two OpenFOAM cases share the same name, as this will lead to problems when merging the meshes. Merge the meshes once this is made sure with the following commands in.

```
 $ mergeMeshes -overwrite OuterAMI-OF-Case InnerAMI-OF-Case
 $ mv OuterAMI-OF-Case AMI-OF-Case
```

This overwrites the OuterAMI-OF-Case case with the merged meshes. Often the non rotating mesh is written first (in this case the OuterAMI-OF-Case). This will assign the mesh of OuterAMI-OF-Case to a cellZone named *region0*, and the mesh of InnerAMI-OF-Case will be assigned to a cellZone named *region1* in the merged mesh. The second command renames the OpenFOAM case to AMI-OF-Case instead.

Enter the AMI-OF-Case and rename the AMI surfaces (the surfaces neighboring each other in the original meshes) to the form *AMI**, for instance *AMI-inner* for the inner surface and *AMI-outer* for the outer surface. No other boundaries may be named this way as this will lead to errors when creating the *faceSet* for the AMI surface.

Next create a *constant/dynamicMeshDict* to prescribe the movement of the inner mesh. The following *constant/dynamicMeshDict* was created for the Taylor-Couette case.

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
dynamicFvMesh   solidBodyMotionFvMesh;

motionSolverLibs ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
{
    cellZone        region1;

    solidBodyMotionFunction  rotatingMotion;
    rotatingMotionCoeffs
    {
        CofG        (0 0 0.05);
        radialVelocity (0 0 0.0572957795131); // deg/s
    }
}
```

Next create a *topoSetDict* to assign a faceSet to the AMI surface between the two merged meshes. The *faceSet* is needed for the boundary conditions assigned later.

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

actions
(
    {
        name    AMI;
        type    faceSet;
        action  new;
        source  patchToFace;
        sourceInfo
        {
            name "AMI.*";
        }
    }
);
```

The boundaries of the AMI surfaces have to be changed. This is done in the file *constant/polyMesh/boundary.* The exported boundaries from Salome will be of the form,

```
AMI-outer
   {
       type            patch;
       nFaces          314;
       startFace       183461;
   }
```

which have to be changed into this form,

```
AMI-outer
   {
       type            cyclicAMI;
       nFaces          314;
       startFace       183461;
       matchTolerance  0.0001;
       neighbourPatch  AMI_inner;
       transform       noOrdering;
   }
```

The same is done for the *AMI-inner* boundary.

Assuming that the starting time is *0*, the initial conditions also have to be changed for the AMI surfaces. Assign in the U and p files in the *0* directory the following boundary conditions for the AMI surfaces.

```
AMI-outer
   {
       type            cyclicAMI;
       value           $internalField;
   }
```

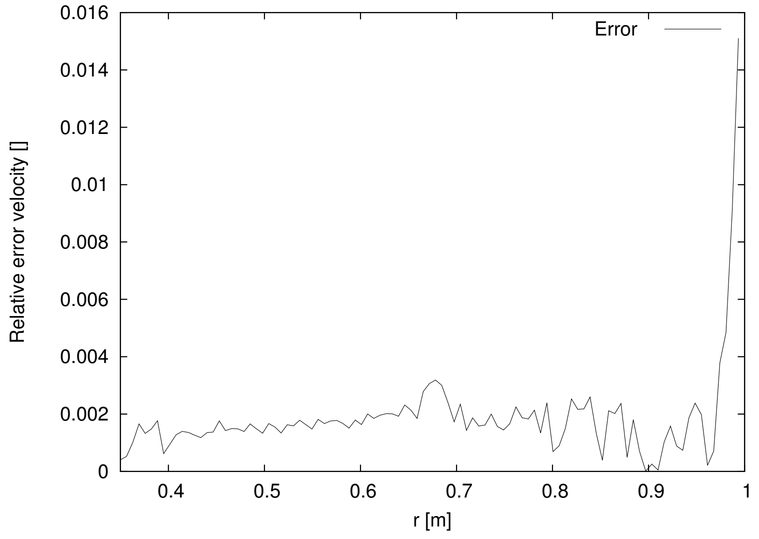
The inner cylinder boundary is assigned a movingWallVelocity type boundary condition and for the pressure a *zeroGradient* boundary condition. For the outer wall a *fixedValue* condition is applied for the velocity and also a zeroGradient boundary condition is set for the pressure. A uniform 0 velocity and pressure field are used as initial conditions.

With these steps the Taylor-Couette case can now be run with AMI. Start the simulation by typing in the terminal,

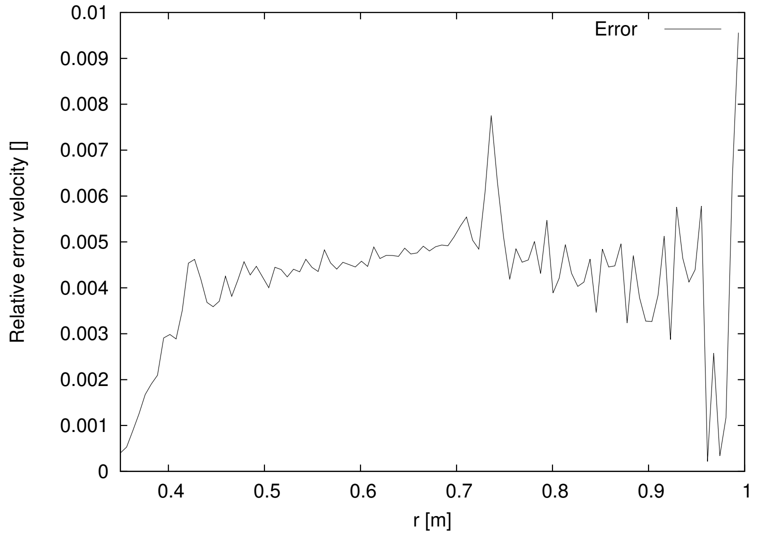
```
$ topoSet
$ pimpleDyMFoam
```

4.1.2.2 Results

After $T = 50000$ [s], which is roughly equivalent to 8 complete revolutions of the inner cylinder, the unsteady simulation is deemed to have reached steady state. The relative error for the velocity is shown in figure 4.6. As can be seen, the solutions are reasonably converged with only an error of $\sim 1\%$ compared to the analytical solution. Both solutions have the largest error close to the stationary wall.



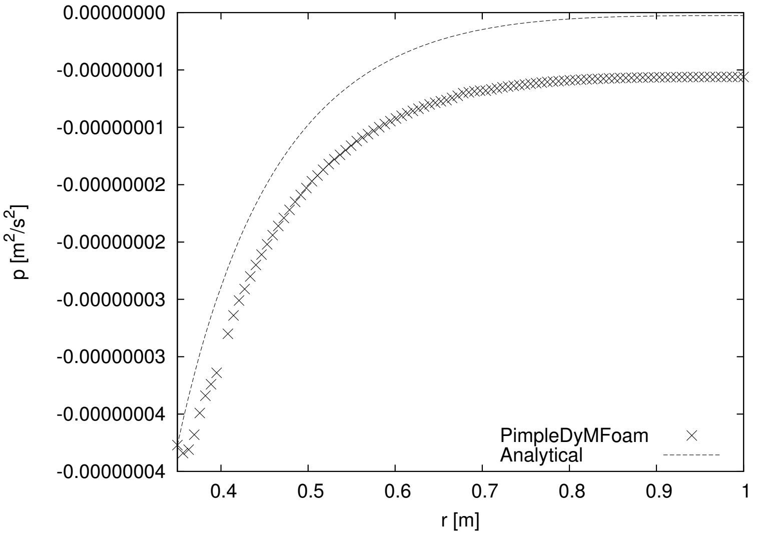
(a) $r_{\text{AMI}} = 0.4$



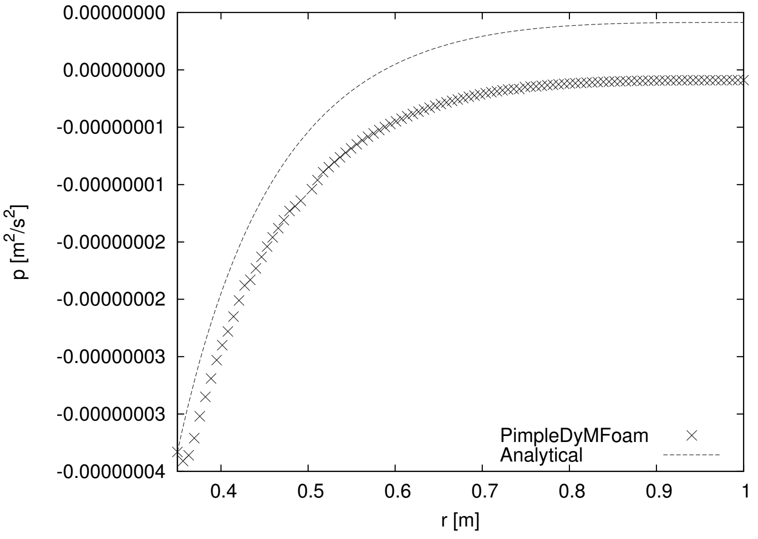
(b) $r_{\text{AMI}} = 0.5$

Figure 4.6: Relative velocity error for (a) AMI surface at r=0.4, (b) AMI surface at r=0.5

For the pressure a rather large offset to the analytical solution is found for both AMI radii, as seen in figure 4.7. Also the influence of where the AMI surface is located seems to influence the solution accuracy. Outside the rotating zone, the pressure offset seems to be constant, thus the pressure solution inside the rotating zone might be introducing an offset compared to the analytical solution.



(a) $r_{\mathrm{AMI}} = 0.4$



(b) $r_{\mathrm{AMI}} = 0.5$

Figure 4.7: Axial pressure for (a) AMI surface at r=0.4, (b) AMI surface at r=0.5

In appendix A.1.2 the schemes and the methods of discretization can be viewed.

4.1.3 Gerris

The case was also conducted in Gerris as to determine the moving mesh capability of the solver. Gerris does not use AMI or MRF to produce rotating cases, but instead moves the mesh and utilizes adaptive mesh refinement around the body.

4.1.3.1 Case creation

Creating the case in Gerris is a simple procedure. As the following case is a *GfsMovingSimulation*, a geometry file for the rotating cylinder is needed. Using the following command line in the terminal a cylinder with radius 0.25 is created.

```
$ shapes ellipse > cylinder.gts
```

Two cylinders are then prescribed in Gerris by the command,

```
SolidMoving cylinder.gts {scale = 1.4} {level = 10}
Solid (ellipse (0,0,1.0,1.0)) {flip = 1}
```

where the flip command is necessary for the solver to realize the flow is internal and the scale command scales the rotating cylinder to the correct size. As the incompressible Navier-Stokes equations are solved a viscosity has to be prescribed. A constant viscosity is given through,

```
SourceViscosity {} 1e-5
```

To rotate only the inner cylinder, the SurfaceBc command can be used in the following way,

```
SurfaceBc U Dirichlet (x*x + y*y > 0.4*0.4 ? 0. : - 1e-3*y)
SurfaceBc V Dirichlet (x*x + y*y > 0.4*0.4 ? 0. :   1e-3*x)
```

A Godunov scheme is used to calculate the convective terms. The refinement level of the mesh is increased around the moving solid boundary. With these settings the Taylor-Couette case is set up. See appendix A.1.3 for the full configuration file.

4.1.3.2 Results

After $T = 50000$s the velocity has converged close to the analytical solution except at the node closest to the stationary wall, as can be seen in figure 4.8. A maximum deviation of 2% from the analytical solution was registered, except at the stationary node which had a deviation of almost 90%. For the pressure a reasonably good convergence is registered, as can be seen in figure 4.9.

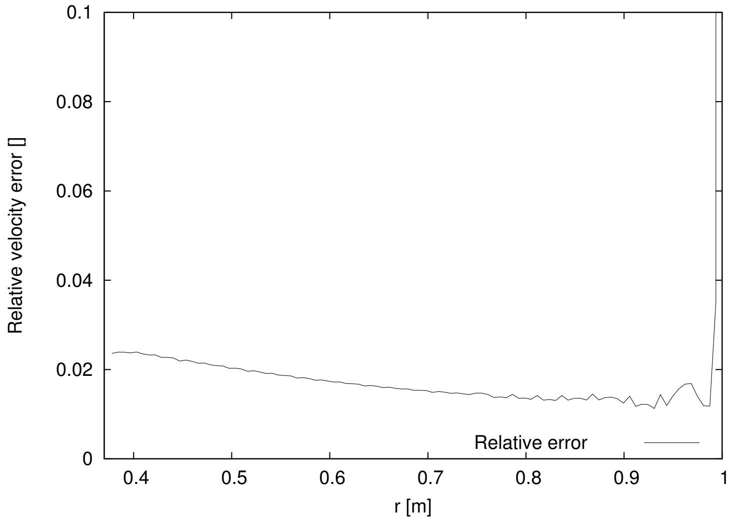


Figure 4.8: Relative velocity error

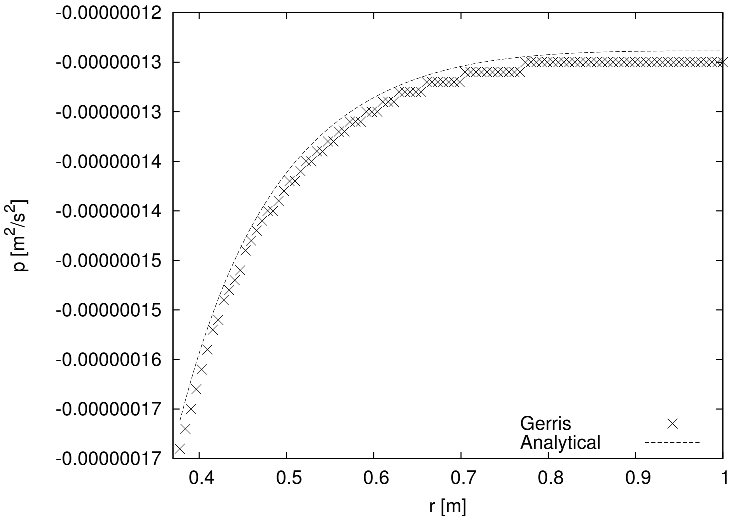


Figure 4.9: Pressure profile

4.2 Turbulent flat plate

A turbulent flat plate was chosen to validated the turbulent incompressible capabilities of OpenFOAM and SU^2. Unfortunately, in SU^2 the incompressible solver was deemed unstable and thus the case was calculated with the compressible solver of SU^2. This is expected to improve the results for the SU^2 solver as this case is a compressible case. The case set up can be seen in figure 4.10.

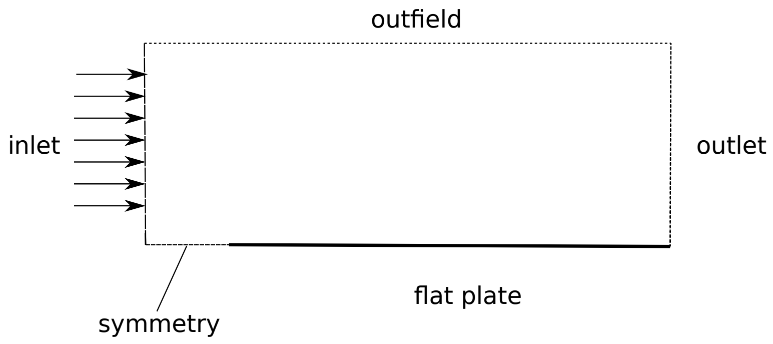


Figure 4.10: Turbulent flat plate

The flat plate is 5 m long in x-direction and the symmetry boundary is 0.5 m long. The height of the simulated area is taken to be 0.8m high in y-direction. From Wieghardt [1] the inlet conditions are, $M = 0.2$, $P = 101325$ [Pa] and $T = 294.44$ [K]. The used fluid is air.

4.2.1 OpenFOAM setup

For the OpenFOAM case an incompressible approach is taken. Thus the pressure and temperature are not of importance. The case is calculated using a $k - \omega$ SST turbulence model. Symmetry boundary conditions are applied at the outfield and symmetry boundaries. For the pressure, zero gradient boundary condition is applied on the flat plate and inlet, and for the outlet a Dirichlet boundary is enforced and set to 0. For the velocity zero gradient condition is set at the outlet and no slip boundary condition at the flat plate. As for the inlet boundary condition a Mach number of 0.2 is equivalent to 68.8 [m/s]. For the turbulent variables k and ω, OpenFOAM wall functions are applied at the flat plate. For ω the *omegaWallFunction* is used and for k the wall function *kqRWallFunction* is applied. For the turbulent viscosity ν_t a wall function is also applied through *nutkWallFunction*.

For the turbulent variables inlet conditions the turbulent kinetic energy k, can be deduced from [6],

$$k = 3/2(UI)^2. \tag{4.2}$$

The variable I is the turbulent intensity. In this simulation it is chosen to $I = 0.05$. Thus in this case the inlet condition is $k = 17.75$ [m^2/s^2]. For ω a mixing length of approximately $l \approx 0.085$ [m] is taken by choice, which gives the inlet condition for ω from,

$$\omega = C_\mu^{-1/4} \frac{\sqrt{k}}{l}. \tag{4.3}$$

The constant is $C_\mu = 0.09$.

4.2.2 SU^2 setup

In SU^2 a compressible approach is done. The Mach number and temperature are assigned to the free stream values. The free stream Reynolds number is set to $Re = 4.5 \cdot 10^6$. The flat plate is given a no slip boundary condition. For the inlet a total temperature and total pressure are assigned. These are calculated as,

$$\begin{aligned} T_{Total} =& T_s \left(1 + \frac{\gamma - 1}{2} M^2\right) \\ P_{Total} =& p_s \left(1 + \frac{1}{2} \gamma M^2\right) \end{aligned} \tag{4.4}$$

where p_s is the static pressure and T_s is the stagnation temperature. The ratio of specific heats is assigned as $\gamma = 1.4$. For turbulence modeling the SA model is chosen. Attempts were made with the $k - \omega$ SST model but unfortunately no converging configuration was found.

4.2.3 Mesh

The mesh is created using OpenFOAM's *blockMesh* tool. The mesh was given 260 cells in the x-direction. 70 of these cells are placed from the inlet until the start of the flat plate, and the rest is distributed over the flat plate. The cells decrease the closer they are the point separating the symmetry boundary condition and the flat plate. In the y-direction 180 cells are distributed with decreased size the closer they are to the flat plate.

4.2.4 Results

With the 260×180 mesh the following results for the skin friction coefficient C_f was calculated and compared to experimental results. The skin friction coefficient is calculated from,

$$C_f = \frac{\tau_w}{\frac{1}{2}\rho U_\infty} \tag{4.5}$$

where τ_w is the wall shear stress, ρ is the density and U_∞ is the free stream velocity [21]. From the simulations the friction coefficients are calculated and shown in figure 4.11.
To evaluate the quality of the mesh a plot of y^+ over the flat plate is conducted for the OpenFOAM case, as seen in figure 4.12.

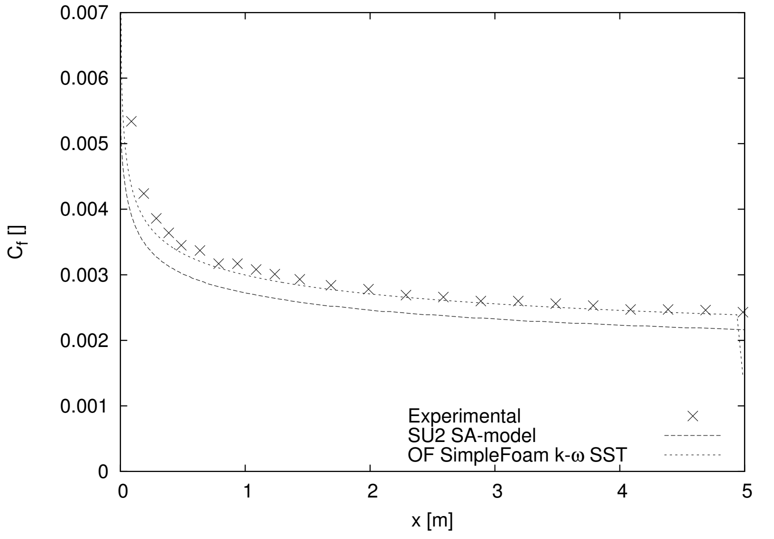


Figure 4.11: Skin friction coefficient

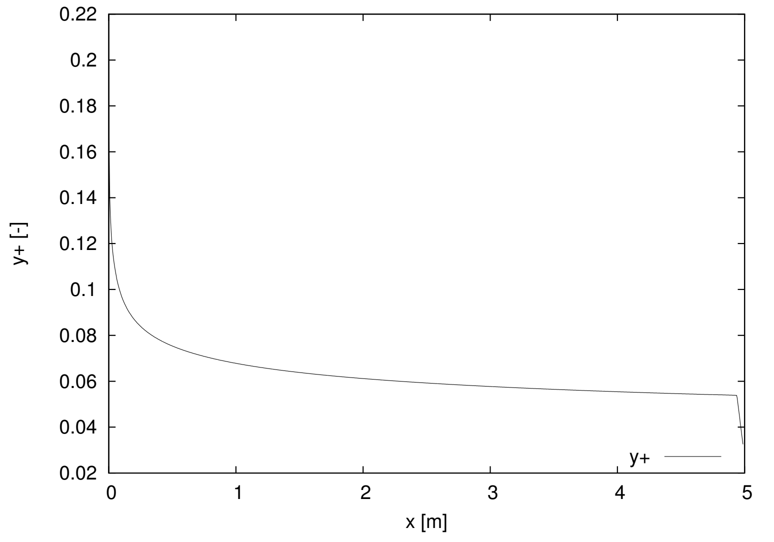
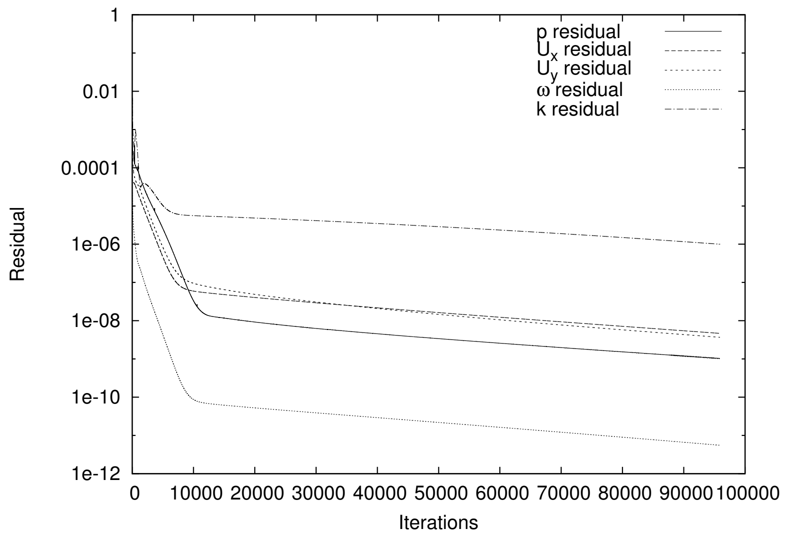
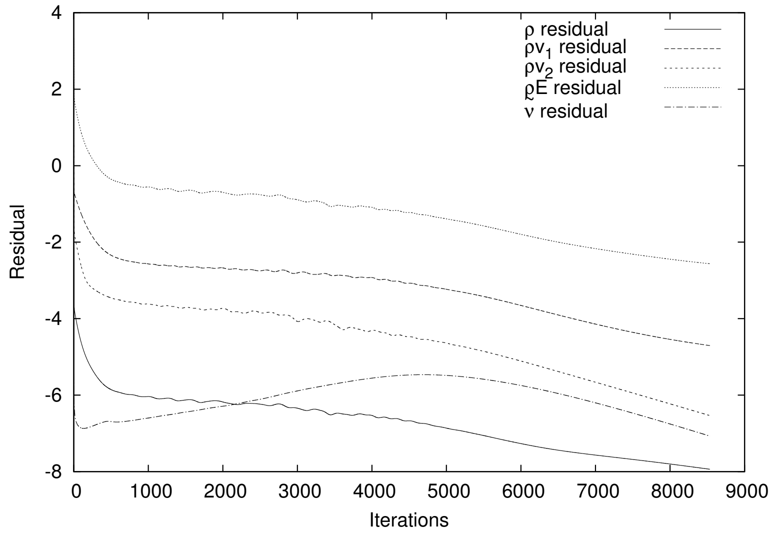


Figure 4.12: $y+$ plot from OpenFOAM

Residuals for both the OpenFOAM and SU^2 case can be seen in figure 4.13. It should be noted that the residual axis of SU^2 is logarithmic. And as can be seen the residuals are reduced to a reasonably low level.



(a) OpenFOAM residuals



(b) SU^2 residuals

Figure 4.13: Residuals of turbulent flat plate simulation

4.3 RAE2822

An investigation of the compressible capabilities of OpenFOAM and SU^2 were also conducted. The case set up can be seen in figure 4.14.

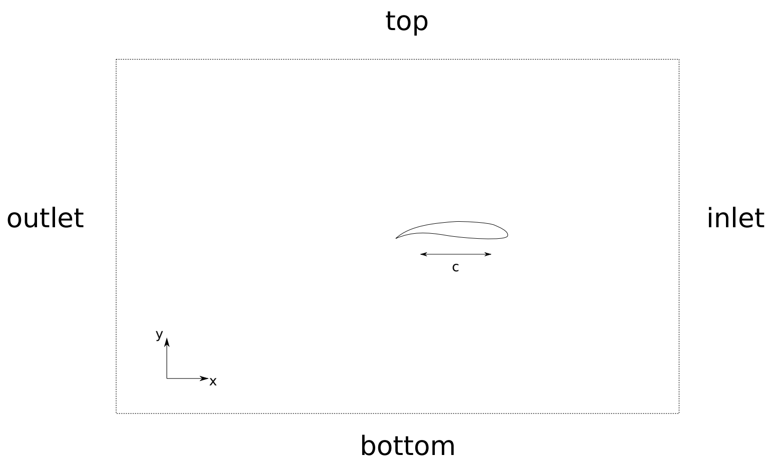


Figure 4.14: RAE2822 case

The calculated airfoil is a RAE2822 with a chord length $c = 0.3048$ [m]. The airfoil resides in a free stream where the Mach number is $M = 0.729$, the static pressure $P_\infty = 108987$ [Pa], a temperature of $T_\infty = 255.55$ [K] and an angle of attack $\alpha = 2.39°$. Air is used as a medium and given the already mentioned data. Given the data the density in the free stream and the dynamic viscosity can be calculated. From the ideal gas law the density follows from,

$$\rho_\infty = \frac{P_\infty}{R_{specific} T_\infty} \tag{4.6}$$

where $R_{specific} = 287.87$ [J/kg·K] for air. This gives the free stream a density of $\rho_\infty = 1.4857$ [kg/m^3]. As for the dynamic viscosity μ, Sutherland's law is used as in equation (2.20), which gives a value for the free stream dynamic viscosity of $\mu = 3.46126 \cdot 10^{-6}$ [kg/(m·s)].

The mesh for this case is done by *blockMesh* created by Daniele Trimarchi from Southampton University. The mesh is spanned in the y-direction by [−1.5,1.5] and in the x-direction by [−3.0,1.5]. Notice that the flow in this configuration is in the negative x-direction. For the SU^2 case the mesh was modified to achieve a lower $y+$-value ($y+ \approx 5$) as well, due to the fact that the utilized version of SU^2 (version 2.0.8) does not have wall functions implemented. Also, as convergence was unable to be found for SU^2 with the created *blockMesh*, the provided plot3D mesh from [2] was converted into SU^2 to test whether convergence could be achieved with this mesh.

For the boundary conditions a free stream boundary condition is applied to all boundaries except the airfoil which has a no-slip boundary condition in SU^2.

In OpenFOAM a wave transmissive condition is applied at the outlet for the pressure and zero gradient boundary conditions are applied elsewhere. For the velocity a free stream boundary conditions is applied everywhere except at the wing where no-slip is enforced. Temperature is set as uniform $T = 255.55$K at the inlet and zero gradient conditions are enforced on all other boundaries. As for turbulence the SA-model is used in both OpenFOAM and SU^2. Inlet conditions for the variable $\tilde{\nu}$ is set according to the source code from SU^2 which is,

$$\tilde{\nu}_\infty = 3\frac{\mu_\infty}{\rho_\infty}. \tag{4.7}$$

In OpenFOAM the same parameters for the Sutherland's law is in SU^2 are used to calculate the temperature dependency of μ_{dyn}.
Measurements are made of the negative pressure coefficient,

$$C_P = -\frac{p - p_\infty}{\frac{1}{2}\rho_\infty u_\infty^2} \tag{4.8}$$

and are compared to experimental data provided at [2].

4.3.1 Results

As the solver *rhoCentralFoam* is unsteady, steady state is assumed to be reached after 0.25 [s] which is roughly 13 times the required time for the free stream to pass through the domain. In figure 4.15 the results obtained using *rhoCentralFoam* can be seen, note that the chord length has been normalized in these plots. Results in SU^2 are more difficult

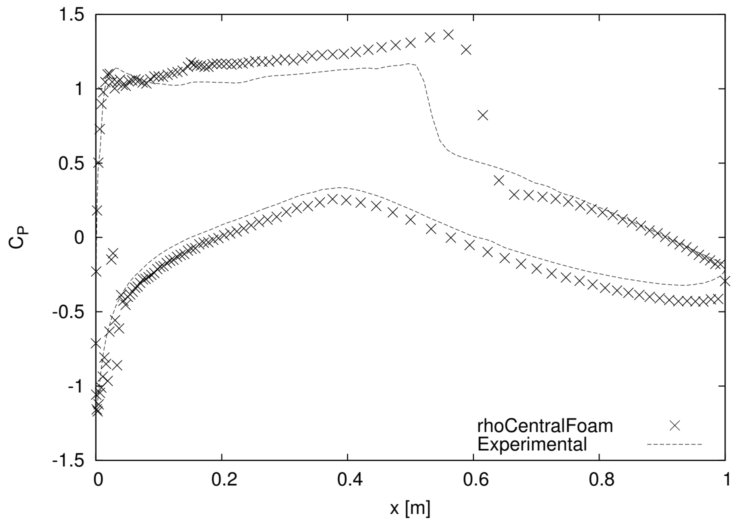


Figure 4.15: C_P from *rhoCentralFoam*

to demonstrate as the solution diverges when seemingly close to convergence. Different configurations were attempted to find a converging solution, unfortunately, none was found during the duration of this thesis work. Demonstrated here is a JST solver with $CFL = 0.8$ at quasi time step $t = 3500$ using the converted mesh from NASA [2]. The pressure coefficient is close to convergence as seen in figure 4.16 but as can be seen from the residual plot in figure 4.17 it diverges shortly after.

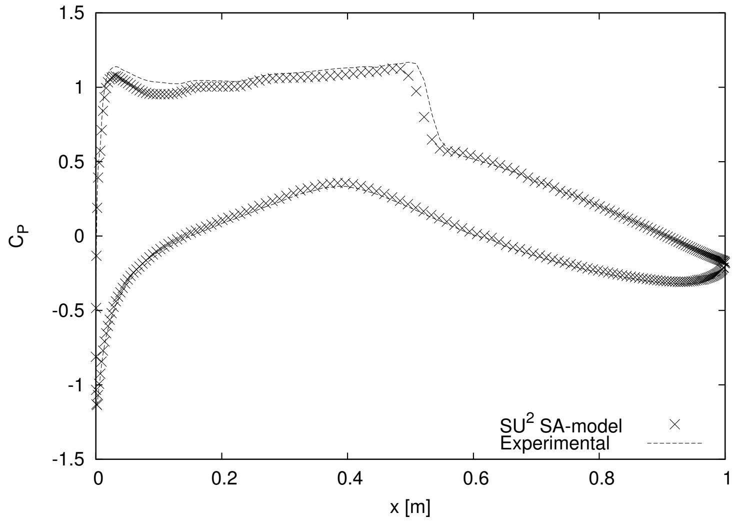


Figure 4.16: C_P right before divergence for SU^2 with NASA mesh

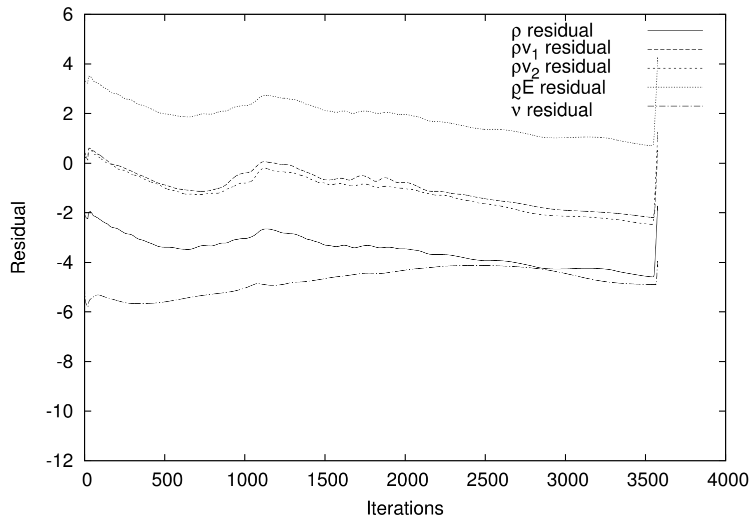


Figure 4.17: Residuals for SU^2 with NASA mesh

Also the mesh created from *blockMesh* in OpenFOAM had convergence issues. Another simulation utilizing a JST algorithm is shown here at quasi time step $t = 6500$. A Roe

2nd order solver was also attempted, but was also unable to produce convergence. In figure 4.18 the pressure coefficient can be viewed right before divergence and in figure 4.19 the residuals can be viewed.

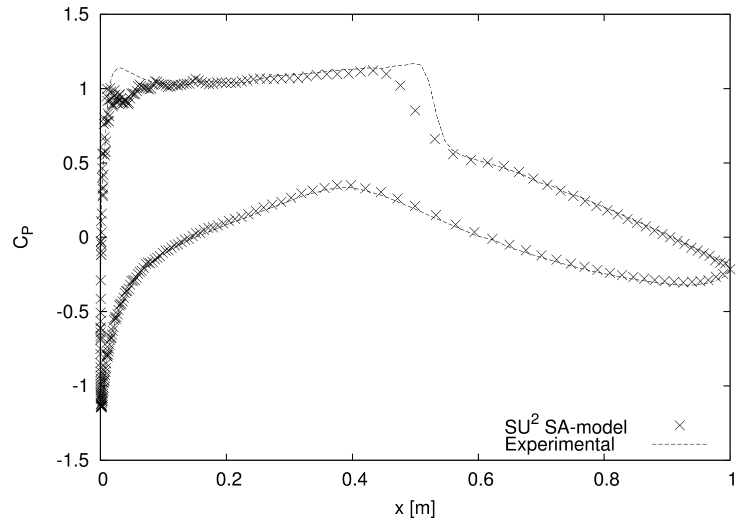


Figure 4.18: C_P right before divergence for SU^2 with OpenFOAM mesh

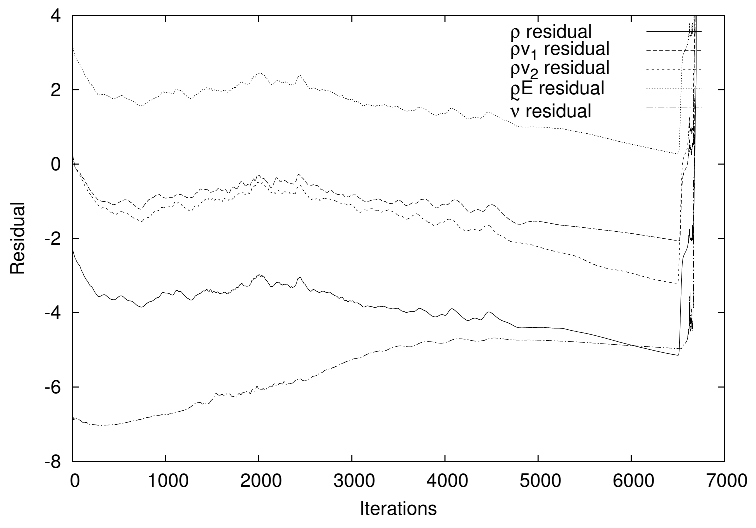


Figure 4.19: Residuals for SU^2 with OpenFOAM mesh

4.4 Integration of SU^2

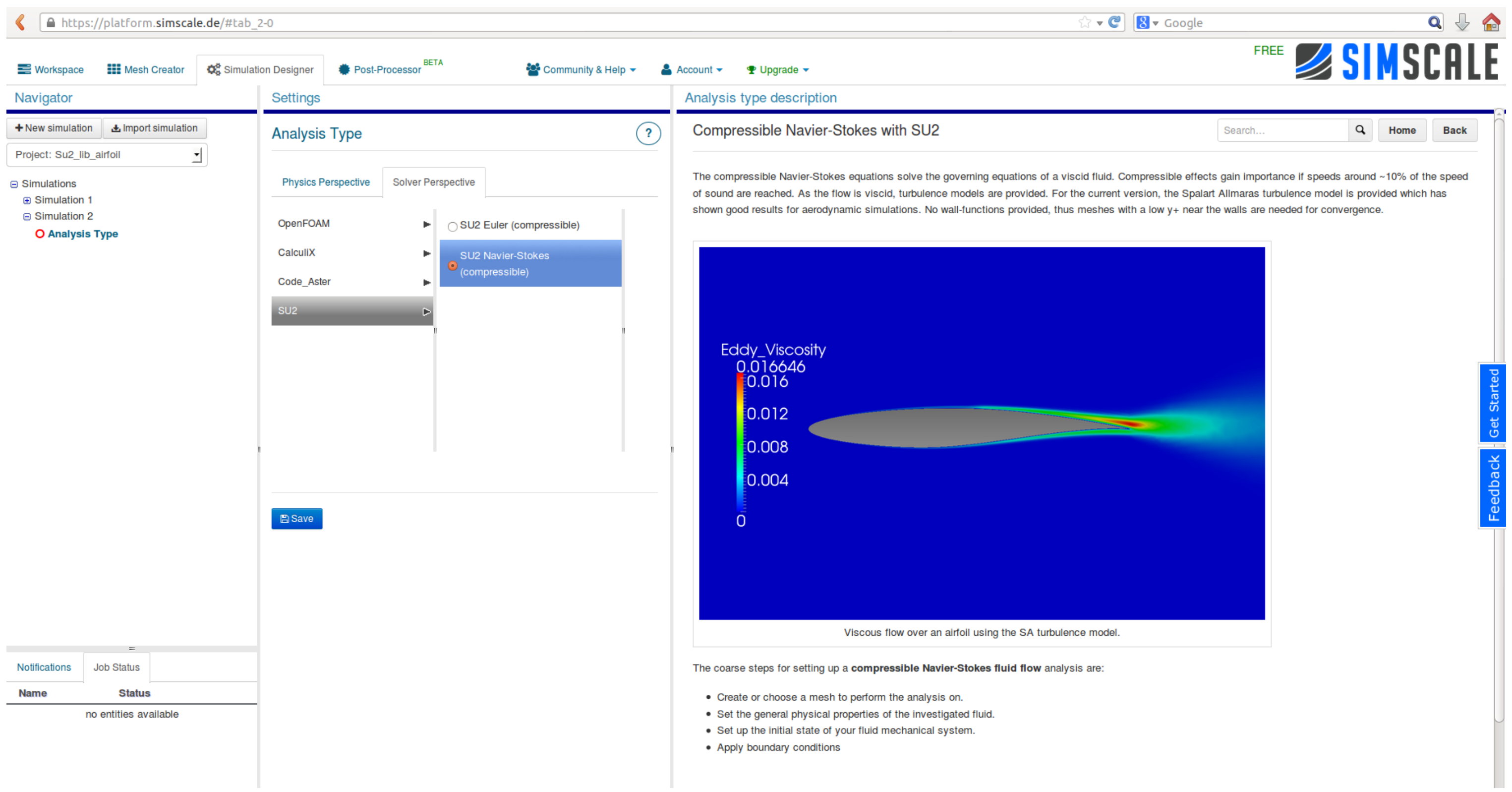
The stable SU^2 version 2.0 was integrated on the SimScale platform. An inviscid and a viscid compressible fluid solver are provided for the customer. After creating a suitable mesh on the platform, SU^2 can be chosen in the Analysis type menu, as can be seen in figure 4.20.

The boundary conditions implemented for SU^2 was chosen to be a farfield BC (Boundary Condition), symmetry BC, inlet BC, outlet BC and a Navier-Stokes BC (no-slip) for the viscid solver and a Euler BC (slip) for the inviscid solver.

Fluid properties can be assigned under the Simulation Model Properties. Here fluid specific properties such as the isentropic exponent (γ), the specific gas constant R and the laminar and turbulent Prandtl numbers are assigned. Also freestream properties such as the Mach number, Reynolds number, pressure and temperature are prescribed here. The kinematic viscosity is given from the assigned Reynolds number and the density calculated using the kinematic viscosity calculated from the Sutherland's law.

SU^2 contains a variety of numerical schemes and solvers, these are prescribed under Numerics, as can be seen in figure 4.21. The convergence criteria can be chosen as either a Cauchy criterion or the reduction of the maximum residual to a certain level. For the 2.0 version the only available turbulence model is the SA-model, thus only this turbulence model is available on the platform. For the discretized equations a linear solver is chosen here, two Krylov based solvers (GMRES and BCGSTAB) are provided in SU^2 and one stationary iterative solver (LU-SGS).

A converter of the user provided input data from the platform was written to create a SU^2 configuration file. The exact procedure can not be disclosed due to trademark secrets.



Analysis Type

○ SU2 Euler (compressible)
[✓ SU2 Navier-Stokes (compressible)]

Compressible Navier-Stokes with SU2

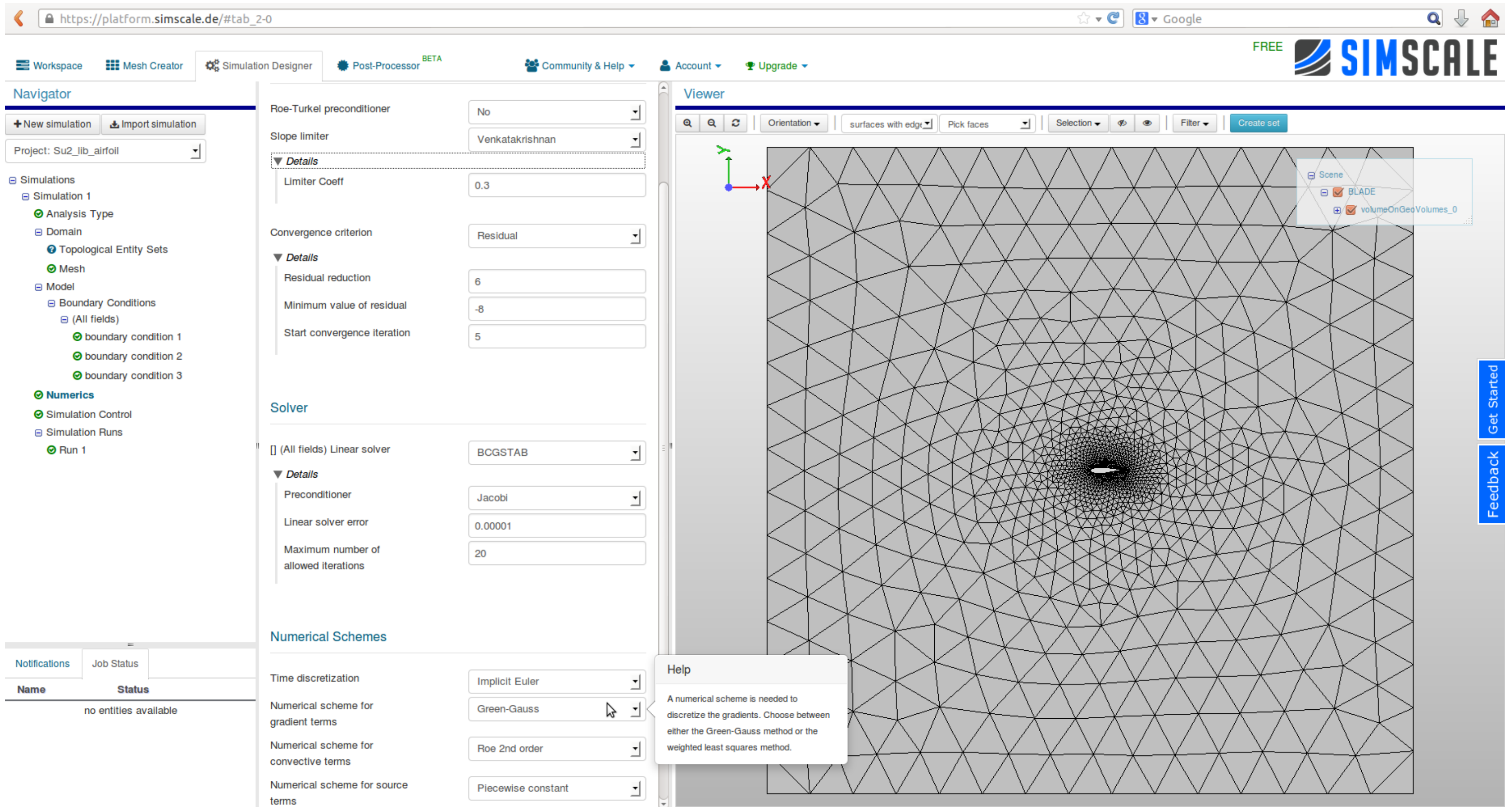
The compressible Navier-Stokes equations solve the governing equations of a viscid fluid. Compressible effects gain importance if speeds around ~10% of the speed of sound are reached. As the flow is viscid, turbulence models are provided. For the current version, the Spalart Allmaras turbulence model is provided which has shown good results for aerodynamic simulations. No wall-functions provided, thus meshes with a low y+ near the walls are needed for convergence.

Viscous flow over an airfoil using the SA turbulence model.

The coarse steps for setting up a **compressible Navier-Stokes fluid flow** analysis are:

- Create or choose a mesh to perform the analysis on.
- Set the general physical properties of the investigated fluid.
- Set up the initial state of your fluid mechanical system.
- Apply boundary conditions

Figure 4.20: Available solvers for SU^2



Roe-Turkel preconditioner: No
Slope limiter: Venkatakrishnan
Details
Limiter Coeff: 0.3

Convergence criterion: Residual
Details
Residual reduction: 6
Minimum value of residual: -8
Start convergence iteration: 5

Solver

[] (All fields) Linear solver: BCGSTAB
Details
Preconditioner: Jacobi
Linear solver error: 0.00001
Maximum number of allowed iterations: 20

Numerical Schemes

Time discretization: Implicit Euler
Numerical scheme for gradient terms: Green-Gauss
Numerical scheme for convective terms: Roe 2nd order
Numerical scheme for source terms: Piecewise constant

Help

A numerical scheme is needed to discretize the gradients. Choose between either the Green-Gauss method or the weighted least squares method.

Figure 4.21: Available numerical schemes for SU^2

5

Conclusions and prospects

The rotating Taylor-Couette validation showed different methods of simulating rotating bodies. What can be observed is that the MRF approach in OpenFOAM gave the best results. This might be expected from a simple geometry as a cylinder and the fact that the flow is steady.

As the approaches using AMI and Gerris are unsteady, steady state might not have been reached after 8 revolution of the inner cylinder. But the velocities seem to have reached reasonably close to the analytical solution. For the velocity the AMI solution is actually closer to the analytical one than the MRF or Gerris simulations. For all simulations, the stationary outer cylinder is the most difficult to simulate, as here the largest relative error to the analytical solution is observed. This can be attributed to the near zero velocity at the walls, which is a demanding criterion for the numerical solution to fulfill. Thus the data point closest to the stationary wall should not be put too much emphasize on.

For the pressure the AMI simulation had a large offset compared to the analytical solution. This is not noted for the Gerris and MRF simulations which produce reasonably accurate results. The explanation to this offset could be the mesh, as AMI might require a finer mesh due to its more advanced algorithm than MRF. However for the incompressible case, the convergence of the pressure equation is not of importance as the aim is to solve the continuity equation and the Navier-Stokes equation.

Gerris produced good results with only $\sim 2\%$ offset compared to the analytical velocity (except at the stationary wall). For this case the performance of the unsteady solver was better, compared to the AMI approach of OpenFOAM. However, the mesh quality between the two solvers cases can not be compared, so it is difficult to say whether one or the other had a significant better mesh. Furthermore, Gerris does not support turbulence models, and thus needs to resolve all turbulent scales through adaptive mesh refinement. This effects simulation time for complex rotating geometries with turbulence, and utilizing AMI from OpenFOAM has an advantage here. Also as Gerris is strictly an incompressible solver, compressible cases can not be set up. This can be done in OpenFOAM with for instance *rhoPimpleDyMFoam* or *rhoCentralDyMFoam*. The strength of Gerris over OpenFOAM for the rotating cases is the ease with which it is set up, and for simpler cases not much extra computational time is needed for the simulation.

To integrate rotating body capability on the SimScale platform, new user features would have to be implemented. For MRF, the user has to chose a volume to create a cell set in which the rotating frame is specified. A variety of shapes can be used for this but the most suitable might be the use of a cylinder. Rotation has to be prescribed in the *fvOptions* file where the origin and the axis of rotation are needed to be specified, which is not that difficult to achieve.

For the AMI approach in OpenFOAM a more tedious work flow is needed to set up a simulation. As a sliding interface is needed between two meshes, two separate meshes have to be created. When the geometry is created a surface has to be added which separates the geometry into an inner rotating geometry and an outer stationary geometry. The inner and outer geometries have to be meshed separately and the meshes have to be transformed into OpenFOAM cases. Boundary conditions will have to be changed and boundaries renamed, this will require some clever scripting to implement on the platform. The work flow is manageable to extend to the platform, but will require more work than for the MRF case.

For the Gerris solver the import of a geometry object, in the *.gts* format, is needed if the object is to be rotated. The tool *stl2gts* can be used to convert a STL geometry file into the correct format. Surface boundary conditions are applied with an analytical expression on the objects boundaries and so a rotating motion can be achieved. The issue with Gerris is the requirement that the computational domain needs to be enclosed in (scalable) unit squares. For external flows this is not an issue, but for internal flows, with complex outflows, difficulties arise when trying to apply the correct boundary conditions onto boundaries.

For the implementation of rotating bodies on the SimScale platform, a natural step forward would be to integrate the MRF capability of OpenFOAM. This would only require that a user can specify a volume where to apply the MRF region. As MRF is only suitable for steady state simulations, AMI would need to be implemented at some point. This is not as straight forward to implement as MRF, but is possible to implement with some new meshing tools. As for Gerris, a complete new work flow is needed to handle this solver on the platform. From a platform perspective to implement Gerris solely for its capabilities for rotating bodies is not feasible. However, Gerris as a software is a flexible and easy to use tool for CFD calculations, it would be recommended to integrate this solver at some point in the future.

The turbulent flat plate case was set up to test the turbulence models of the different solvers. In OpenFOAM the simulation was calculated as incompressible unlike SU^2 which is calculated as compressible. This would suggest that SU^2 would perform better than OpenFOAM as the case is compressible, but incompressible simulations give almost the same result. For OpenFOAM a $k-\omega$ SST model was chosen, which was attempted to use as well for SU^2, but a converging solution was only achieved for the SA-model. The skin friction coefficient, from experimental data, was approximated well in the OpenFOAM case, whereas a noticeable offset is noted in the converged SU^2 solution. As no wall functions are implemented in SU^2 at this time, a low enough y^+ value is needed, this is confirmed to be the case from figure 4.12. As the SA-model is used, it is expected to perform worse than the $k-\omega$ SST model. But it would seem as if the offset is too large to only be explained by the turbulence model. There might be some convergence issues with SU^2, as it was difficult to set up the case incompressible or with a $k-\omega$ SST turbulence model, more investigation will have to be put into this. Also it has to be taken into consideration that SU^2 is still under heavy development, and performance can vary greatly between versions.

For the compressible flow validation a transonic airfoil (RAE2822) was investigated. The OpenFOAM case, using *rhoCentralFoam*, give the pressure coefficient profile as can be seen in figure 4.15. It seems as if the shock is overpredicted and smeared out, this can be explained by the fact that a linear upwind method was used for the discretization of the divergence terms. A TVD scheme should be used to achieve better result for this case.

The SU^2 cases show initially good convergence and seem from the residuals to be heading for convergence, before abruptly diverging. The mesh created with *blockMesh* seem to have larger convergence issues than the mesh taken from NASA. But both diverge while being close to convergence. Both a central scheme (JST) and a Riemann approximative solver (Roe method) were applied to the problem, but both failed to reach convergence. The case, as it contains shocks is expected to be numerically unstable. However as the divergence is occurring so abruptly in the simulation, there might be some convergence issues in the steady state algorithm of SU^2. However there is too little data to draw a conclusion from this limited data set, and more set up variations could be tested for the airfoil. Especially interesting would be to find convergence with the NASA mesh as this could be directly compared to the results achieved from CFL3D (NASA's CFD solver).

From a solver perspective, SU^2 should be superior to *rhoCentralFoam* as it has several central and approximate Riemann solvers, whereas *rhoCentralFoam* only has two available central solvers. Unfortunately the capabilities of *rhoPimpleFoam* and *rhoSimpleFoam* were not investigated, otherwise a comparison between SU^2 and OpenFOAM as compressible solvers could be drawn.

For the integration of SU^2 on the SimScale platform the basic compressible capabilities were integrated. At this moment the multigrid capability of SU^2 was not put on the platform, this is something to be done in the future. When specifying a simulation in SU^2 the Reynolds number is assigned rather than the viscosity, and the inlet velocity is specified by total pressure and total temperature than the actual velocity. To increase user friendliness, a tool to set up dimensionalized simulations can be created. As only the basic capabilities of SU^2 were put on the platform, a variety of analysis types are still available for integration. Especially interesting would be to integrate the adjoint solver which the developers at SU^2 have put a lot of time into. However, further integration of the SU^2 software would be advised to be done after the release of the upcoming 3.0

version.

As the three different solvers have been investigated some strengths and weaknesses became clear. OpenFOAM has a lot of different solvers and can solve a variety of cases. But it seems to be limited when it comes to compressible flows and the case set up of OpenFOAM is tedious.

For SU^2 a variety of solvers was found for compressible cases and it is simple to set up. However it suffers from, what was found in this thesis, being difficult to achieve convergence. Although it should be kept in mind that this solver is still undergoing development and that these issues probably will be resolved in the near future.

Gerris was found to be a capable solver for incompressible flows and it was extremely easy to set up cases. The drawbacks of the solver was found to be its requirement to define the computational domain with a set of unit cubes. This makes it difficult to integrate on the SimScale platform.

Bibliography

[1] K. Wieghardt, W. Tillman, *On the Turbulent Friction Layer for Rising Pressure*, Tech. Rep. TM-1314, NACA (1951).

[2] NASA (2008), *RAE 2822 Transonic Airfoil: Study #4*, downloaded: (21-11-2013). URL `http://www.grc.nasa.gov/WWW/wind/valid/raetaf/raetaf04/raetaf04.html`

[3] F. Palacios, R. M. Colonno, A. C. Aranake, A. Campos, S. R. Copeland, T. D. Economon, A. K. Lonkar, T. W. Lukaczyk, T. W. R. Taylor, J. J. Alonso, *Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design*, in: 51st AIAA Aerospace Sciences Meeting and Exhibit, Grapevine, Texas, USA, 2013, AIAA Paper 2013-0287.

[4] Stéphane Popinet (2013), *Gerris Flow Solver*, downloaded: (21-11-2013). URL `http://gfs.sourceforge.net/wiki/index.php/Main_Page`

[5] OpenFOAM Foundation (2013), *OpenFOAM user guide*, downloaded: (4-11-2013). URL `http://www.openfoam.org/docs/user/`

[6] H. Versteeg, W. Malalasekera, *An Introduction to Computational Fluid Dynamics*, 2nd Edition, Pearson Education Limited, Harlow, England, 2007.

[7] P. R. Spalart, S. R. Allmaras, *A One-Equation Turbulence Model for Aerodynamic Flows*, Paper 92-0439, AIAA (1992).

[8] L. Davidsson, *Fluid mechanics, turbulent flow and turbulence modeling*, Course material in MSc courses, Division of Fluid Dynamics, Dept. of Applied Mechanics, Chalmers University of Technology (2013).

[9] J. Ferziger, M. Peric, *Computational Methods for Fluid Dynamics*, 3rd Edition, Springer, Berlin, 2002.

[10] K. Kozel, P. Louda, J. Prihoda, *Numerical Solution of an Unsteady Flow Using Artificial Compressibility Method*, in: Proceedings of the Czech-Japanese Seminar in Applied Mathematics, Czech Technical University, 2006, pp. 148–155.

[11] C. J. Greenshields, H. G. Weller, L. Gasparini, R. J. M, *Implementation of semi-discrete, non-staggered central schemes in a colocated, polyhedral, finite volume framework, for high-speed viscous flows*, Int. J. Numer. Meth. Fluids 63 (2010) 1–21.

[12] S. Popinet, *Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries*, J. Comp. Phys. 190 (2003) 572–600.

[13] R. J. Leveque, *Finite Volume Methods for Hyperbolic Problems*, 1st Edition, Cambridge University Press, Cambridge, 2002.

[14] M. S. Darwish, F. Moukalled, *TVD schemes for unstructured grids*, International Journal of Heat and Mass Transfer 46 (2003) 599–611.

[15] V. Venkatakrishnan, *Convergence to steady state solutions of the Euler equations on unstructured grids with limiters*, J. Comp. Phys. 118 (1995) 120–130.

[16] A. Kurganov, E. Tadmor, *New High-Resolution Central Schemes for Nonlinear Conservation Laws and Convection–Diffusion Equations*, J. Comp. Phys. 160 (2000) 241–282.

[17] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, 2003.

[18] S. Yoon, A. Jameson, *Lower-Upper Symmetric-Gauss-Seidel method for the Euler and Navier-Stokes equations*, AIAA Journal 26 (9).

[19] P. E. Farrel, J. R. Maddison, *Conservative intepolation between volume meshes by local Galerkin projection*, Computational Methods for Applied Mechanical Engineering 200 (2011) 89–100.

[20] aerospacedesignlab (2013), *Configuration file*, downloaded: (6-11-2013).
URL `http://adl.stanford.edu/docs/display/SUSQUARED/Configuration+file`

[21] T. von Karman, *Turbulence and Skin Friction*, J. of the Aeronautical Sciences 1 (1) (1934) 1–20.

A

Appendix

A.1 Taylor-Couette

Taylor-Couette flow is a flow between two cylinders. The inner radius will be denoted as r_1 and the outer radius as r_2. Assuming constant viscosity, incompressible flow, axisymmetrical flow and that $u_r = 0$ and $u_z = 0$. Navier-Stokes equation in cylindrical coordinates can be reduced to,

$$\begin{aligned}\frac{u_\theta^2}{r} &= \frac{\partial p}{\partial r}\\ \frac{u_\theta}{r} &= \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u_\theta}{\partial r}\right)\end{aligned} \tag{A.1}$$

The pressure p is not the physical pressure but the physical pressure wheighted with $frac1\rho$. Expanding the second equation of (A.1),

$$r^2\frac{\partial^2 u_\theta}{\partial r^2} + r\frac{\partial u_\theta}{\partial r} - u_\theta = 0 \tag{A.2}$$

and set the solution as $u_\theta = r^\gamma$, the following solution is found if Dirichlet boundary conditions are applied to the boundaries $u_\theta(r_1) = u_1$ and $u_\theta(r_2) = u_2$.

$$\begin{aligned}u_\theta =& Ar + \frac{B}{r}\\ A =& \frac{(u_2 r_2 - u_1 r_1)}{r_2^2 - r_1^2}\\ B =& u_1 r_1 - A r_1^2\end{aligned} \tag{A.3}$$

For the pressure the first equation of (A.1) is integrated and the following equation is found,

$$p = \frac{A^2 r^2}{2} + 2AB\log(r) - \frac{B^2}{2r^2} + C. \tag{A.4}$$

C is a constant which physically is of no importance as only pressure differences are of importance in the incompressible formulation.

A.1.1 Taylor-Couette OpenFOAM MRF files

For *system/fvScheme* this is the file used in the Taylor-Couette case.

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
    grad(U)         Gauss linear 1;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(HbyA) linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    p               ;
}
```

For *system/fvSolution* the following file is specified in the MRF case.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration off;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels      1;

        tolerance        1e-12;
        relTol           0;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-12;
        relTol          0;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 2;
    pRefCell        0;
    pRefValue       0;
    residualControl
        {
            p               1e-11;
            U               1e-11;
            nuTilda         1e-12;
        }
}

relaxationFactors
{
    fields
    {
        p               0.3;
    }
    equations
    {
        U               0.5;
    }
}
```

A.1.2 Taylor-Couette OpenFOAM AMI files

For the AMI case the following schemes and discretizations are made. In *system/fvScheme* the following file is used,

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
    grad(U)         Gauss linear 1;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);
//    div(phi,U)      Gauss upwind;
    div((nuEff*dev(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(HbyA) linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    pcorr           ;
    p               ;
}
```

The following schemes were used for the case,

```
 FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    pcorr
    {
        solver          GAMG;
        smoother        GaussSeidel;
        nPreSweeps      0;
        nPostSweeps     2;
        cacheAgglomeration off;
        agglomerator    faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels      1;

        tolerance        1e-10;
        relTol           0;
    }

    p
    {
        $pcorr;
        tolerance       1e-10;
        relTol          0;
    }

    pFinal
    {
        $p;
        tolerance       1e-10;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }

    UFinal
    {
        $U;
        tolerance       1e-8;
        relTol          0;
    }

}

PIMPLE
{
    correctPhi          no;
    nOuterCorrectors    1;
    nCorrectors         2;
    nNonOrthogonalCorrectors 4;

    pRefCell            0;
    pRefValue           0;
}

SIMPLE
{
    nNonOrthogonalCorrectors 2;
    pRefCell        0;
    pRefValue       0;
}

relaxationFactors
{
    fields
    {
        p               0.3;
    }
    equations
    {
        U               0.5;
    }
}
```

A.1.3 Taylor-Couette Gerris file

The following *.gfs* file is used to run the Taylor-Couette case. Every 5000s of the simulation a VTK file is written with the solution.

```
1 0 GfsSimulationMoving GfsBox GfsGEdge {} {
  GfsTime { end = 50000 }
  Refine 7
  GfsRefineSolid 9
  PhysicalParams { L = 2.2 }
  SolidMoving cylinder.gts {scale = 1.4} {level = 10}
  Solid (ellipse (0,0,1.0,1.0)) {flip = 1}
  ApproxProjectionParams { tolerance = 1e-9 }
  AdvectionParams { scheme = godunov
                    moving_order = 1
    }
  SourceViscosity {} 1e-5

  SurfaceBc U Dirichlet (x*x + y*y > 0.4 ? 0. : - 1e-3*y)
  SurfaceBc V Dirichlet (x*x + y*y > 0.4 ? 0. :   1e-3*x)
  OutputSimulation { step = 5000 } result%f.vtk {format = VTK}
  OutputSimulation { start = end } result-end.vtk {format = VTK}
}
GfsBox {}
```