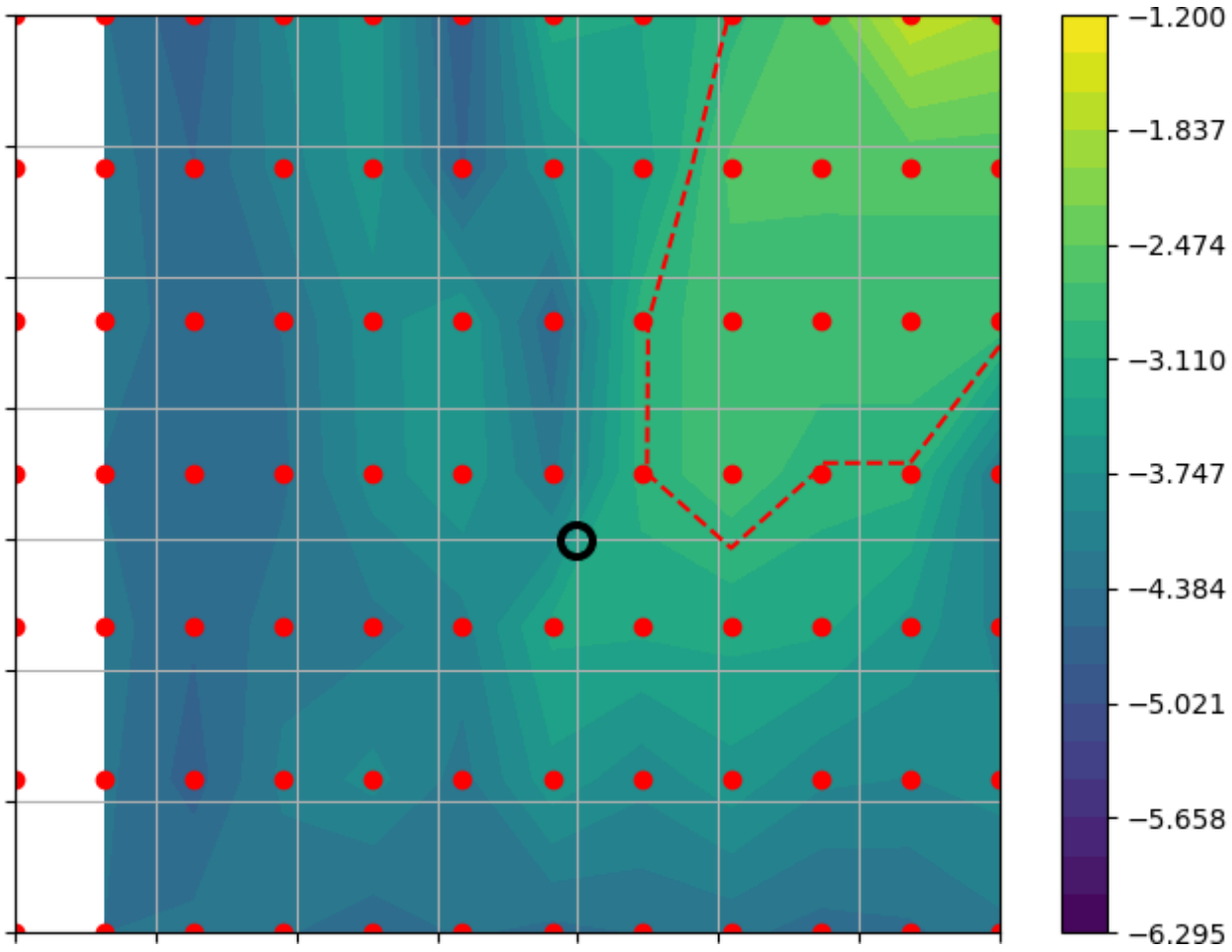



# Evaluation and development of an in-house GPU based CFD solver

Master's thesis in Applied Mechanics

Isak Lundgren

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

# Evaluation and development of an in-house GPU based CFD solver

Isak Lundgren


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
*Division of Fluid Dynamics*
Chalmers University of Technology
Gothenburg, Sweden 2024

Evaluation and development of an
in-house GPU based CFD solver
Isak Lundgren



Supervisor: Mattia Ricchi, GKN Aerospace Sweden AB
Examiner: Niklas Andersson, Department of Mechanics and Maritime Sciences

Master's Thesis 2024
Department of Mechanics and Maritime Sciences
Division of Fluid Dynamics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Contour plot of solution error relative converged state, logarithmic scale.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Evaluation and development of an in-house GPU based CFD solver

Isak Lundgren
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

# Abstract

An in-house GPU based CFD solver has been validated with respect to total pressure loss on a turbine rear structure compared to results from ANSYS Fluent and the previous version of the code itself. The order of the normalized error was found to be $\mathcal{O}(10^{-2})$ and $\mathcal{O}(10^{-4})$ respectively. The solver has also been analyzed with respect to its code structure, finding it made up of three parts: a Python part, a C part and a CUDA C++ part. A critical memory error was found in the C part which pronounces from seemingly trivial source code changes. It does not present as a problem, however, if these trivial source code changes are reversed. Previous investigations on the solver found incorrect total pressure readings on the domain inlet, which were located and discussed in the source code. Finally, asymptotic convergence criteria were added and their settings optimized to give a compromise between solver speed and accuracy for a generic turbine rear structure simulation.

Keywords: Validation, G3D, CFD, Compressible, Explicit, Runge-Kutta, CUDA, C/C++, Python, Convergence Criteria

# Acknowledgements

This thesis has been conducted in cooperation with GKN Aerospace Sweden AB and I would like to thank everyone working with me for this opportunity and making me feel welcome. I thank my supervisor, Mattia Ricchi, who guided me through all the GKN administration and theoretical knowledge on TRS designing. I thank my informal supervisor, Jonas Larsson, supplying me with expert knowledge on both TRS designs and fundamental CFD. I thank Ola Embreus who helped me in difficult programming situations, had great technical discussions and prevented me from falling into a dead end with a simple comment on a paper. I would also like to thank Mattias Billson and Marcus Lejon who gave both technical advice and suggestions to further the thesis.

I would like to thank my examiner, Niklas Andersson from Chalmers University of Technology, who went far beyond his assigned duty, supplying me with technical knowledge and guidance.

Finally, I'd also like to thank my fellow thesis students here at GKN for some good discussions on entropy.

Isak Lundgren, Trollhättan, May 2024

# List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

| | |
|---|---|
| ADP | Aerodynamic Design Point |
| ADPL | Adjusted Difference in total Pressure Loss |
| CFD | Computational Fluid Dynamics |
| CFL | Courant-Friedrich-Levy number |
| CUDA | Compute Unified Device Architecture |
| FANS | Favre Averaged Navier-Stokes |
| GKN | GKN Aerospace Sweden AB |
| RANS | Reynolds Averaged Navier-Stokes |
| TRS | Turbine Rear Structure |

# Nomenclature

Below is the nomenclature of indices and variables that have been used throughout this thesis.

## Indices

| | |
|---|---|
| $i$,$j$,$k$ | Indices for cardinal direction, mesh position and summation |

## Variables

| | |
|---|---|
| $\Psi$ | Generic flow variable |
| $x_j$ | Spacial coordniate in cardinal direction $j$ |
| $t$ | Time |
| $\rho$ | Density |
| $u_j$ | Velocity in cardinal direction $j$ |
| $p$ | Static pressure |
| $\sigma_{ij}$ | Viscous stress at face $i$ with cardinal direction $j$ |
| $\tau_{ij}$ | Reynolds stress at face $i$ with cardinal direction $j$ |
| $e$ | Internal energy |
| $h$ | Enthalpy |
| $q_i$ | Heat flux in cardinal direction $i$ |
| $T$ | Static temperature |
| $C_p$ | Isobaric heat capacity |
| $\delta_{ij}$ | Kronecker delta |
| $Pr$ | Prantl number |
| $\mu$ | dynamic viscosity |
| $\mu_t$ | turbulent viscosity |
| $k$ | Turbulent kinetic energy or subsolution in Runge-Kutta scheme |
| $\epsilon$ | Turbulent dissipation rate |

| | |
|---|---|
| $C_\Psi$ | Constant pertaining to flow variable $\Psi$ |
| $S$ | Strain rate tensor or surface area normal vector or conservative source term vector |
| $\Omega$ | Vorticity tensor or control volume |
| $\lambda$ | Eigenvalue |
| $\kappa$ | von Kármán constant |
| $Q$ | Conservative variables vector |
| $F$ | Conservative Flux tensor |
| $r$ | Characteristic variable |
| $A_{\text{Criterion}}$ | Amplitude criteron |
| $L_{\text{Criterion}}$ | Lean criteron |
| $eval$ | Evaluation length in number of iterations |

# Contents

# List of Figures

# List of Tables

# 1
# Introduction

## 1.1 Background

A primary component of developing an engine of any kind is a rigorous understanding of the flow development inside it. In modern industry, the dominating method for achieving such an understanding is by using computational fluid dynamics (CFD) to solve the Reynolds-Averaged Navier-Stokes (RANS) equations [1], with turbulence models to capture the effects of turbulence on the flow. In cooperation with GKN Aerospace Sweden AB, henceforth referred to as GKN, this thesis aims to validate and maintain an existing numerical solver developed for that purpouse.

The solver built at GKN, known as Trollsol, uses a 3-step Runge-Kutta time scheme method to find a steady-state solution to the compressible formulation of the RANS equations with a $k - \epsilon$ turbulence model. It is made to run on NVIDIA GPUs with their Compute Unified Device Architecture®(CUDA), which includes programming instructions and dedicated hardware to enable parallelization [2]. The usage of parallelization in computational physics, including fluid dynamics, have shown to be faster in a multitude of simulation setups [2–4]. It has furthermore been shown faster in explicit CFD codes [5].

As opposed to commercially available CFD solvers, Trollsol is specialized for integration with a aerodynamic design program called VolVane. VolVane is used to make designs for turbine rear structures (TRS), which have the main purposes of removing swirl from the outgoing low pressure turbine flow in a jet engine and providing said engine with an attachment point to the rest of the aircraft [6]. The vanes used in a TRS are classified as regular vanes, vanes with an engine-mount recess and vanes with increased thickness to allow for passage of oil tubes. Previous validation studies of Trollsol have only included regular vanes and tube vanes [7]. An overview of a TRS and its location in the engine is shown in figure 1.1.

Previous studies with pre-validated TRS designs in Trollsol showed that the solver adequately represents the flow behaviour compared to solutions produced with the commercial software ANSYS Fluent [7]. It was shown that the behavior of pressure loss at off-design inlet swirl angles, that is incoming flow angles that the vanes were not designed to operate at, matched that of the ANSYS Fluent predicted pressure loss, albeit with an offset factor unique to vane geometry. Related to this, it was also shown that total pressure did not match predictions from ANSYS Fluent, giving errors on the order of $10^2$ Pa. A test to mitigate these differences by matching the inlet total pressure was unsuccessful, leading to further speculation that the method of applying the boundary conditions was incorrect. In an internal GKN study, it

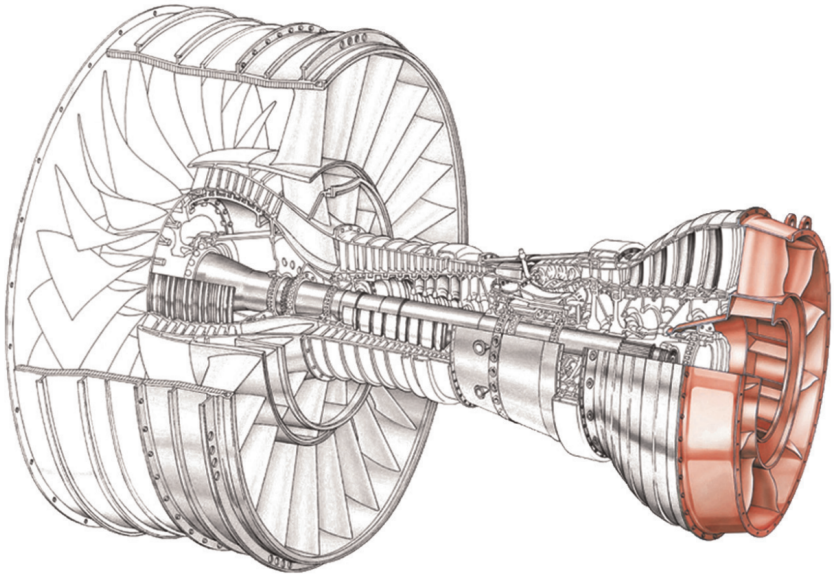


**Figure 1.1:** A high bypass ratio jet engine with the turbine rear structure highlighted in red [8].

was found that the inlet boundary conditions show after post-processing were not independent from the tangential coordinate, as was expected from the inlet profile, see figure 1.2. This problem was linked to how total pressure is extrapolated to the boundary, which is simply copying the value from the adjacent cell.

## 1.2 Purpose

The primary purpose of this thesis is to investigate the current state of Trollsol with regards to accuracy in predicting pressure loss and outlet radial swirl distribution on a TRS. Secondarily, the thesis aims to create a comprehensive documentation for Trollsol and further develop its capabilities.

## 1.3 Problem description

This thesis aims to expand knowledge about the solver in its current state, which is embodied by the following questions:

- How does the current version of Trollsol compare with previous versions?
- What is the structure of the source code and how does it relate to the governing equations?
- What is the cause of previous pressure loss errors?
- What can be further developed in Trollsol?

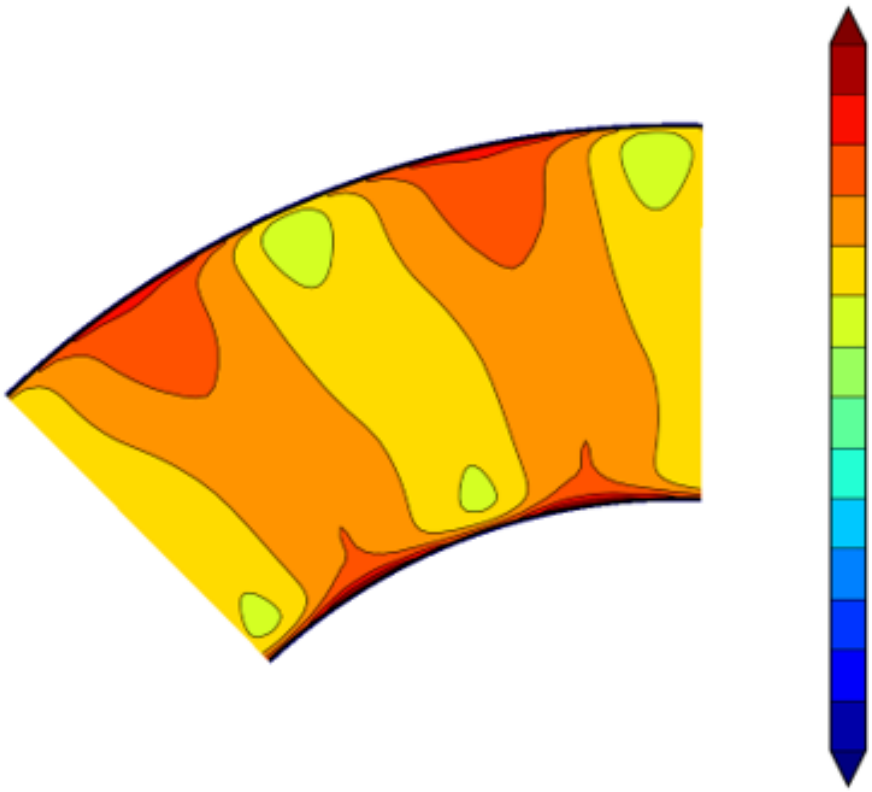


**Figure 1.2:** Total pressure on the inlet in previous cases run with Trollsol. The profile is expected to be uniform in the tangential direction. The fluctionations approximately reach 0.3% of total pressure average magnitude. Image taken from "Debugging inlet B.C in g3dcuda", internal report.

## 1.4 Limitations

As this thesis is conducted during 20 weeks, or 30 hp, a limitation on the scope is put on the work. This work will not include any simulation in ANSYS Fluent, opting instead to rely on previous simulations on the same geometry. As a consequence, the validation study will also be limited to regular vanes and oil tube vanes, as discussed in section 1.1.

On the prospects of which code should be looked at, only Trollsol itself along with its dependencies are studied. This means that tools utilizing Trollsol such as Volvane are excluded from this work, but libraries built at GKN which Trollsol uses are included.

# 2

# Governing equations

As opposed to the incompressible RANS equations, the compressible RANS equations or Favre-Averaged Navier-Stokes (FANS) equations utilize a density-weighted average to describe physical quantities [9]. With the Reynolds time-averaging as a reference in equation 2.1, the Favre time-averaging is described in equation 2.2. $\Psi$ in the equations refer to any continuous physical quantity and $\rho$ refers to the fluid density.

$$\begin{aligned} \overline{\Psi} =& \frac{1}{T}\int_T \Psi dt, \\ \Psi' =& \Psi - \overline{\Psi}, \end{aligned} \tag{2.1}$$

$$\begin{aligned} \widetilde{\Psi} =& \frac{\overline{\rho\Psi}}{\overline{\rho}}, \\ \Psi'' =& \Psi - \widetilde{\Psi}. \end{aligned} \tag{2.2}$$

Applying Favre averaging to the instantaneous Navier Stokes equations yields the FANS continuity equation, equation 2.3, the FANS conservation of momentum equation, equation 2.4 and the FANS conservation of energy equation, equation 2.5.

$$\frac{\partial\overline{\rho}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{\rho}\widetilde{u_j}) = 0, \tag{2.3}$$

$$\frac{\partial\overline{\rho}\widetilde{u_i}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{\rho}\widetilde{u_i}\widetilde{u_j}) = -\frac{\partial\overline{p}}{\partial x_i} + \frac{\partial\overline{\sigma_{ij}}}{\partial x_j} + \frac{\partial\tau_{ij}}{\partial x_j}, \tag{2.4}$$

$$\frac{\partial\overline{\rho}\widetilde{e}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{\rho}\widetilde{u_j}\widetilde{h}) = \frac{\partial}{\partial x_j}(\overline{\sigma_{ij}}\widetilde{u_i} + \overline{\sigma_{ij}u_i''}) - \frac{\partial}{\partial x_j}(\overline{q_j} + C_p\overline{\rho u_j''T''} - \widetilde{u_i}\tau_{ij} + \frac{1}{2}\overline{\rho u_i''u_i''u_j''}), \tag{2.5}$$

where the viscous stress $\overline{\sigma_{ij}}$, Reynolds stress $\tau_{ij}$, enthalpy $\widetilde{h}$ and heat flux $\overline{q_j}$ are described in equations 2.6, 2.7, 2.8 and 2.9 respectively. In said equations, $C_p$ is the isobaric heat capacity, $\delta_{ij}$ is the Kronecker delta, $Pr$ is the Prantl number and $\mu$ is the dynamic viscosity.

$$\overline{\sigma_{ij}} \approx 2\widetilde{\mu}\left[\frac{1}{2}\left(\frac{\partial\widetilde{u_i}}{\partial x_j} + \frac{\partial\widetilde{u_j}}{\partial x_i}\right) - \frac{1}{3}\frac{\partial\widetilde{u_k}}{\partial x_k}\delta_{ij}\right], \tag{2.6}$$

$$\tau_{ij} = -\overline{\rho u_i''u_j''}, \tag{2.7}$$

$$\widetilde{h} = \widetilde{e} + \overline{p}/\overline{\rho}, \tag{2.8}$$

$$\overline{q_j} \approx -\frac{C_p \widetilde{\mu}}{Pr} \frac{\partial \widetilde{T}}{\partial x_j}. \tag{2.9}$$

In this thesis the set turbulence model utilize the Boussinesq approximation which introduces a turbulent eddy viscosity $\mu_t$ and a turbulent kinetic energy $k$. It is an approximation for the Reynolds stress term and reads as equation 2.10.

$$\tau_{ij} = 2\widetilde{\mu_t}\left[\frac{1}{2}\left(\frac{\partial \widetilde{u_i}}{\partial x_j} + \frac{\partial \widetilde{u_j}}{\partial x_i}\right) - \frac{1}{3}\frac{\partial \widetilde{u_k}}{\partial x_k}\delta_{ij}\right] - \frac{2}{3}\overline{\rho} k \delta_{ij},\ k \equiv \frac{\widetilde{u_i''u_i''}}{2}. \tag{2.10}$$

The equation of state for the fluid is described in equation 2.11 and temperaure in equation 2.12.

$$\overline{p} = (\gamma - 1)[\overline{\rho}\widetilde{e}_0 - \frac{1}{2}\overline{\rho}\widetilde{u_i}\widetilde{u_i} - \overline{\rho}k], \tag{2.11}$$

$$\widetilde{T} = \frac{\widetilde{p}}{\overline{\rho}R} \tag{2.12}$$

where $\gamma$ is the heat capacity ratio.

As was done with the Boussinesq approximation, further terms need to be modeled in the energy equation to close the system of equations. This, since the fluctuating terms $\Phi''$ are not explicitly calculated in either turbulence model presented. The turbulent heat flux is modeled as a Reynolds analogy in equation 2.13 and the terms associated with molecular diffusion and turbulent transport in the energy equation are modeled together in equation 2.14. With these simplifications, a reformulation of the energy equation is presented in equation 2.15. $\sigma_k$ refers to a coefficient that is associated with the modeling equation for $k$ and $Pr_t$ is a turbulent Prantl number.

$$C_p\overline{\rho u_j'' T''} \approx -\frac{C_p\widetilde{\mu_t}}{Pr_t}\frac{\partial \widetilde{T}}{\partial x_j}, \tag{2.13}$$

$$\overline{\sigma_{ij}u_i''} - \frac{1}{2}\overline{\rho u_i'' u_i'' u_j''} \approx \left(\widetilde{\mu} + \frac{\widetilde{\mu_t}}{\sigma_k}\right)\frac{\partial k}{\partial x_j}, \tag{2.14}$$

$$\frac{\partial \overline{\rho}\widetilde{e}}{\partial t} + \frac{\partial}{\partial x_j}(\overline{\rho}\widetilde{u_j}\widetilde{h}) = \frac{\partial}{\partial x_j}\left[C_p\left(\frac{\widetilde{\mu}}{Pr} + \frac{\widetilde{\mu_t}}{Pr_t}\right)\frac{\partial}{\partial x_j}\left(\frac{\widetilde{p}}{\overline{\rho}}\right) + \widetilde{u_i}(\tau_{ij} + \overline{\sigma_{ij}}) + \left(\widetilde{\mu} + \frac{\widetilde{\mu_t}}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right]. \tag{2.15}$$

## 2.1 The realizable $k - \epsilon$ turbulence mode

Trollsol uses a high Reynolds number version of the $k - \epsilon$ model [10], meaning that wall damping functions are not involved in contrast to low Reynolds number $k - \epsilon$ models. It does however also use the Boussinesq approximation, equation 2.10, and

viscous diffusion in the turbulence transport equations, both of which are used by other $k-\epsilon$ models [11, 12].

The transport equations for $k$ and $\epsilon$ are described in equations 2.16 and 2.17 respectively. The production term $P_k$ is described in equation 2.18 and the turbulent viscosity is described in equation 2.19. $C_\mu$, $C_{\epsilon 1}$, $C_{\epsilon 2}$, $\sigma_k$ and $\sigma_\epsilon$ are constants specified in table 2.1 and mimic those used in the Launder-Sharma model [12].

$$\frac{\partial \overline{\rho} k}{\partial t} + \frac{\partial \overline{\rho} \widetilde{u_j} k}{\partial x_j} = P_k - \rho\epsilon + \frac{\partial}{\partial x_j}\left[(\widetilde{\mu} + \frac{\widetilde{\mu_t}}{\sigma_k})\frac{\partial k}{\partial x_j}\right], \tag{2.16}$$

$$\frac{\partial \overline{\rho} \epsilon}{\partial t} + \frac{\partial \overline{\rho} \widetilde{u_j} \epsilon}{\partial x_j} = C_{\epsilon 1}\frac{\epsilon}{k}P_k - C_{\epsilon 2}\frac{\overline{\rho}\epsilon^2}{k} + \frac{\partial}{\partial x_j}\left[(\widetilde{\mu} + \frac{\widetilde{\mu_t}}{\sigma_\epsilon})\frac{\partial \epsilon}{\partial x_j}\right], \tag{2.17}$$

$$P_k = \tau_{ij}\frac{\partial u_i}{\partial x_j}, \tag{2.18}$$

$$\mu_t = C_\mu \frac{\overline{\rho} k^2}{\epsilon}. \tag{2.19}$$

**Table 2.1:** Constants and their corresponding values or expressions pertaining to the high Reynolds number $k-\epsilon$ turbulence model. Constants mimic the Launder-Sharma model [12].

| **Constant** | **Value / Expression** |
|---|---|
| $C_\mu$ | 0.09 |
| $C_{\epsilon 1}$ | 1.44 |
| $C_{\epsilon 2}$ | 1.92 |
| $\sigma_k$ | 1 |
| $\sigma_\epsilon$ | 1.3 |

### 2.1.1 Turbulent viscosity limitation

In points where the flow stagnates, the $k-\epsilon$ model has an issue with excessive production of turbulent kinetic energy compared to empirical data [13]. This results in a high turbulent viscosity, which can be limited with a realizability constraint. The constraint is presented in equation 2.20 [14].

$$\overline{u_1'' u_1''},\ \overline{u_2'' u_2''},\ \overline{u_3'' u_3''} \geq 0 \tag{2.20}$$

In principal-axes coordinates, that is a coordinate system where strain only occurs in the three principle directions, the strain rate tensor is diagonal with its eigenvalues as diagonal elements. This means that the strain rate magnitude $\mathbf{S}$ squared is equal to the sum of squares of eigenvalues and the trace of the strain rate tensor is the sum of the eigenvalues, equation 2.21,

$$S_{ij}S_{ij} = \mathbf{S}^2 = \lambda_1^2 + \lambda_2^2 + \lambda_3^2,\ tr(S_{ij}) = \lambda_1 + \lambda_2 + \lambda_3. \tag{2.21}$$

Assuming incompressible flow[1], meaning $tr(S_{ij}) = 0$, it can be shown that the eigenvalues are as described in equation 2.22,

$$|\lambda_\alpha| \leq \sqrt{2\mathbf{S}^2/3}. \tag{2.22}$$

Since the strain rate magnitude is invariant of the coordinate system, it can be computed without a change of coordinate system. Combining equations 2.7 and 2.10, writing in principal axes of the strain rate tensor ($S_{\alpha\alpha} = \lambda_\alpha$) and assuming incompressible flow conditions ($\frac{\partial u_k}{\partial x_k} = 0$), the relation in equation 2.23 is produced.

$$-\overline{\rho u''_\alpha u''_\alpha} = 2\widetilde{\mu_t}\lambda_\alpha - \frac{2}{3}\overline{\rho}k. \tag{2.23}$$

Combining equation 2.20 and equation 2.23 while assuming density is constant yields equation 2.24.

$$2\widetilde{\mu_t}\lambda_\alpha \leq \frac{2}{3}\overline{\rho}k. \tag{2.24}$$

Changing the relation in equation 2.22 from less than or equal to to an equals sign [15], then inserting said relation into equation 2.24, a statement for the limitation of turbulent viscosity is produced in equation 2.25.

$$\mu_t \leq \frac{1}{\sqrt{6}}\frac{\overline{\rho}k}{\mathbf{S}}. \tag{2.25}$$

### 2.1.2 Kato-Launder modification

Similar to the problem discussed in section 2.1.1, the Kato-Launder modification [13] instead substitutes the production expression with one that is proportional to vorticity magnitude, equation 2.26, to reduce the growth of turbulent kinetic energy near stagnation. The argument used for this modification is that the flow near stagnation is almost irrotational, while the vorticity magnitude is identical to the shear stress magnitude in pure shear flow.

$$P_k = C_\mu \epsilon \mathbf{S}\mathbf{\Omega}, \tag{2.26}$$

where the vorticity magnitude $\mathbf{\Omega}$ is described in equation 2.27.

$$\mathbf{\Omega} \equiv \sqrt{\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right)^2}. \tag{2.27}$$

## 2.2 Wall functions

To alleviate the use of computer resources, the turbulent boundary layer which forms near walls can be modeled instead of resolved. This is done through the so called wall functions, which assume that the flow near the wall behaves like a

[1]Only the limiter on turbulent viscosity assumes that the flow is incompressible. The use of the limiter is then extended to compressible flow cases with the assumption of applicability.

fully developed boundary layer and uses empirical correlations [15]. For the wall functions, the correlation used is the log-law, which describes the relation between the dimensionless constants $u^+$ and $y^+$ in the approximate region $30 < y^+ < 200$.[2] The relation is described in equation 2.28, where $\kappa$ is the von Kármán constant equal to 0.40, and B is an integration constant fitted emprically equal to 4.9. The dimensionless constants are described in equation 2.29, where $U$ is the tangential velocity relative to the wall, $\rho$ is the density, $\tau_w$ is the wall shear stress, $\delta y$ is the normal distance from the wall to the wall adjacent cell center and $\mu$ is the dynamic viscosity.

$$u^+ = \frac{1}{\kappa} \ln (y^+) + B, \tag{2.28}$$

$$u^+ = U\sqrt{\frac{\rho}{\tau_w}},\ y^+ = \frac{\delta y \sqrt{\rho \tau_w}}{\mu}. \tag{2.29}$$

Combining equations 2.28 and 2.29, the single unknown $\tau_w$ is found iteratively through the Newton-Raphson method. It is then used as a flux boundary condition to the momentum equations, see section 3.5.

[2]The region in which the log-law is applicable is not consistent between sources, but most claim a lower bound of 30.

# 3
# Numerical methods

Trollsol is a structured explicit coupled solver: structured meaning that the solution grid is ordered and every cell is hexahedral; explicit meaning that an iteration of the solver variables are calculated through a formula strictly dependent on sovler variables from the previous iteration; and coupled meaning that all solver variables are solved for simultaneously. These schemes are made through discretizations in both space and time of the compact conservative formulations of the governing equations, equation 3.1. This derivation is based on the works of Larsson [16].

$$\frac{\partial Q}{\partial t} + \frac{\partial F_j}{\partial x_j} = S, \tag{3.1}$$

where the tensors $Q$, $F_j$ and $S$ are described by equation 3.2,

$$Q = \begin{bmatrix} \rho \\ \rho u_i \\ \rho e_0 \\ \rho k \\ \rho \epsilon \end{bmatrix}, \; F_j = \begin{bmatrix} \rho u_j \\ \rho u_i u_j + p\delta_{ij} - \tau_{ij}^{\text{tot}} \\ (\rho e_0 + p)u_j + q_j - u_i \tau_{ij}^{\text{tot}} \\ \rho k u_j + d_j^{(k)} \\ \rho \epsilon u_j + d_j^{(\epsilon)} \end{bmatrix}, \; S = \begin{bmatrix} 0 \\ 0 \\ 0 \\ s^{(k)} \\ s^{(\epsilon)} \end{bmatrix}, \tag{3.2}$$

where $\tau_{ij}^{\text{tot}}$ is the sum of the viscous shear stress, equation 2.6, and the Reynolds stress, equation 2.10. The heat flux vector $q_j$ is described in equation 3.3 and the turbulent diffusion and source terms are described in equations 3.4,

$$q_j = -C_p \left( \frac{\widetilde{\mu}}{Pr} + \frac{\widetilde{\mu_t}}{Pr_t} \right) \frac{\partial}{\partial x_j} \left( \frac{\widetilde{p}}{\overline{\rho}} \right) - \left( \widetilde{\mu} + \frac{\widetilde{\mu_t}}{\sigma_k} \right) \frac{\partial k}{\partial x_j}, \tag{3.3}$$

$$\begin{aligned} d^{(k)} &= -\frac{\partial}{\partial x_j} \left[ \left( \mu + \frac{\mu_t}{\sigma_k} \right) \frac{\partial k}{\partial x_j} \right], \\ d^{(\epsilon)} &= -\frac{\partial}{\partial x_j} \left[ \left( \mu + \frac{\mu_t}{\sigma_\epsilon} \right) \frac{\partial \epsilon}{\partial x_j} \right], \\ s^{(k)} &= P_k - \rho\epsilon, \\ s^{(\epsilon)} &= C_{\epsilon 1} \frac{\epsilon}{k} P_k - C_{\epsilon 2} \frac{\rho \epsilon^2}{k}, \end{aligned} \tag{3.4}$$

To discretize the scheme in space, equation 3.1 is integrated over a control volume $\Omega$, resulting in equation 3.5. If the control volume is time-independent, the Gauss' theorem can be used to rewrite the expression as equation 3.6.

$$\int_\Omega \frac{\partial Q}{\partial t} dV + \int_\Omega \frac{\partial F_j}{\partial x_j} dV = \int_\Omega S dV, \tag{3.5}$$

$$V \frac{d}{dt} \overline{Q} + \int_{\partial\Omega} F_j \cdot d\mathcal{S}_j = V\overline{S}, \tag{3.6}$$

where $\overline{Q}$ and $\overline{S}$ are the volume averages of $Q$ and $S$ respectively on the control volume, $V$ is the volume of the control volume and $d\mathcal{S}_j$ is a surface normal vector. Given a cubic control volume, if the flux vector is assumed to be constant on each cube face, the second term in equation 3.6 can be rewritten as equation 3.7, leaving the final discretized governing equations as equation 3.8.

$$\sum_{\text{faces}} F_j^{\text{face}} \cdot \mathcal{S}_j^{\text{face}}, \tag{3.7}$$

where $F_j^{\text{face}}$ is the flux vector averaged on a control volume face and $\mathcal{S}_j^{\text{face}}$ is a surface normal vector with length equal to the area of the face.

$$V \frac{d}{dt} \overline{Q} + \sum_{\text{faces}} F_j^{\text{face}} \cdot \mathcal{S}_j^{\text{face}} = V\overline{S}. \tag{3.8}$$

## 3.1 Explicit Runge-Kutta time stepping

The discretization of the governing equations converts the partial differential equations to a system of ODE:s or an initial value problem, classified by that the derivative of a solution vector is equal to a function of said function vector, along with an initial solution vector for the initial time step [17]. Rearranging equation 3.8 and identifing key features of the expression enables a clear formulation of the initial value problem, equation 3.9.

$$\mathbf{y}(t + \Delta t) = \mathbf{y}(t) + \int_t^{t+\Delta t} f(\mathbf{y}(\tau)) d\tau, \tag{3.9}$$

where $\mathbf{y}$ and $f$ are identified in equation 3.10,

$$\mathbf{y} = \overline{Q},\ f(\mathbf{y}) = \overline{S} - \frac{1}{V} \sum_{\text{faces}} F_j^{\text{face}} \cdot \mathcal{S}_j^{\text{face}}. \tag{3.10}$$

The three stage explicit Runge-Kutta scheme then details how the next time step is reached, according to equation 3.11.

$$\begin{aligned} \mathbf{k}_1 &= f(\mathbf{y}_n), \\ \mathbf{k}_2 &= f(\mathbf{y}_n + \Delta t a_{21} \mathbf{k}_1), \\ \mathbf{k}_3 &= f(\mathbf{y}_n + \Delta t (a_{31} \mathbf{k}_1 + a_{32} \mathbf{k}_2)), \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \Delta t (b_1 \mathbf{k}_1 + b_2 \mathbf{k}_2 + b_3 \mathbf{k}_3), \end{aligned} \tag{3.11}$$

where $\mathbf{k}_i$ is subsolution i in the scheme. The Runge-Kutta matrix $a_{ij}$ and the weights vector $b_i$ are described in table 3.1.

**Table 3.1:** Butcher tableu for the Runge-Kutta scheme implemented in Trollsol. The top left quadrant represents nodes not applicable in this case, the top right represents the Runge-Kutta matrix $a_{ij}$ and the bottom right represents the weights $b_i$.

| | | | |
|---|---|---|---|
| – | | | |
| – | 1 | | |
| – | 1/2 | 1/2 | |
| | 1/2 | 0 | 1/2 |

## 3.2 Courant-Friedrich-Levy condition

The Courant-Friedrich-Levy condition, or the CFL number, is a condition important for describing stability in hyperbolic equations [18]. It is defined in equation 3.12 and describes a relation between a length scale $\Delta x$ and a characteristic length scale $c\Delta t$ where $c$ is a characteristic velocity. In terms of a compressible CFD solver, $c$ are defined velocities corresponding to wave propagation speed in the fluid and diffusive propagation speed. The CFL number when doing steady-state simulations is used to find a local timestep $\Delta t$ for each cell of which to propogate the numerical solution with. A standard explicit time stepping scheme for compressible flow equations requires CFL to be below 1 in all control volumes.

$$\mathrm{CFL} = \frac{c\Delta t}{\Delta x}. \tag{3.12}$$

## 3.3 Upwinding schemes

In accordance with equation 3.8, the value of the flux vector has to be evaluated at all the faces of each control volume. Trollsol uses an upwind scheme to calculate these fluxes based on characteristic variables in the flow. These characteristic variables are presented in equation 3.13. A detailed derivation of these variables can be found in a lecture by Lars-Erik Eriksson [19], but the underlying idea is to approximate the inviscid part of the flux vector $F$ as a product between the spacial derivative of the conservative variables $Q$ and a so called "flux Jacobian". Said approximation is then transformed from conservative variables to primitive variables, which is then transformed to characteristic variables based on the eigenvectors to the flux Jacobian. The reason why this transformation to characteristic variables is done is that it ensures interpolation, such as basing a face value on adjacent cell values, happens in accordance with the governing equations.

$$
\begin{aligned}
r^{(1)} &= \rho - \frac{p}{(c^f)^2}, \\
r^{(2)} &= \frac{-S_2 u_1 + S_1 u_2}{\sqrt{S_1^2 + S_2^2}}, \\
r^{(3)} &= \frac{-S_3(S_1 u_1 + S_2 u_2) + (S_1^2 + S_2^2)u_3}{\sqrt{S_1^2 + S_2^2}\sqrt{S_1^2 + S_2^2 + S_3^2}}, \\
r^{(4)} &= \frac{\rho^f}{2c^f}\frac{S_1 u_1 + S_2 u_2 + S_3 u_3}{\sqrt{S_1^2 + S_2^2 + S_3^2}} + \frac{p}{2(c^f)^2}, \\
r^{(5)} &= -\frac{\rho^f}{2c^f}\frac{S_1 u_1 + S_2 u_2 + S_3 u_3}{\sqrt{S_1^2 + S_2^2 + S_3^2}} + \frac{p}{2(c^f)^2}, \\
r^{(6)} &= k, \\
r^{(7)} &= \epsilon,
\end{aligned}
\tag{3.13}
$$

where $c$ is the speed of sound, $\rho$ is density, $u_1$, $u_2$, $u_3$ are flow velocities in the cardinal directions, $p$ is static pressure, $k$ is turbulent kinetic energy and $\epsilon$ is turbulent dissipation rate. $S_1$, $S_2$ and $S_3$ are vector components of the face area normal vector. Any quantity marked as $\Psi^f$ is a face value approximated as the mean of the two cells adjacent to the face, otherwise a quantity is the value at a cell center.

Upwinding is done on the four closest cells to the face, as seen in figure 3.1, with interpolation coefficients $C_n$ that can be set at runtime. The equation for getting the face value characteristic is shown in equation 3.14, which is based on the sign of the eigenvalues $\lambda_n$ to the flux Jacobian.

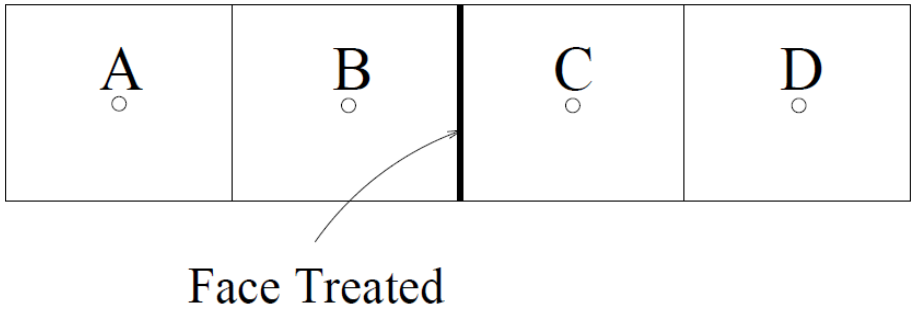



**Figure 3.1:** Four closest cells to a face that will be interpolated to. Image taken from [16].

$$
r_{face}^{(n)} = \begin{cases} C_1 r_A^{(n)} + C_2 r_B^{(n)} + C_3 r_C^{(n)} + C_4 r_D^{(n)}, & \text{if } \lambda_n \geq 0 \\ C_4 r_A^{(n)} + C_3 r_B^{(n)} + C_2 r_C^{(n)} + C_1 r_D^{(n)}, & \text{if } \lambda_n < 0 \end{cases},
\tag{3.14}
$$

where the subscripts $A$, $B$, $C$ and $D$ correspond to the cells in figure 3.1. The characteristic variables are then transformed back to primitive variables as show in equation 3.15.

$$
\begin{aligned}
\rho_{face} &= r^{(1)}_{face} + r^{(4)}_{face} + r^{(5)}_{face}, \\
u_{1,face} &= -\frac{S_2 r^{(2)}_{face}}{\sqrt{S_1^2 + S_2^2}} - \frac{S_1 S_3 r^{(3)}_{face}}{\sqrt{S_1^2 + S_2^2}\sqrt{S_1^2 + S_2^2 + S_3^2}} + \frac{c^f}{\rho^f}\frac{S_1\left(r^{(4)}_{face} - r^{(5)}_{face}\right)}{\sqrt{S_1^2 + S_2^2 + S_3^2}}, \\
u_{2,face} &= +\frac{S_1 r^{(2)}_{face}}{\sqrt{S_1^2 + S_2^2}} - \frac{S_2 S_3 r^{(3)}_{face}}{\sqrt{S_1^2 + S_2^2}\sqrt{S_1^2 + S_2^2 + S_3^2}} + \frac{c^f}{\rho^f}\frac{S_2\left(r^{(4)}_{face} - r^{(5)}_{face}\right)}{\sqrt{S_1^2 + S_2^2 + S_3^2}}, \\
u_{3,face} &= \frac{r^{(3)}_{face}\sqrt{S_1^2 + S_2^2}}{\sqrt{S_1^2 + S_2^2 + S_3^2}} + \frac{c^f}{\rho^f}\frac{S_3\left(r^{(4)}_{face} - r^{(5)}_{face}\right)}{\sqrt{S_1^2 + S_2^2 + S_3^2}}, \\
p_{face} &= (c^f)^2\left(r^{(4)}_{face} + r^{(5)}_{face}\right), \\
k_{face} &= r^{(6)}_{face}, \\
\epsilon_{face} &= r^{(7)}_{face},
\end{aligned}
\tag{3.15}
$$

where $\Psi_{face}$ is a variable at the cell face treated with the upwinding scheme.

## 3.4 Finite volume differentiation

To calculate the diffusive fluxes, both the value and the gradient of the primitive variables at the face are needed. In this case the value is simply approximated as the average of the two neighboring cells. For the gradients, however, a finite volume differentiation is used. In figure 3.2, a new control volume $\Omega'$ is introduced, which extrudes the treated face to the neighboring cell centers.

Assuming that the gradient is constant in $\Omega'$, it can be calculated with Gauss' theorem as in equation 3.16.

$$
\left(\frac{\partial \Psi}{\partial x_i}\right)^{face} = \frac{1}{V}\int_{\Omega'}\frac{\partial \Psi}{\partial x_i}dV = \frac{1}{V}\int_{\partial\Omega'}\Psi \cdot dS_i = \frac{1}{V}\sum_{\Omega' faces}\Psi^{\Omega' face}\cdot S_i^{\Omega' face}, \tag{3.16}
$$

where $V$ is the volume of $\Omega'$ and $S_i^{\Omega' face}$ are the face area vectors to $\Omega'$.

## 3.5 Wall functions

To calculate the wall shear stress from wall functions as described in section 2.2, a Newton Raphson method is done on equation 2.28. In Trollsol, the first guess for $\tau_w$ is set according to the linear law, equation 3.17.

$$
\tau_w = \mu_w \frac{u_{\text{tangential}}}{y_w}, \tag{3.17}
$$

where $\tau_w$ and $\mu_w$ are the shear stress and dynamic viscosity at the wall respectively, $u_{\text{tangential}}$ is the fluid velocity tangential to the wall and $y_w$ is the distance from the wall to the wall-adjacent cell center. Next, $y^+$ and $u^+$ are calculated as

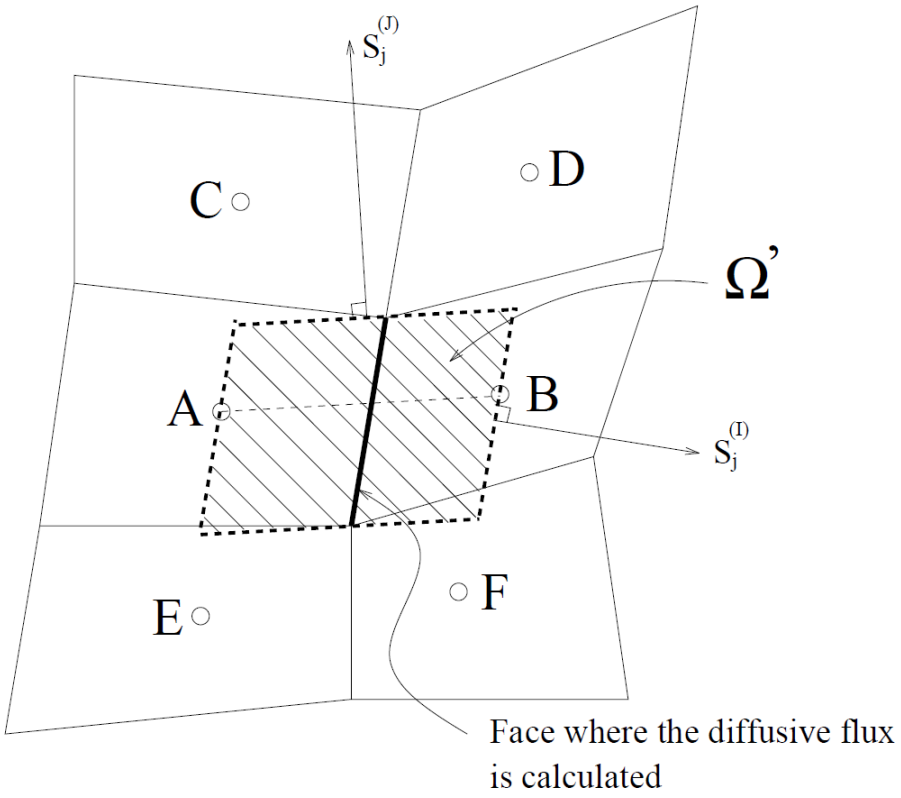



**Figure 3.2:** 2D schematic of the finite volume differentiation method. Image taken from [16].

in equation 2.29. If this preliminary calculation results in a $y^+$ value greater than 10.67848, an engieneering limit for where to switch to a log-law model, a function based on a rewrite of equation 2.28, which is presented in equation 3.18, is solved for a value of zero with respect to wall shear stress.

$$f(\tau_w) = \frac{1}{\kappa}\log\left(y^+(\tau_w)\right) + B - u^+(\tau_w), \tag{3.18}$$

where $y^+(\tau_w)$ and $u^+(\tau_w)$, again, are calculated as functions of $\tau_w$ in equation 2.29. This function describing the log-law is then solved in a Newton Raphson manner, described by the following process:

1. The value of function $f$ for the previous $\tau_w$ is evaluated.
2. The first derivative of function $f$, which is found analytically assuming $y^+$ and $u^+$ are constants, for the previous $\tau_w$ is evaluated.
3. The expression $-\frac{f}{\frac{\partial f}{\partial \tau_w}}$ is added $\tau_w$, replacing the previous one.
4. $y^+$ and $u^+$ are updated as in equation 2.29 with the new $\tau_w$.
5. If the absolute value of $\frac{\partial f}{\partial \tau_w}$ is larger than 0.01% of $\tau_w$, the process is repeated, otherwise a solution has been found.

After a value of $\tau_w$ has been found, it is added as a diffusive flux ($DF$) to the flux-face area multiplication vector $F_j^{\text{face}} \cdot \mathcal{S}_j^{\text{face}}$ on the wall, as described by equation 3.19

$$DF = \begin{bmatrix} 0 \\ \tau_w|\mathcal{S}|\frac{UPR}{U_{\text{tangential}}} \\ \tau_w|\mathcal{S}|\frac{VPR}{U_{\text{tangential}}} \\ \tau_w|\mathcal{S}|\frac{WPR}{U_{\text{tangential}}} \\ DF_2 \cdot \text{Wall}_u + DF_3 \cdot \text{Wall}_v + DF_4 \cdot \text{Wall}_w \\ 0 \\ 0 \end{bmatrix} \tag{3.19}$$

where $\text{Wall}_{u,v,w}$ is the speed of the wall in reference to the still reference frame, and U,V,WPR are modified versions of fluid velocity at the wall relative to the wall, equation 3.20.

$$UPR_i = u_{i,\text{Wall adjacent cell}} - \frac{u_{j,\text{Wall adjacent cell}}\mathcal{S}_j^{\text{wall}}\mathcal{S}_i^{\text{wall}}}{|\mathcal{S}|^2} - \text{Wall}_{u_i}. \tag{3.20}$$

## 3.6 Boundary averaging schemes

The values of physical quantities are averaged in three separate ways: flux averaging, mass averaging and area averaging. Mass and area averaging involves calculating a physical quantity from the conservative fluxes (equation 3.2) on each cell face coinciding with the boundary and then averaging said values weighted with either mass flux or cell area, see equation 3.21.

$$\begin{aligned}
\Psi_{\text{Cell face},i} &= f(F_1, ..., F_7)_{\text{Cell face},i}, \\
\Psi_{\text{Mass average}} &= \frac{1}{(\rho|\mathbf{u}|)_{\text{Boundary}}} \sum_{i}^{\text{Boundary cell faces}} (\rho|\mathbf{u}|)_{\text{Cell face},i} \Psi_{\text{Cell face},i}, \\
\Psi_{\text{Area average}} &= \frac{1}{A_{\text{Boundary}}} \sum_{i}^{\text{Boundary cell faces}} A_{\text{Cell face},i} \Psi_{\text{Cell face},i}.
\end{aligned} \tag{3.21}$$

The so called flux averaging, presented by Lars-Erik Eriksson in the internal document "Pressure loss evaluation by flux averaging", instead sums all conservative fluxes and calculates a physical quantity based on these sums, equation 3.22. The physical interpretation of the value received from a flux averaging procedure is the quantity at highest possible entropy state for a given conservative flux situation. In other terms, it is like "realizing" all losses in a flow state by continuing the flow through an infinite, frictionless and adiabatic channel until the flow is completely uniform. This is simply due to the flux average being analogous to a mass/area average if the flow was completely uniform.

$$\begin{aligned}
F_{i,\text{Boundary}} &= \sum_{j}^{\text{Boundary cell faces}} F_{i,\text{Cell face},j}, \\
\Psi_{\text{Flux average}} &= f(F_1, ..., F_7)_{\text{Boundary}}.
\end{aligned} \tag{3.22}$$

# 4

# Methods

The following chapter describes the methods used to study Trollsol. In all simulation cases, the object of observation is a TRS setup with specific boundary conditions. Figure 4.1 shows the mesh of the domain at the shroud of the TRS along with the boundary conditions used and figure 4.2 displays the same information from a side view. The inlet boundary condition (Left in figures) is specified with total enthalpy, total pressure, static pressure, a velocity vector, turbulent kinetic energy, and turbulent dissipation rate. Static pressure is used only if there's reversed flow on the boundary. The outlet boundary condition (Right in figures) is specified only with static pressure.

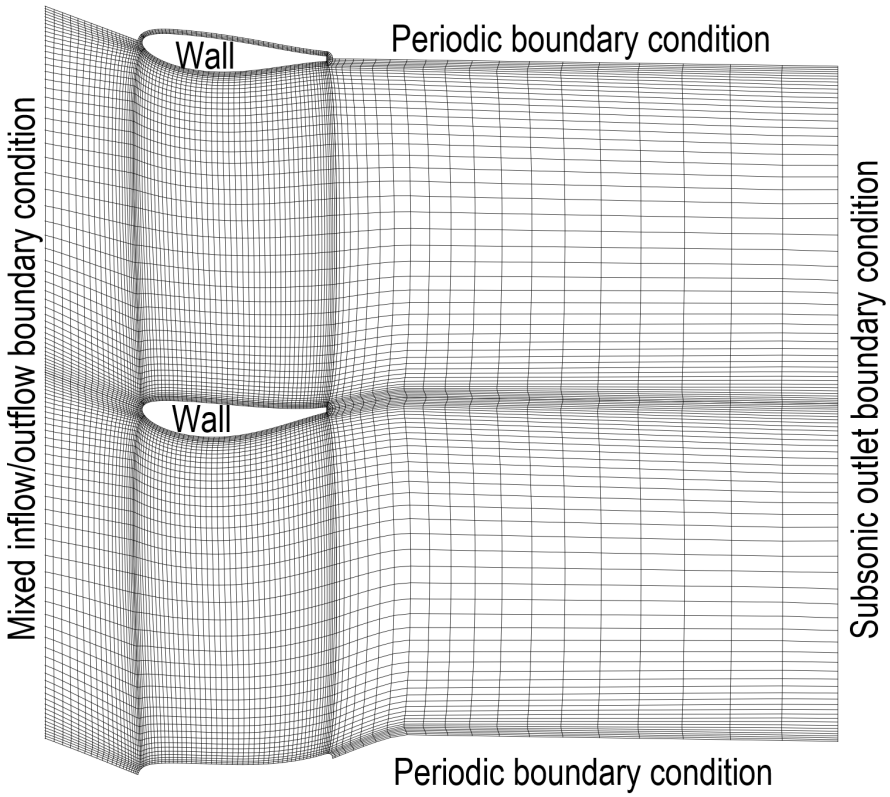



**Figure 4.1:** Mesh view at the TRS shroud. The horizontal and vertical direction represents the axial and tangential direction respectively.

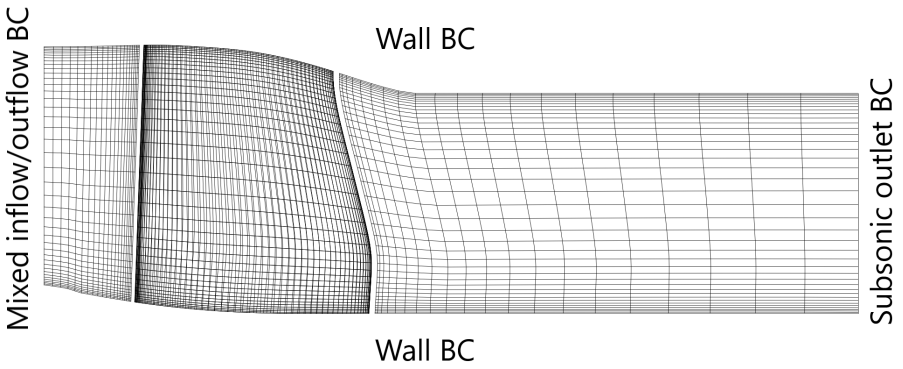



**Figure 4.2:** Mesh side view of the TRS. The horizontal and vertical direction represents the axial and radial direction respectively.

## 4.1 Validation study

To ensure that the solver behaves as it did previously, a reproduction of the past validation study is conducted, as described by Johansson [7]. First, previous data is collected in an Excel sheet made for easy parsing with a Python script. Then the geometry and boundary conditions from the previous validation study is exported from the old version of Volvane to the new version. The inlet swirl angle boundary condition, specified by a preset radial distribution, is varied by a constant difference in all radial points. Finally, the configurations run in the previous validation study are run again in the current version of Volvane.

### 4.1.1 Comparison with previous Trollsol version

After the results are collected from the new validation study, they are compared with the old data by a root mean square method of a studied variable with respect to inlet swirl angle, along with a minimum and maximum deviation of said variable. The primary variable to be studied is the mass-averaged total pressure loss, defined in equation 4.1. The original idea was to look into all the other variables presented in the previous validation study, but due to both data loss and lack of necessity (see sec. 5.1), only pressure loss with respect to inlet swirl angle is studied.

$$\text{Pressure loss} = \frac{P_{0,\text{ inlet, Mass average}} - P_{0,\text{ outlet, Mass average}}}{P_{0,\text{ inlet, Mass average}}} \tag{4.1}$$

### 4.1.2 Comparison to Fluent data

The subject of Trollsols capability in comparison to other commercial flow solvers is brought back from the past validation study, extending the scope to estimate an order of accuracy in comparison to a high-fidelity solution. The idea is to use said order of accuracy as a baseline for future capabilities of Trollsol. As found by Johansson [7], the pressure loss obtained from Trollsol is roughly equal to the pressure loss obtained from Fluent offset with a constant factor when a TRS has not begun to separate. This factor is independent of the inlet swirl angle provided

**Table 4.1:** Swirl angles at which a TRS separates. All values are fetched from the previous validation study by Johansson [7].

| | **Trollsol** | **LowRe-RKE** | **SST** $k-\omega$ | **Non-eq-wf** |
|---|---|---|---|---|
| TRS1 | 14 | 12 | 4 | 10 |
| TRS2 | - | - | - | - |
| TRS3 | 14 | 8 | 4 | 6 |
| TRS4 | 14 | 10 | 0 | 14 |
| TRS5 | 12 | 8 | 0 | 8 |

no separation, but is dependent on the geometry of the TRS. Given that future use cases of Trollsol use this approach to adjust the pressure loss, the order of accuracy calculated needs to mirror this process. The new measurement "Adjusted Difference in total Pressure Loss" (ADPL) is introduced in equation 4.2, where comp is the simulation method compared against. This measurement estimates the offset factor by taking the fraction of the total pressure loss in the compared method and Trollsol at the ADP angle. Due to the nature of the process, only values measured at swirl angles below separation are compared. These angles were found by Johansson and are presented in table 4.1.

$$\mathrm{ADPL}_{\mathrm{comp}} = |\Delta P_{0,\ \mathrm{comp}} - \Delta P_{0,\ \mathrm{CUDA\ 2024}} \cdot \frac{\Delta P_{0,\ \mathrm{comp,\ ADP\ angle}}}{\Delta P_{0,\ \mathrm{CUDA\ 2024,\ ADP\ angle}}}|. \tag{4.2}$$

By calculating the mean ADPL on the reference data provided by Johansson, an order of accuracy for the usage of Trollsol can be found.

## 4.2 Source code structure

As this study is limited to the inner workings of Trollsol itself, the source code studied includes C/C++ CUDA libraries and the Python module Trollsol. As the code is a mixture of languages, the built-in python debugger, the GNU debugger and the CUDA-GDB debugger are used. An entry point to the code is located and the code is followed step by step from there to gain an understanding of the overall structure. Going forward, the routines encountered are linked with their respective theoretical counterpart and a code-related reason for the interpolation error at the inlet previously mentioned is sought out.

Furthermore a testing executable is created which runs all relevant routines on a 3 by 3 hexagonal mesh. The code for this executable is compiled with the NVCC (CUDA) compiler, making debugging the CUDA library itself easier since no additional interfaces are used and ensures that library routines behaves the same as before during changes.

For future internal references to the library and module, a Sphinx documentation website is created with code information gathered in Doxygen.

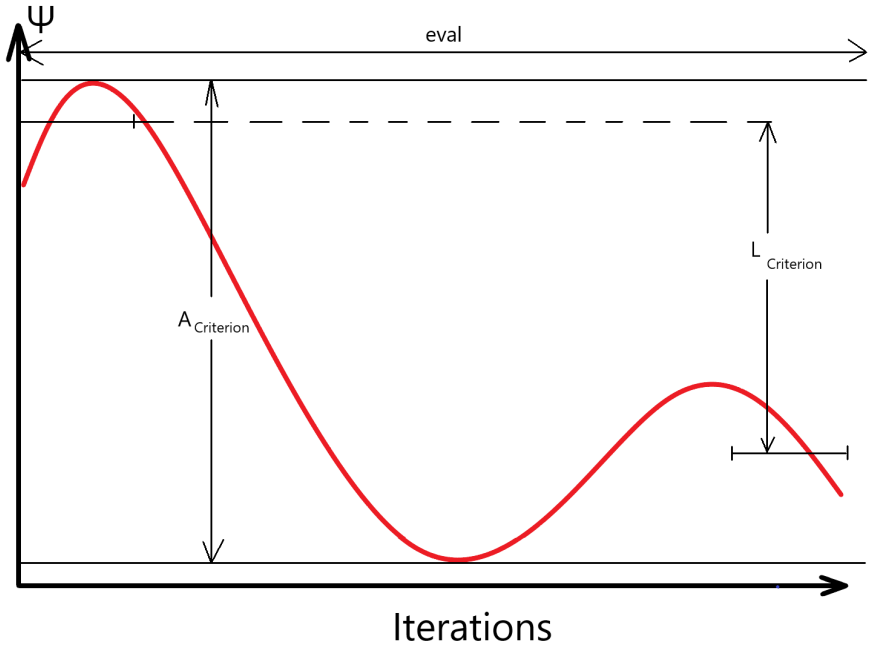



**Figure 4.3:** Visualization of what is measured by $A_{\text{Criterion}}$ and $L_{\text{Criterion}}$.

## 4.3 Convergence criteria implementation

To extend the capabilites of Trollsol, a new routine dictating when the solver should stop was implemented. Previously the amount of timesteps taken was dictated only directly by the user, which in most cases was set to 2000 steps. In general it cannot be said that a solution to the governing equations is converged after a set number of time steps, but rather it has to be evaluated on the fluctuations of physical quantities studied. The newly added convergence criteria handles this through a check in amplitude and lean over a set previous amount of timesteps. This is dubbed an "Asymptotic" criterion. The definition of criterion amplitude $A_{\text{Criterion}}$ is presented in equation 4.3, while the definition of criterion lean $L_{\text{Criterion}}$ is presented in equation 4.4.

$$A_{\text{Criterion}} = \frac{\max\{\Psi_{n-eval}, ..., \Psi_n\} - \min\{\Psi_{n-eval}, ..., \Psi_n\}}{\text{mean}\ \{\Psi_{n-eval}, ..., \Psi_n\}}, \tag{4.3}$$

$$L_{\text{Criterion}} = \frac{\text{mean}\{\Psi_{n-eval}, ..., \Psi_{n-eval+\text{floor}(eval/10)}\} - \text{mean}\{\Psi_{n-\text{floor}(eval/10)}, ..., \Psi_n\}}{\text{mean}\{\Psi_{n-eval}, ..., \Psi_n\}}, \tag{4.4}$$

where $\Psi$ is a physical quantity chosen to observe and evaluate on, the index representing which time step the quantity is taken from, $n$ is the current timestep and *eval* is a set previous amount of timesteps chosen to evaluate the criterion on. A visualization of how these criterion would work is presented in figure 4.3.

Note that the definition of both amplitude and lean in the convergence critiera contains a division operation with the mean of a set in the denominator. Should the mean of the physical quantity approach zero the criterion evaluations will approach

infinity, meaning in practice that the criterion will be unstable. Therefore when choosing a physical quantity to observe with this criterion, it should be expected to have a non-zero value in a converged state.

### 4.3.1 Finding optimal criteria settings

Trollsol in its current state and in the forseeable future is a tool used in the design stage of a new TRS geometry. For a large range of proposed designs, Trollsol must then be used in a manner where not only the approximate solution to the governing equations are found fast, but also in a manner where the end user tampers as little as possible with the solver settings. It is therefore crucial to find optimal settings for the convergence criteria that makes sure sufficient convergence is found for every case. Writing more explicitly, ideally the questions to be answered regarding these settings are as follows:

- What setup of the asymptotic convergence criteria ensures that we can definitively say all possible simulation setups have converged if it is reached and not converged if it is not reached?
- In the set of such setups of the convergence criteria, which setup requires the least amount of time steps?

Practically it is not possible to reach the conclusion of convergence for all possible simulations setups. On one side this is due to time constraints of the thesis, but on the other it is also because of the vast variations of different simulation setups possible. Focusing on the goal of Trollsol, the viability of the convergence criteria settings is tested on a single TRS for increasing swirl angles, starting from the design point and ending at an angle where the solver does not converge. As the design phase of a TRS focuses primarily on the pressure loss over the part, the convergence criteria is set to monitor both the inlet and outlet flux averaged total pressure as this operation is implemented in G3DCUDA.

To evaluate that all setups tested actually have reached sufficient accuracy to be called converged, for each swirl angle one case is run where the convergence is judged manually. This is done through physical arguments and looking at residuals and monitors from the case. When a case with a swirl angle is said to be converged, a tested convergence criteria setting is then compared with this manually checked converged case in terms of mass averaged primitive variables. The argument goes that if sufficient accuracy is reached in a set of primitive variables which have the same span as the conservative fluxes, any sort of analysis on the boundary will have the same order of accuracy as found on these primitive variables. The formula to evaluate the error from a sample solution from a convergence critera with the converged solution is presented in equation 4.5, where $\Psi$ is an arbitrary variable.

$$\text{Error from convergence criteria}(\Psi) = \frac{\left|\Psi_{\text{Convergence criteria}} - \Psi_{\text{Manually converged}}\right|}{\Psi_{\text{Manually converged}}}. \tag{4.5}$$

# 5
# Results and discussion

The analysis of the GPU solver has been conducted in multiple steps which each to some extent build upon each other. First, the validation study answers the accuracy of the solver both in comparison to its own instance from 2018 and in comparison to the commercial solver ANSYS Fluent. The source code is then parsed through, looking into potential sources of errors that could explain the accuracy or the lack thereof in the validation. Finally, a new set of functions is added to the solver extending its capabilities in the form of convergence criteria. This extension is analyzed from the view of a design process, seeking to answer the question of how the solver will get accurate enough results with the least possible amount of timesteps.

## 5.1 Validation study

The mass-averaged total pressure loss in TRS1 for a range of swirl angles with different simulation methods is presented in figure 5.1. For reference, "CUDA_PROD" refers to the Trollsol version in use 2018, "CUDA_DEV" refers to the development version of Trollsol 2018, "RKE_LowRe" refers to a realizable $k-\epsilon$ turbulence model simulation with a low Reynolds number mesh run in Fluent, "non_eq_wf" refers to a realizable $k-\epsilon$ turbulence model simulation with a high Reynolds number mesh run in Fluent along with so called " Non-equilibrium wall-functions", "SST_LowRe" refers to a SST $k-\omega$ turbulence model simulation with a low Reynolds number mesh run in Fluent and finally "CUDA_2024" refers to the current Trollsol version. From a simple visual inspection, the current of Trollsol coincides well with both versions from 2018.

Making a quantative study on the coincidence of both versions, the differences in total pressure loss between CUDA_2024 and CUDA_DEV (2018) for every TRS is presented in table 5.1. The difference between the two versions reaches a maximum of $2.1 \cdot 10^{-4}$, but mostly remains one or more orders below this value. A comparison is also done between CUDA_2024 and the Fluent simulations with the adjustment proposed by Johansson (See section 4.1.2), measured with ADPL. The root mean square value of ADPL for all Fluent simulations and all TRS on angles where the flow is attached are presented in figure 5.2, where the error order of magnitude is found to be $\mathcal{O}(10^{-2})$. The values of the fraction $\frac{\Delta P_{0,\text{ comp, ADP angle}}}{\Delta P_{0,\text{ CUDA 2024, ADP angle}}}$, used to evaluate ADPL for the different TRSs, are as following: 1.16 for TRS1, 1.09 for TRS3, 1.05 for TRS4 and 1.01 for TRS5.

Given the results from both comparison between the previous and recent ver-

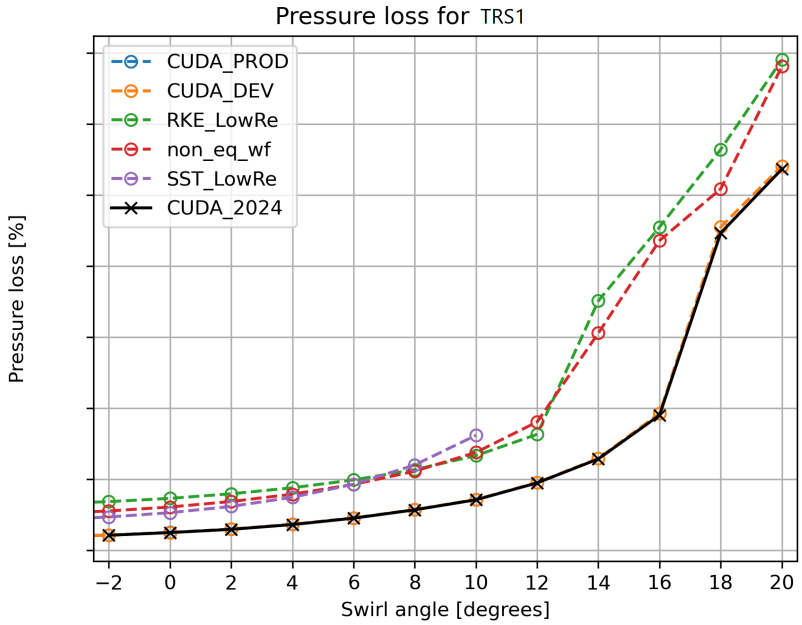



**Figure 5.1:** Total pressure loss, $(P_{0,inlet} - P_{0,outlet})/P_{0,inlet}$, with respect to inlet swirl angle. Difference between CUDA_2024 and CUDA_DEV (2018): RMS: 0.00713%, max: 0.0021%, min: $1.49 \cdot 10^{-5}$%. *Note that CUDA_PRO D is not visible in the graph since it completely coincides with CUDA_DE*

sion of Trollsol and between Trollsol and Fluent with adjustment, it is evident that the difference is significantly higher between Trollsol and Fluent with adjustment. Normally one would question the validity of Fluent as well as the validity of Trollsol, but due to internal validation work, agreements with clients of GKN and vast engineering expertise on design of TRS from key figures at GKN, the results from Fluent are assumed to be "correct". With such an assumption made, the difference between Trollsol and Fluent with an adjustment becomes the leading source of error in Trollsol. If this is the case, the difference between the previous and recent version of Trollsol becomes negligible as it is two orders of magnitude lower than the leading error term, concluding that CUDA_2024 produces essentially equal results to those produced by CUDA_DEV.

It should be noted that the ADPL produced seems to be dependent on the magnitude of the swirl angle. This is apparent from figure 5.2, since ADPL from SST_LowRe is lower than ADPL from RKE_LowRe on most TRSs, and SST_LowRe separates a lot earlier than the other models (table 4.1). This does not, however, take away from the aforementioned conclusion on the state of Trollsol, as the order of accuracy still can be said to be $\mathcal{O}(10^{-2})$ even for the lowest value of ADPL.

What the validation study shows is that Trollsol can be used in a design phase setting. As long as a design does not result in a flow separation, the resulting solution can be used in comparison with another flow situation simulated in Trollsol. When a design that fits the requirements of the user is found, a high fidelity simulation tool, like ANSYS fluent, can be used to more precisely replicate the flow

**Table 5.1:** Difference in total pressure loss between CUDA_2024 and CUDA_DEV (2018) for every TRS studied. The presented values are RMS averaged, minimum values and maximum values of the difference in total pressure loss over all swirl angles for each case.

| **TRS** | **RMS** | **Max** | **Min** |
|---|---|---|---|
| TRS1 | 0.00713% | 0.021% | $1.49{\cdot}10^{-5}$% |
| TRS2 | - | - | - |
| TRS3 | 0.001% | 0.00266% | 0.000122% |
| TRS4 | 0.000285% | 0.000616% | $1.31{\cdot}10^{-5}$% |
| TRS5 | 0.000672% | 0.00134% | $3.71{\cdot}10^{-5}$% |

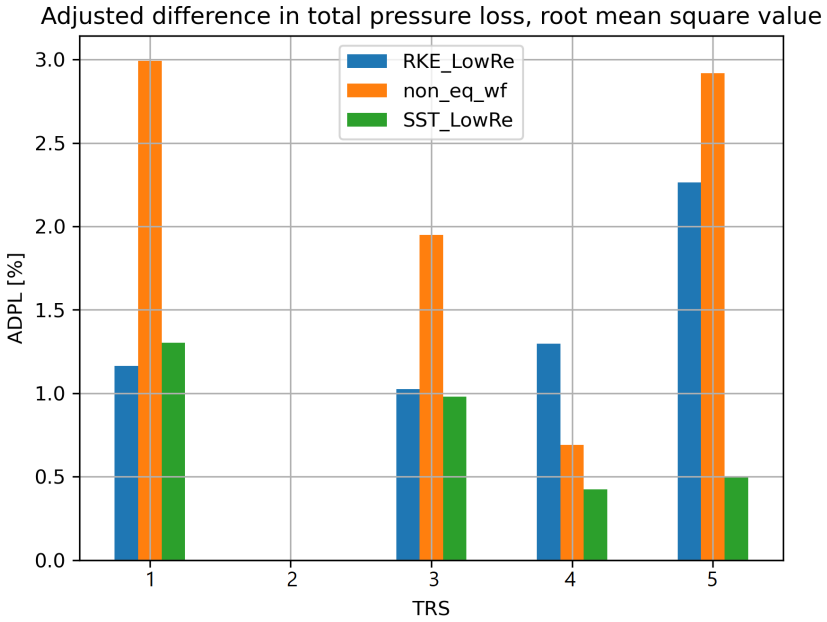



**Figure 5.2:** Adjusted difference in total pressure loss (ADPL) for each Fluent simulation method and each TRS. Note that the entry for TRS2 is empty due to a lack of data. Essentially, the numbers presented here are a measurement of which order of accuracy Trollsol has with Fluent simulations as a reference. Presented data points to the error being $\mathcal{O}(10^{-2})$.

situation. During this transition to high fidelity solutions, the user can be sure that the relation between different flow situations in the same geometry is kept, albeit with a geometry-dependent offset factor.

## 5.2 Source code structure

The overarching structure of Trollsol is based on three separate libraries: "Trollsol", a Python library; "G3DCUDA Wrapper", a C library and "G3DCUDA", a C/C++ CUDA library. The core functions, such as all numerical methods for solving the Navier-Stokes equations, along with all performance-critical routines, are located in G3DCUDA. Meanwhile, Trollsol handles the reading of meshes, boundaries, solver settings, as well as non-performance-critical routines such as displaying curated information to the user, tracking elapsed time, controlling solution progression, and post-processing routines such as value averaging, plotting, and file-writing. The role of the G3DCUDA Wrapper is to translate information back and fourth between Trollsol and G3DCUDA, using the official C-Python library. A schematic of this relation is presented in figure 5.3. Since the wrapper is written in pure C, everything that is imported from G3DCUDA to the wrapper has to be C compatible, meaning that tools such as classes and operator overloading cannot be used in what is presented to the wrapper.

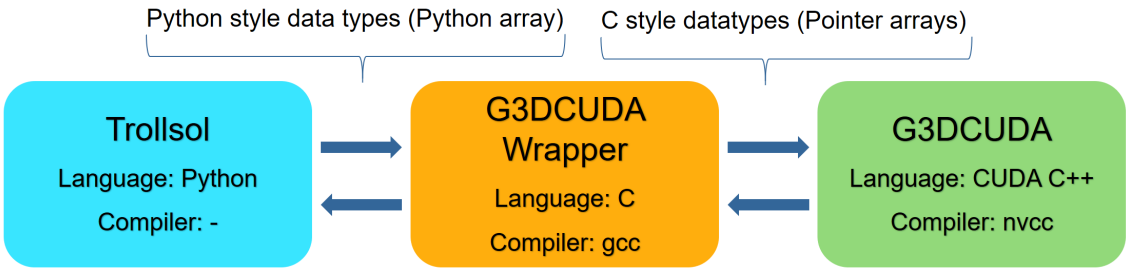



**Figure 5.3:** Simplified explanation of the G3DCUDA wrapper. Arrows represent transfer of data.

As stated before, Trollsol, or more specifically G3DCUDA, is an explicit compressible flow solver with local time stepping. To take a time step with an enhanced order of accuracy, the time discretization scheme is a third-order Runge-Kutta scheme. In code form, this is presented as a number of submethods as shown in figure 5.4. To save disk space, only two subsolutions are saved at any given instance, compared to the four subsolutions specified in equation 3.11. The variables that store these subsolutions are called *solv* and *solv*0. The gradient in time is also stored in a variable called *tder* which is generated by running other submethods to get face fluxes. A rundown on how information is moved between *solv*, *solv*0 and *tder* is shown in table 5.2.

Focusing on the subroutines found in the second, third and fourth columns in figure 5.4, they ultimately calculate the gradient in time for each pseudo time step in the Runge-Kutta scheme. "Auxvr" calculates primitive variables from the conservative variables stored in *solv*. "Vgrad" computes the first order velocity gradients on all faces in the domain. "Flux" divides into two categories, these being convective and diffusive fluxes. The convective fluxes are calculated on all faces in the domain, specified by the upwind scheme presented in section 3.3, while the diffusive fluxes are calculated on interior faces, mesh interfaces and walls. "KESRC" and "KESRClim" calculates and limits the source terms originating from the discretization of

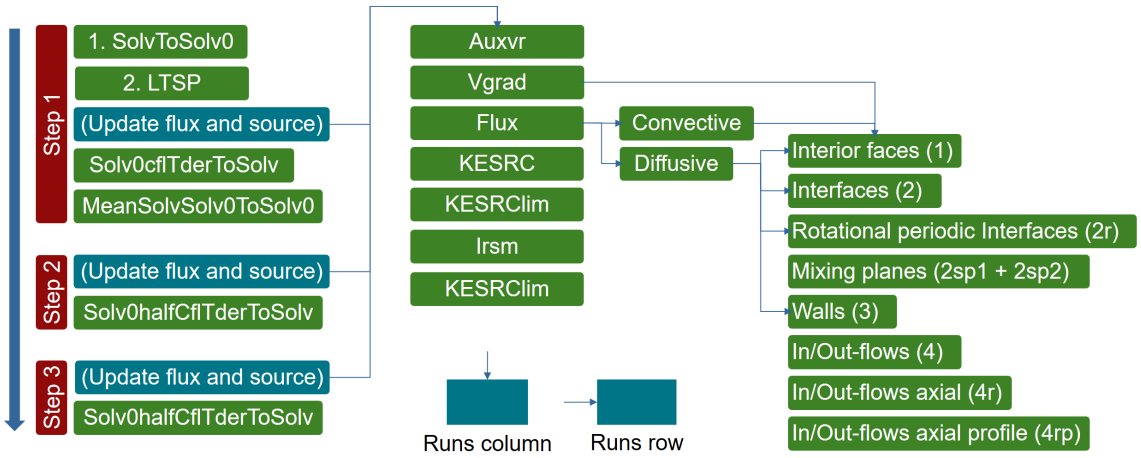



**Figure 5.4:** Overarching structure of the three step Runge-Kutta explicit solver. The column position from left to right corresponds to the level of function calls and the arrow corresponds to calling for the next level and retrieving data. As an example, Vgrad calls for all functions in the lowest level of functions.

the $k - \epsilon$ scheme. "Irsm", or the "implicit residual smoothing method" is a method which recalculates the *tder* variable by smoothing the solution in all the cardinal directions [20]. The effect of this method is that information about neighboring cells are shared, meaning that stability of the time discretization scheme is increased, subsequently allowing for CFL numbers above 1.

### 5.2.1 Test routine

A test routine has been added to G3DCUDA which is compiled purely with NVCC. The purpose of this routine is to make sure G3DCUDA produces identical results compared to previous versions, ensuring that no mistake is allowed to pass through to a release version. The routine also gives an easier way to step through the code with a debugger, given that a new developer is not familiar with the code.

The test works by generating an equally spaced cubical 3x3 mesh. The domain contains an inlet, an outlet and four walls. The first time the test is run, all solver variables are saved to a file, creating a reference for validation. When the test is run during development, the new solver variables are compared to the reference with a tolerance equal to the accuracy of floating point precision, namely $10^{-6}$. If the difference between any of the reference and new solver variables exceeds this tolerance, the test notifies the user of failure.

### 5.2.2 Pressure discrepancy

With regards to the discrepancy in total pressure on the inlet, discussed in section 1.1, the problem is present in two separate instances. Firstly, the discrepancy is present when computing convective fluxes through the inlet faces. Secondly, the discrepancy is present when post processing a solution by interpolating to a boundary face.

**Table 5.2:** Explanation of data transfer between *solv*, *solv*0 and *tder*. The table should be read from left to right, then top to bottom, meaning that the Method happens first and the contents of the variables are shown after the method has run. The routines used in each step are written in the first column and correspond to the methods seen in the first column in figure 5.4. The timestep $\Delta t$ is calculated in the method "LSTP," an abbreviation of "Local Time StePping factor.

| **Method** | ***solv*** | ***solv*0** | ***tder*** |
|---|---|---|---|
| SolvToSolv0 | $y_n$ | $y_n$ | - |
| (Update flux and source) | $y_n$ | $y_n$ | $k_1$ |
| Solv0cflTderToSolv | $y_n + \Delta t k_1$ | $y_n$ | $k_1$ |
| MeanSolvSolv0ToSolv0 | $y_n + \Delta t k_1$ | $y_n + \frac{1}{2}\Delta t k_1$ | $k_1$ |
| (Update flux and source) | $y_n + \Delta t k_1$ | $y_n + \frac{1}{2}\Delta t k_1$ | $k_2$ |
| Solv0halfCflTderToSolv | $y_n + \frac{1}{2}\Delta t k_1 + \frac{1}{2}\Delta t k_2$ | $y_n + \frac{1}{2}\Delta t k_1$ | $k_2$ |
| (Update flux and source) | $y_n + \frac{1}{2}\Delta t k_1 + \frac{1}{2}\Delta t k_2$ | $y_n + \frac{1}{2}\Delta t k_1$ | $k_3$ |
| Solv0halfCflTderToSolv | $y_n + \frac{1}{2}\Delta t k_1 + \frac{1}{2}\Delta t k_3 = y_{n+1}$ | $y_n + \frac{1}{2}\Delta t k_1$ | $k_3$ |

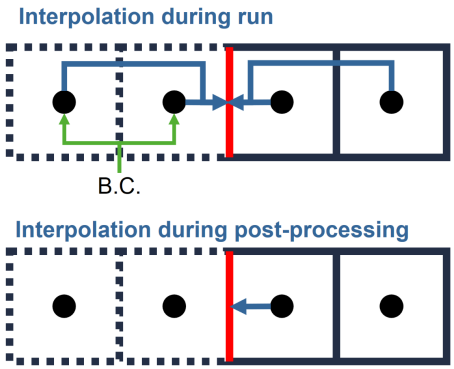



**Figure 5.5:** Interpolation schemes to boundary. Top: interpolation during run, boundary conditions are copied to two ghost cells and the boundary value (value on red line) is computed from the same interpolation scheme used on inner faces. Bottom: interpolation during post, value from the outermost cell is copied as a boundary value.

In the first instance the problem arises with the ghost cell approach. One or two ghost cells are created outside of the domain adjacent to a cell on the inlet boundary. The boundary condition properties are copied into these cells, afterwards the convective fluxes through the boundary are computed in the same manner as if they were internal faces. This leaves a discrepancy compared to the original meaning of the boundary condition, namely that the values specified should be applied exactly on the boundary, not adjacent to it. An illustration is provided in the upper half of figure 5.5.

In the second instance the problem stems from a zeroth order interpolation from the outermost cell to the boundary. This is run after a solution is found by the solver and does not have any impact on the solution itself, but could cause interpretation errors by users of the program. An illustration is provided in the lower half of figure 5.5.

The question then arises on what to do about this discrepancy. Due to time constraints it was not possible to remedy these errors, but there are still propositions to be made on future work. As found previously in another study, this discrepancy might not be major enough to do anything about since the errors are small and a completely accurate remodeling of the boundary condition would require significant changes in the numerical schemes. Another solution might then be to match the errors in the first and second instance where discrepancy occurs with each other. In a post-processing state, interest is placed in what the solver has produced but, in it's current state, the total pressure on the inlet reported by Trollsol is not the one used by G3DCUDA. The fix is then to copy the runtime boundary condition routines to the routines during postprocessing.

### 5.2.3 Memory issues

A problem that remains unresolved is unexpected behavior when changing position of member variables in a given Struct. In C/C++ programming, a Struct is a datatype serving as a collection of variables [21]. When a variable in a Struct, a "member variable", is reached it is called by name. The order in which these member variables are declared in the Struct is therefore irrelevant, generally speaking. The issue presented when running Trollsol, however, is that this order of declaration does make an impact on the results. In figure 5.6 the order of declaration in an example Struct is changed and in figure 5.7 the consequences of such a change is displayed. Note that this discrepancy happens for a change in a specific Struct in G3DCUDA containing pointers.

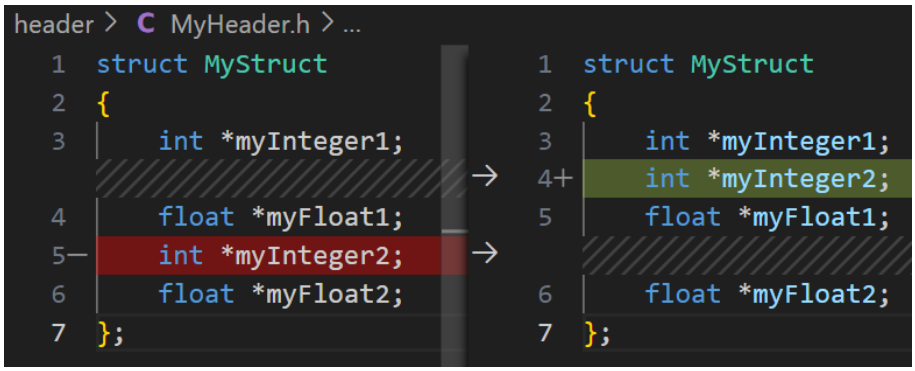
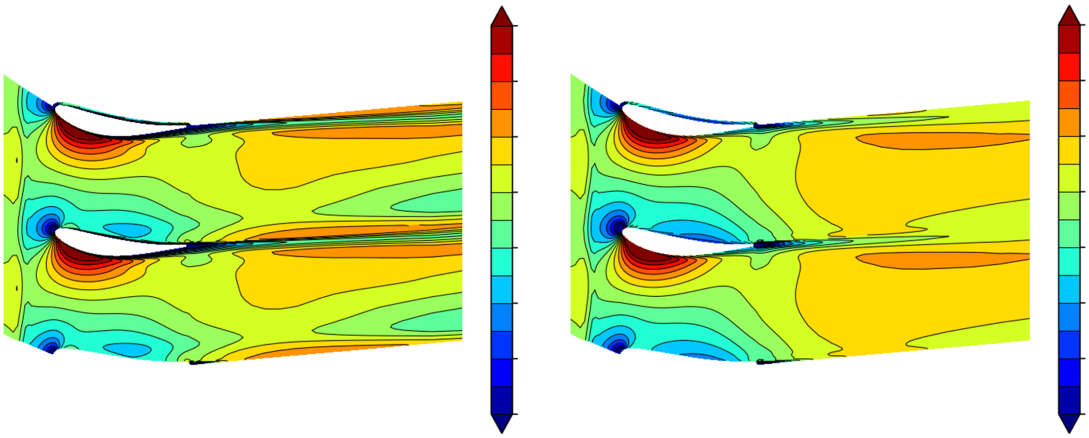



**Figure 5.6:** Example of switching out the position of a member variable declaration.

On a low level, the order of member variable declaration in a Struct dictates what position, or address, the variables will have in computer memory [22]. This indicates that an error stemming from the ordering of member variables probably has something to do with an issue in reading/writing to memory, a so called invalid read/write. An explanation of what an invalid read/write is can be found in figure 5.8. To find these types of issues, two programs are used. The first program is *Valgrind* which finds memory issues on the CPU-side and the second program is *compute-sanitizer*, which finds memory issues on the GPU side. Running Trollsol with compute-sanitizer revealed no GPU memory issues and running Valgrind on the new G3DCUDA test executable revealed no CPU memory issues in the G3DCUDA



**(a)** Current and correct simulation. **(b)** Modified and incorrect simulation.

**Figure 5.7:** Mass flow contour plots at radius = 10% between a correct and incorrect implementation of G3DCUDA. The color scales are essentially equal.

library. Running Valgrind on Trollsol directly gives a lot of false positive memory issues, which stems from Python itself handling memory allocation/deallocation differently from C/C++. Thus, it stands to reason that any memory issues are located in the G3DCUDA Wrapper library, but it is not exactly clear neither where nor what gives rise to these issues.

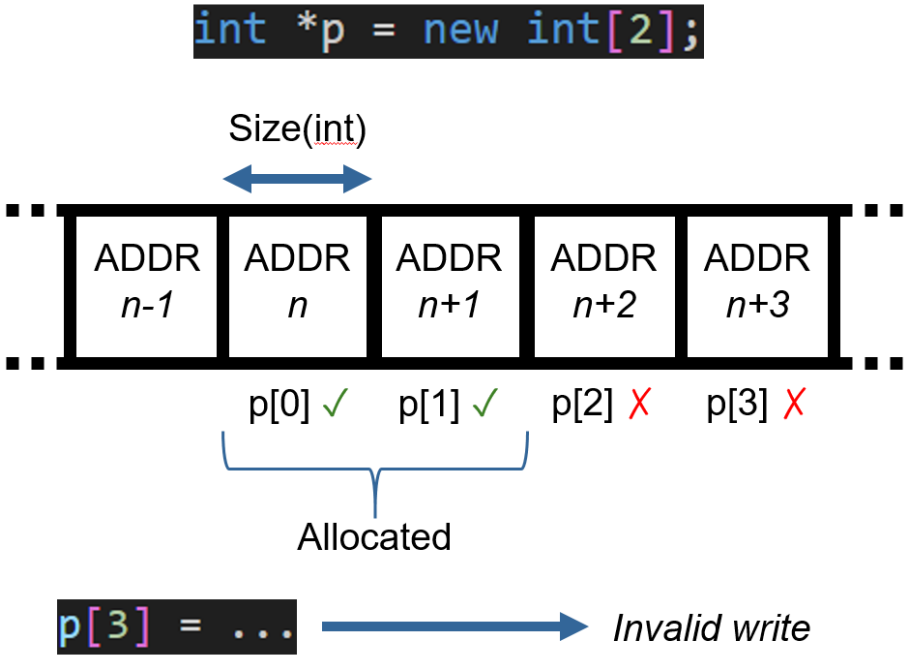



**Figure 5.8:** A visualization of computer memory and how an invalid read/write occurs. First, 2 integers are allocated with a pointer p to the address of the first integer. The values of integer i is reached through the statement p[i-1]. If a value that was not allocated is reached, it will not produce an error but will read/write to a memory position that is unknown.

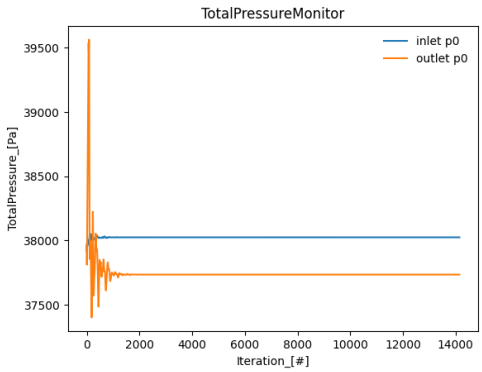
## 5.3 Convergence criteria sensitivity study

The convergence criteria specified in section 4.3 have been implemented into Trollsol. It is now possible for the user to first specify a given flux averaged quantity on a boundary as a monitor, meaning that said quantity will be saved over all iterations. A user may then use monitors to create convergence criteria, which will stop the solver when reached. The user may also create plots from both the monitors and criteria, as shown in figures 5.9a, 5.9b respectively. Optimal settings for these criteria are found through evaluation at both the ADP inlet angle and off-design angles.

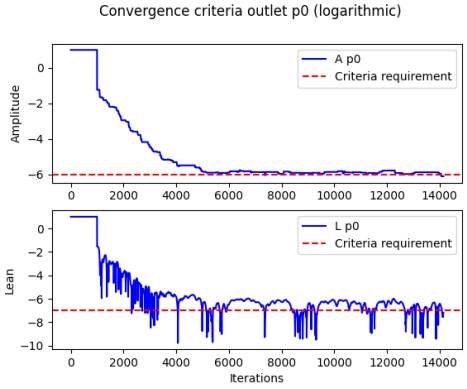
### 5.3.1 Evaluation at ADP angle

The errors from multiple area-, mass-, and flux averaged quantities on the inlet and outlet have been evaluated on TRS1 on the ADP, as described in equation 4.5. The mass averaged total pressure loss has also been evaluated in the same manner. The convergence criteria tested regards flux averaged total pressure on the inlet and outlet with the same settings as each other. The setting parameters are sweeped as following: *eval* ranges linearly from 50 to 250 with 5 entries, $A_{\max}$ ranges logarithmically from $10^{-6.5}$ to $10^{-3}$ with 12 entries. $L_{\max}$ ranges logarithmically from $10^{-8.5}$ to $10^{-5}$ with 7 entries.

The errors from total pressure loss are presented in figure 5.10 and the corre-




**(a)** Flux averaged total pressure on the inlet and outlet of an example TRS simulation domain.




**(b)** Example of a convergence criteria plot. The upper plot represents the value of $A_{\text{Criterion}}$ and the lower plot represents the value of $L_{\text{Criterion}}$. The dashed red line represents the maximum value a criterion can have to be fulfilled.

**Figure 5.9:** Example of implemented plots.

sponding amount of iterations taken to hit the convergence criteria are presented in figure 5.11. The errors from other averaged quantities are presented in appendix B, but are significantly lower in magnitude and therefore not as critical for the evaluation of criteria settings. As argued in section 5.1, a new error term in Trollsol should be significantly lower than $10^{-2}$, meaning that a good compromise for a convergence criteria setting should be $\mathcal{O}(10^{-3})$. A dashed red line has been inserted in the plots (fig 5.10) at this level for clarity.

The choice of *eval* directly influences the amount of timesteps required for the solution to be called converged. A too small *eval* means that the range $A_{\text{Criterion}}$ and $L_{\text{Criterion}}$ are evaluated on fails to capture relevant time-dependent fluctuations, while a too big *eval* means that the criteria captures irrelevant time-dependent fluctuations. An optimized *eval* would then be one of a length where some flow feature, say an eddy, could be transported out of the simulation domain in an *eval* amount of timesteps. Since Trollsol is an explicit solver with local timestepping, this corresponds to the number of cells in the streamwise direction divided by the CFL number, equation 5.1.

$$\text{Optimal } eval \approx \frac{\#\text{ Cells in streamwise direction}}{\text{CFL}}. \tag{5.1}$$

The TRS studied has 90 streamwise cells and the CFL number used is 3.5, giving an optimal *eval* of around 26. In practice *eval* should be chosen bigger, since this would account for eventual interactions between flow structures. The study points to a good value of *eval* for TRS1 as 250 for a CFL of 3.5.

In figure 5.10, the choice $L_{\text{Max}}$ seems to influence the consistency of $A_{\text{Max}}$ to hit convergence that is appropriate for itself. In other terms, the lean makes sure that a measurement in amplitude does not hide a drift upwards or downwards, but actually does not change on average. If $L_{\text{Max}}$ is too big in comparison to $A_{\text{Max}}$ it becomes

irrelevant and if it is too small it completely dominates, making $A_{\text{Criterion}}$ irrelevant. An optimized $L_{\text{Max}}$ is then one that does play influence in the convergence but does not drown out $A_{\text{Criterion}}$.

Finally, $A_{\text{Max}}$ is a direct limit on the amplitude of fluctuations. It is ideally directly proportional to all errors from the averaged quantities provided that *eval* and $L_{\text{Max}}$ fulfill their roles, meaning that $A_{\text{Max}}$ should be chosen such that any error does not exceed $10^{-3}$. Taking all of these points into account, a set of parameters that are a good compromise between solver speed and accuracy are presented in equation 5.2, running with a CFL of 3.5. If another CFL number is used, the recommended *eval* can be chosen as in equation 5.3.

$$eval = 250,\ A_{\text{Max}} = 10^{-4.5},\ L_{\text{Max}} = 10^{-7}, \tag{5.2}$$

$$\text{Recommended } eval \approx C_{eval}\frac{\#\text{ Cells in streamwise direction}}{\text{CFL}},\ C_{eval} = 10. \tag{5.3}$$

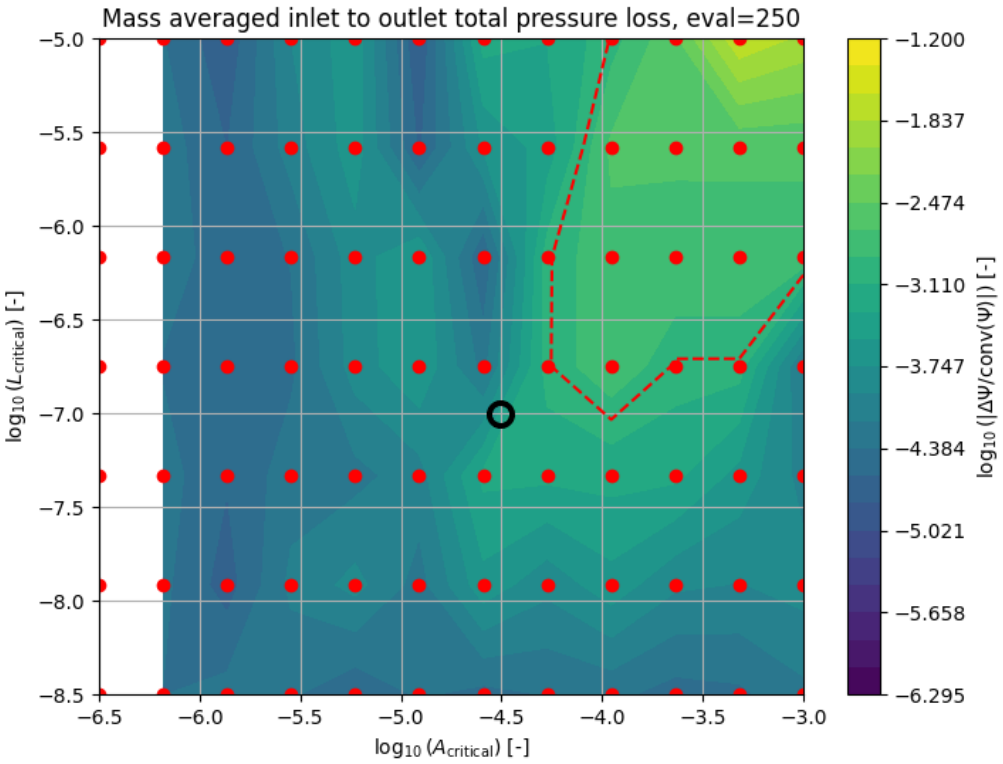



**Figure 5.10:** Contour plot of the error between a solution from convergence criteria and a solution with manually ensured convergence (Formula 4.5, logarithmically scaled). The $x$ and $y$ axis represents the $A_{\text{critical}}$ and $L_{\text{critical}}$ on a given simulation setup, logarithmically scaled. *eval* is 250. The physical quantity evaluated on is the mass averaged inlet to outlet total pressure loss. The red dots represent tested data points on which the contour plots are based. The red dashed line is a contour line at level $10^{-3}$. The black circle indicates a point chosen to give a good combination of solver speed and accuracy (Equation 5.2).

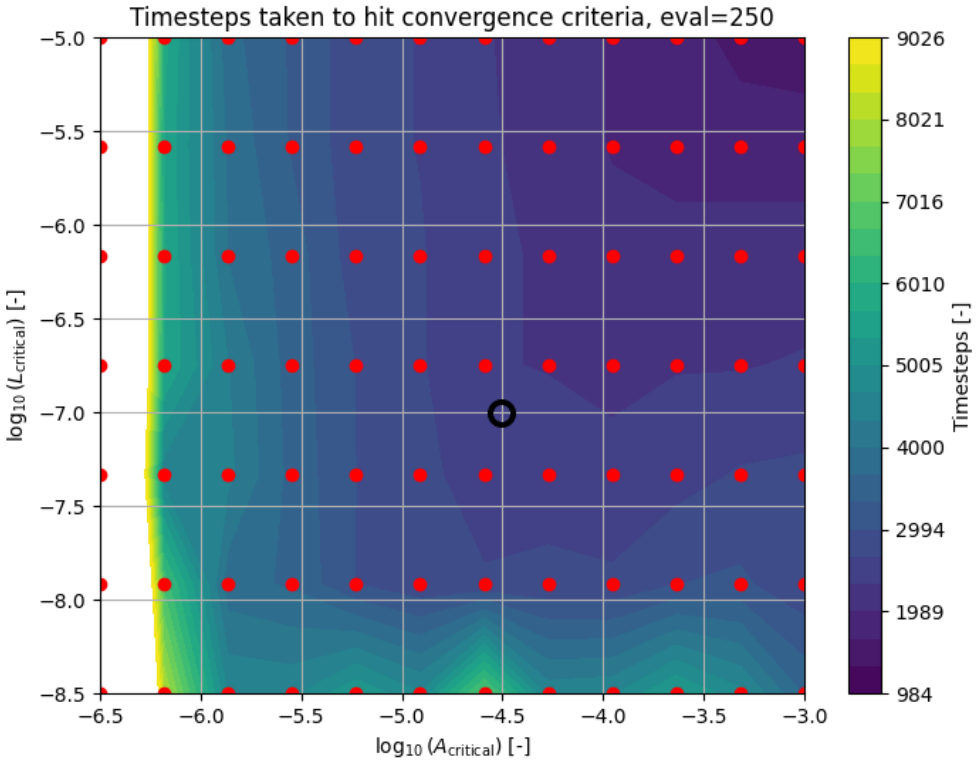



**Figure 5.11:** Contour plot of the timesteps taken to fulfill convergence criteria. The $x$ and $y$ axis represents the $A_{\text{critical}}$ and $L_{\text{critical}}$ on a given simulation setup, logarithmically scaled. *eval* is 250. The scale is cut off after the last entry that converged using criteria, otherwise 20 000 timesteps were taken. The red dots represent tested data points on which the contour plots are based. The black circle indicates a point chosen to give a good combination of solver speed and accuracy (Equation 5.2).

### 5.3.2 Evaluation at off-design swirl angles

Having found a set of criteria parameters, the total pressure loss error compared to manually converged cases for different swirl angles are presented in figure 5.12 with a CFL number of 3.5 and in figure 5.13 with a CFL number of 0.9. The manually converged cases have a CFL number of 3.5 for swirl angles from -4 to 14 degrees, thereafter reducing the CFL number to 1.5 and below. First, this study indicates that sufficient accuracy is reached with the convergence criteria, given that the CFL number is sufficiently low. Second, the study indicates that the convergence criteria were fulfilled even though convergence was not achieved when running with a too high of a CFL number.

The high discrepancy found when running a high CFL number does raise the question if the parameters to the convergence criteria should be lowered further. Empirical evidence points to the lowest value of $A_{\text{Criterion}}$ attainable being in the order of $10^{-6}$, which is only hit if the solution is actually converged (assumed), though lowering $A_{\text{Max}}$ to this level significantly raises the required amount of timesteps (figure 5.11). Another solution would be to reduce the CFL number, but this also raises the amount of timesteps. Yet another method would be to look at wall shear stress contours at the suction side of the blade, which could indicate separation. If separation is found, one could either try a new design or reduce the CFL number accordingly, otherwise one could assume that convergence can occur.

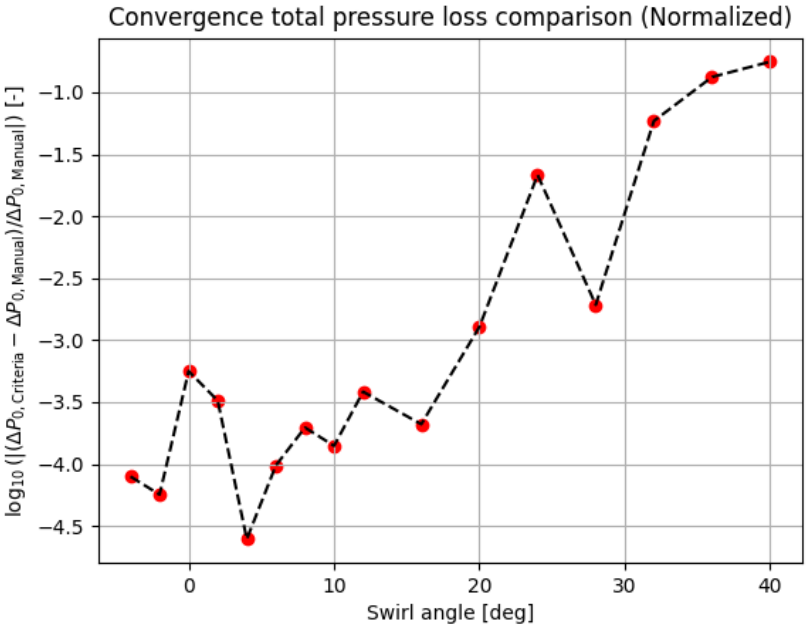



**Figure 5.12:** Comparison of total pressure loss between Trollsol simulations converged on either set criteria or manual check. When run with convergence criteria, CFL was set to 3.5. When run manually, CFL varied between 3.5 and 0.9 to achieve optimal results. Swirl angles up to ADP+25 were analyzed, but simulations beyond ADP+20 diverged and did not give results (Nan values).

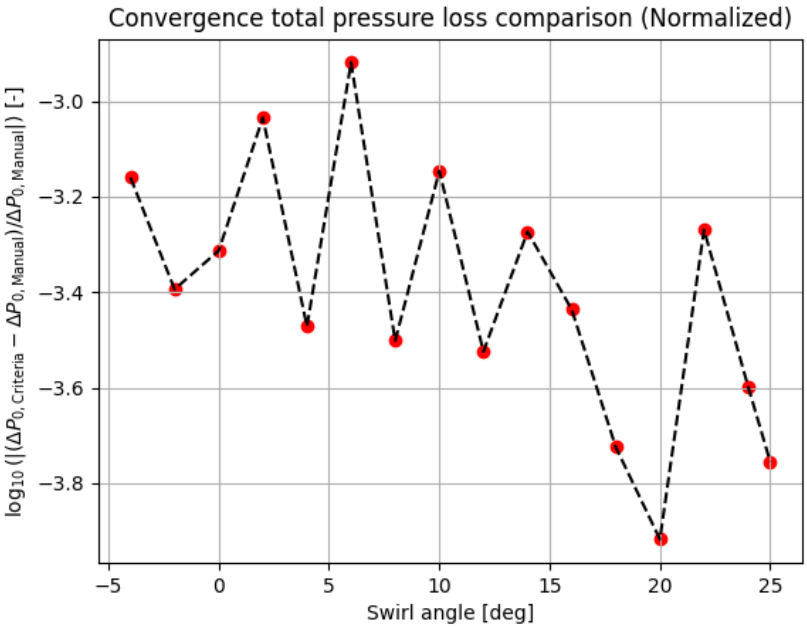



**Figure 5.13:** Comparison of total pressure loss between Trollsol simulations converged on either set criteria or manual check. When run with convergence criteria, CFL was set to 0.9. When run manually, CFL varied between 3.5 and 0.9 to achieve optimal results.

## 5.4 Future work

While the viability of Trollsol and G3DCUDA has been analyzed and improved in this study, there are still questions and issues that should be resolved for optimal usability. In the subsections below, such questions are presented.

### 5.4.1 Inlet/outlet boundary primitive interpolation

Although the reason for the varying total pressure inlet boundary has been found (see section 5.2.2), it is not clear how big of an impact it has on the results. A future study should firstly focus on creating a consistent interpolation of all primitive variables to the boundaries between the interpolation used while running the solver and the interpolation used when retrieving post-processing data. To continue from there one would need to set up some sort of simulation where one can choose between two interpolations: one with the ghost cell approach as it is implemented today and one with an approach that sets the boundary conditions completely correct. If the results from these two simulation setups would differ significantly, the approach would need to be changed.

### 5.4.2 Fixing the memory issues

The memory issues presented in the G3DCUDA Wrapper still persists (see section 5.2.3) although its location is known roughly. A future development should be able to locate the exact source of the unexpected behaviour and rectify it. A suggestion would be to utilize *Valgrind* while running Trollsol, but somehow filter out the false positives generated by Python itself.

### 5.4.3 Excepting non-converging solution states

A problem that might stem from the current version of the convergence criteria is that they can assume that convergence has been reached even though it cannot be reached due to other solver settings. An example of this is seen in figure 5.12, where a too large CFL number is chosen for an unstable swirl angle. The ideal case would be that the convergence criteria are therefore not hit which would signal to the user that something is wrong, but in the current implementation the convergence criteria are hit and the user might get the impression that a solution with good accuracy has been found. A solution to this might be to set convergence criteria on the total pressure loss directly instead of looking at total pressure on the inlet/outlet, but regardless of the solution, it should not give the wrong impression to the user.

### 5.4.4 Development in programs utilizing Trollsol

Although Trollsol has been updated, the new features requires special handling from programs that utilize Trollsol. An example is Volvane, where specification for convergence criteria settings that can be sent to Trollsol would need to be implemented.

# 6
# Conclusion

The current version of Trollsol has been analyzed in terms of accuracy in total pressure loss compared to its previous version and ANSYS Fluent. It was found that the current and previous versions coincide with an error in the order of $\mathcal{O}(10^{-4})$, proving negligible change when comparing to the error between Trollsol and Fluent of $\mathcal{O}(10^{-2})$.

The source code of Trollsol, G3DCUDA Wrapper and G3DCUDA has been analyzed and documented, clarifying the link between the code and the governing equations. A test routine was added to the G3DCUDA library, easing future development efforts. A vital error in the G3DCUDA Wrapper was found pertaining to low level memory issues in the code, which does not seem to impact the results in its current state, but does give unexpected results from seemingly trivial changes in the code which could lead to unexpected errors in the future.

The origins of the unexpected total pressure variation on the inlet boundary was found to be from two different sources. On one hand while the solver is running a ghost cell approach is used, meaning that boundary condition values are placed slightly offset from the position they ought to occupy. On the other hand when retrieving post-processed data, interpolation to the boundary from inside of the domain is done in a zeroth order fashion leading to substantial errors. A solution to these problems have been proposed, but not implemented due to time constraints.

Asymptotic convergence criteria have been added to Trollsol in contrast to previously stopping on a specific iteration number. To these criteria a good setting compromising between solver speed and accuracy has been found, showing to consequently stop the solver right when good accuracy is found given that other solver settings are set up correct. A drawback to these criteria are false positives, where according to criteria a good solution has been found but in reality the solution is off significantly from a properly handled case.

# Bibliography


[1] N. Andersson, *Lecture notes in compressible flow*, Jan. 2013.

[2] D. Spiridon, "Advances in cuda for computational physics", *Bulletin of the Transilvania University of Brasov. Series III: Mathematics and Computer Science*, 227–236, Dec. 2023, ISSN: 2810-2029. DOI: 10.31926/but.mif.2023.3.65.2.19. [Online]. Available: http://dx.doi.org/10.31926/but.mif.2023.3.65.2.19.

[3] "Implementing gpu acceleration into the pycalc-les code using cupy", [Online]. Available: https://www.tfd.chalmers.se/~lada/phd-les/pyCALC-LES_report_Johannes.pdf.

[4] G. Amador and A. Gomes, "Cuda-based linear solvers for stable fluids", in *2010 International Conference on Information Science and Applications*, 2010, pp. 1–8. DOI: 10.1109/ICISA.2010.5480268.

[5] W. Xue, C. W. Jackson, and C. J. Roy, "An improved framework of gpu computing for cfd applications on structured grids using openacc", *Journal of Parallel and Distributed Computing*, vol. 156, pp. 64–85, 2021.

[6] V. Vikhorev, V. Chernoray, O. Thulin, S. Deshpande, and J. Larsson, "Detailed experimental study of the flow in a turbine rear structure at engine-realistic flow conditions", *Journal of Turbomachinery*, vol. 143, no. 9, May 2021, ISSN: 1528-8900. DOI: 10.1115/1.4050451. [Online]. Available: http://dx.doi.org/10.1115/1.4050451.

[7] C. Johansson, "Evaluation of an in-house gpu based cfd solver", *Department of Mechanics and Maritime Sciences*, 2018. [Online]. Available: https://hdl.handle.net/20.500.12380/256206.

[8] C. Levandowski, D. Corin Stig, D. Bergsjö, A. Forslund, U. Högman, R. Söderberg, and H. Johannesson, "An integrated approach to technology platform and product platform development", *Concurrent Engineering*, vol. 21, Mar. 2012. DOI: 10.1177/1063293X12467808.

[9] C. Rumsey, *Implementing turbulence models into the compressible rans equations*, accessed 2024-01-23, 2023. [Online]. Available: https://turbmodels.larc.nasa.gov/implementrans.html.

[10] W. Jones and B. Launder, "The prediction of laminarization with a two-equation model of turbulence", *International Journal of Heat and Mass Transfer*, vol. 15, pp. 301–314, Feb. 1972. DOI: 10.1016/0017-9310(72)90076-2.

[11] K.-Y. Chien, "Predictions of channel and boundary-layer flows with a low-reynolds-number turbulence model", *AIAA Journal*, vol. 20, no. 1, 33–38, Jan. 1982, ISSN: 1533-385X. DOI: 10.2514/3.51043. [Online]. Available: http://dx.doi.org/10.2514/3.51043.

[12] B. Launder and B. Sharma, "Application of energy dissipation model of turbulence to the calculation of flow near spinning disc", *Letters Heat Mass Transfer*, vol. 1, pp. 131–137, Nov. 1974. DOI: 10.1016/0735-1933(74)90024-4.

[13] M. Kato and B. Launder, "The modelling of turbulent flow around stationary and vibrating square cylinders", Jan. 1993.

[14] P. A. Durbin, "On the k-3 stagnation point anomaly", *International Journal of Heat and Fluid Flow*, vol. 17, pp. 89–90, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:122268727.

[15] L. Davidsson, "An introduction to turbulence models", Göteborg, Sweden, Tech. Rep., 2022.

[16] J. Larsson, "Numerical simulation of turbulent flows for turbine blade heat transfer applications", Oct. 1999.

[17] E. Hairer and G. Wanner, "Runge–kutta methods, explicit, implicit", *Encyclopedia of Applied and Computational Mathematics*, pp. 1282–1285, 2015.

[18] J. D. Anderson, *Computational Fluid Dynamics*, en, ser. McGraw-Hill series in mechanical engineering. New York, NY: McGraw-Hill Professional, Feb. 1995, pp. 162–165.

[19] L.-E. Eriksson, *Numerical simulation of compressible flows, lecture 1*, Mar. 2005.

[20] A. Jameson, "Numerical solution of the euler equation for compressible inviscid fluids", Feb. 1985.

[21] *C - structures*, 2024. [Online]. Available: https://www.tutorialspoint.com/cprogramming/c_structures.htm.

[22] S. J. (https://stackoverflow.com/users/13005/steve jessop), *Optimizing member variable order in c++*, Stack Overflow, (version: 2024-04-29). [Online]. Available: https://stackoverflow.com/q/892767.

# A

# Appendix 1: Validation study

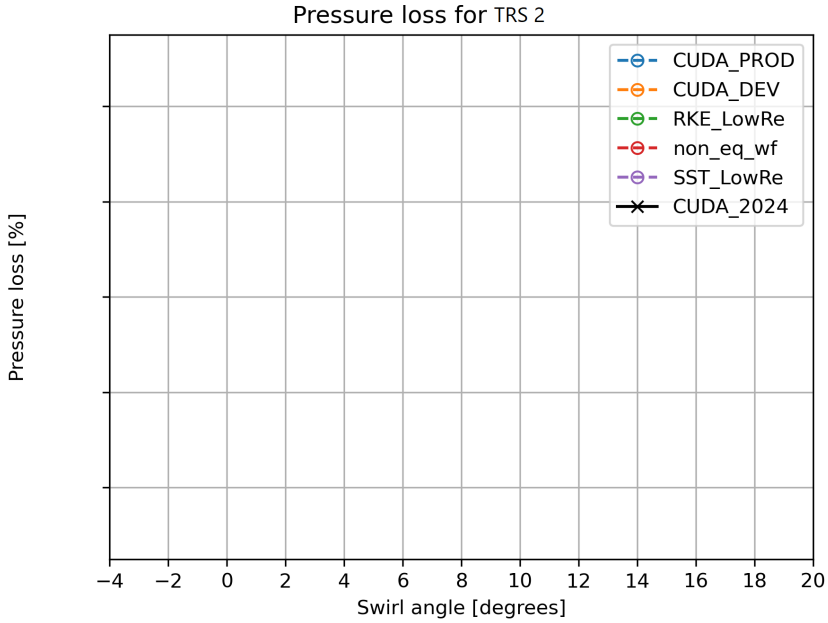



**Figure A.1:** Total pressure loss, $(P_{0,inlet} - P_{0,outlet})/P_{0,inlet}$, with respect to inlet swirl angle. Difference between CUDA 2024 and CUDA DEV (2018): No swirl angle entries.

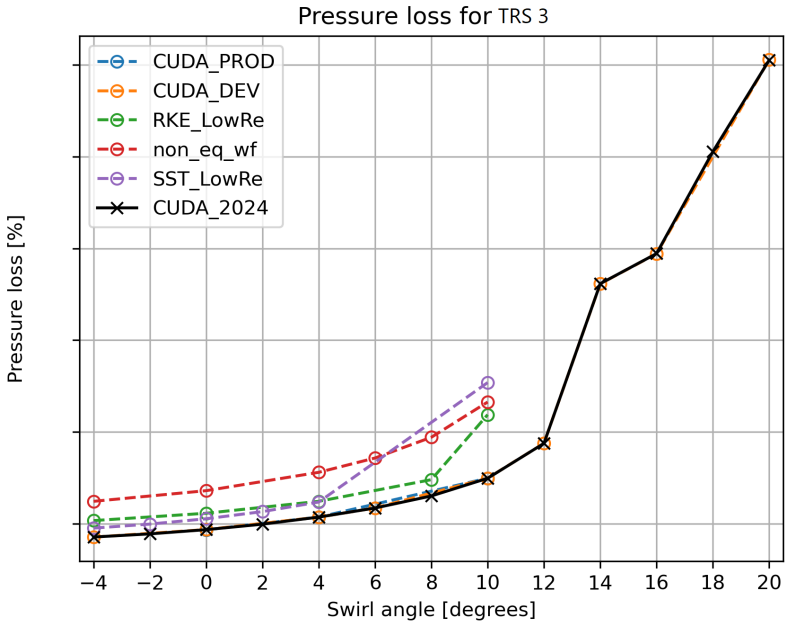



**Figure A.2:** Total pressure loss, $(P_{0,inlet} - P_{0,outlet})/P_{0,inlet}$, with respect to inlet swirl angle. Difference between CUDA 2024 and CUDA DEV (2018): RMS: 0.001%, max: 0.00266%, min: 0.000122%.

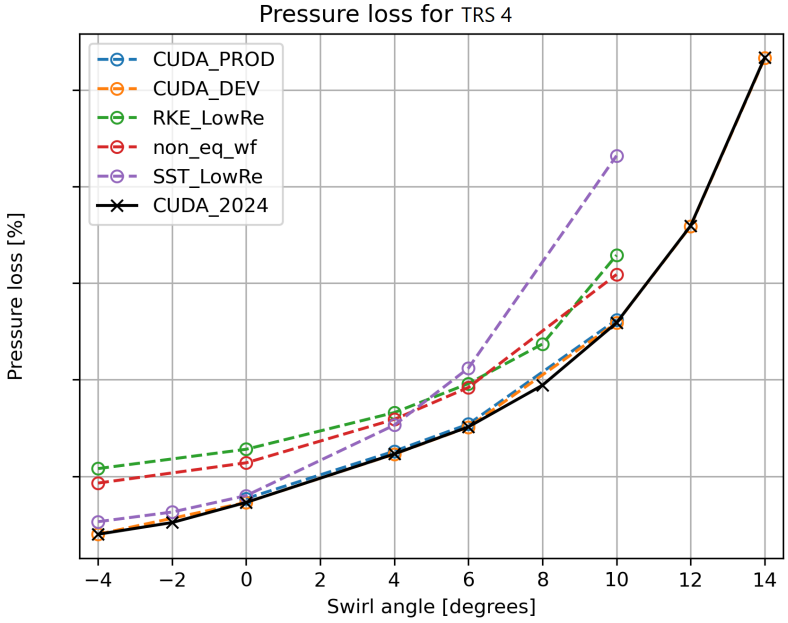



**Figure A.3:** Total pressure loss, $(P_{0,inlet} - P_{0,outlet})/P_{0,inlet}$, with respect to inlet swirl angle. Difference between CUDA 2024 and CUDA DEV (2018): RMS: 0.000285%, max: 0.000616%, min: $1.31 \cdot 10^{-5}$%.

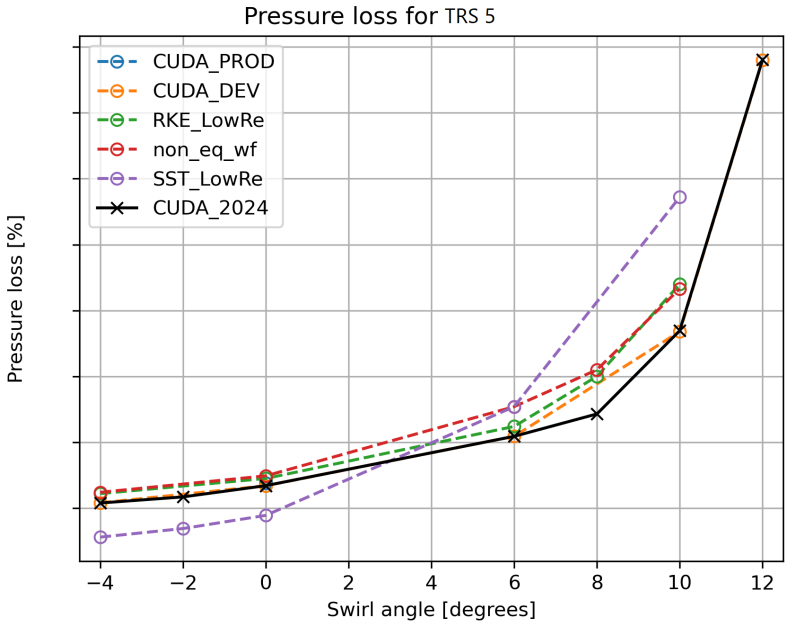



**Figure A.4:** Total pressure loss, $(P_{0,inlet} - P_{0,outlet})/P_{0,inlet}$, with respect to inlet swirl angle. Difference between CUDA 2024 and CUDA DEV (2018): RMS: 0.000672%, max: 0.00134%, min: $1.31 \cdot 10^{-5}$%.

# B

# Appendix 2: Convergence criteria study

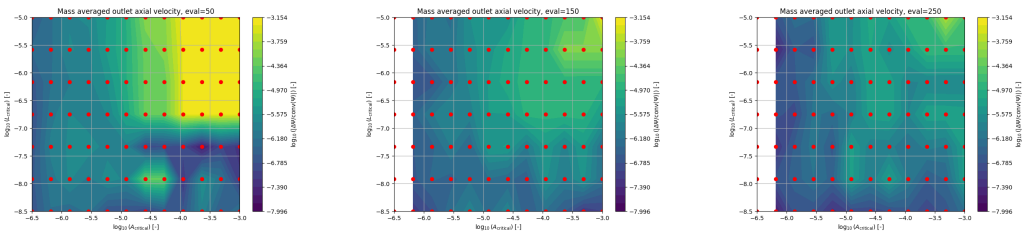



**Figure B.1:** Mass averaged outlet axial velocity for TRS1. The contours describe accuracy compared to the manually converged case. The *eval* lengths tested are 50, 150, 250.

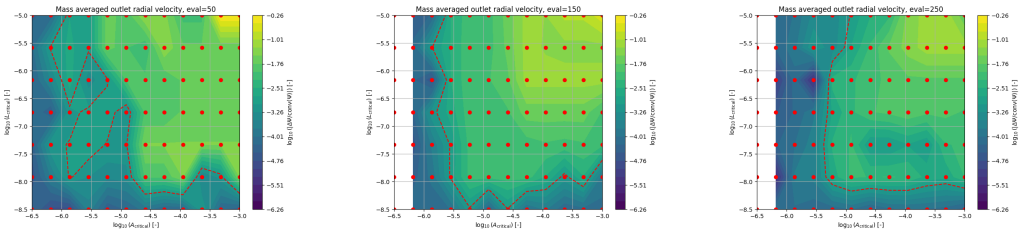



**Figure B.2:** Mass averaged outlet radial velocity for TRS1. The contours describe accuracy compared to the manually converged case. The *eval* lengths tested are 50, 150, 250. Note that although the error is high in this instance, this quantity is very close to zero and thus skews the result, as mentioned in sec 4.3.

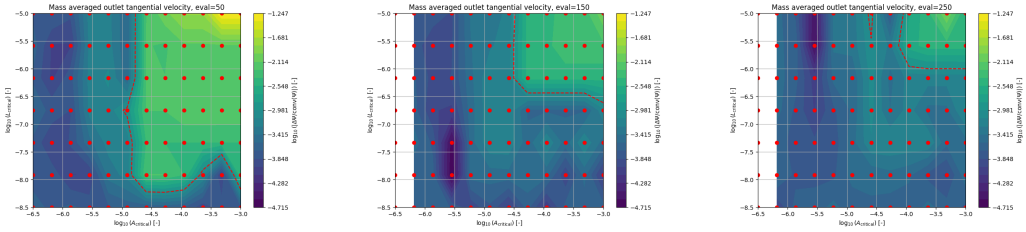



**Figure B.3:** Mass averaged outlet tangential velocity for TRS1. The contours describe accuracy compared to the manually converged case. The *eval* lengths tested are 50, 150, 250. Note that although the error is high in this instance, this quantity is very close to zero and thus skews the result, as mentioned in sec 4.3.

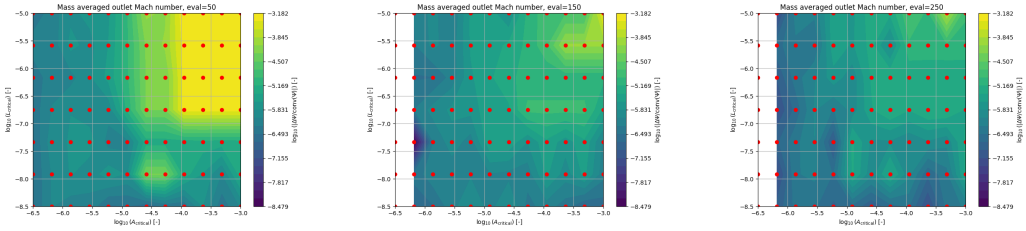



**Figure B.4:** Mass averaged outlet Mach number for TRS1. The contours describe accuracy compared to the manually converged case. The *eval* lengths tested are 50, 150, 250.

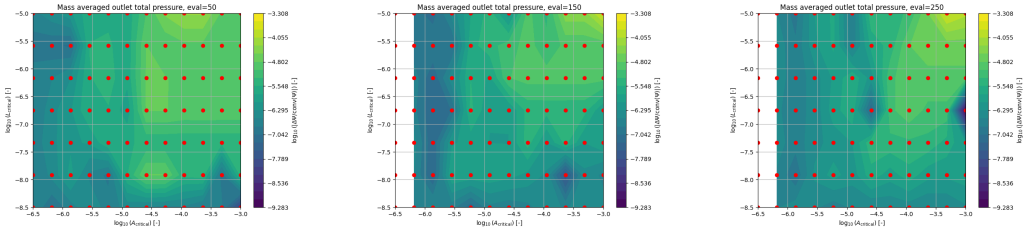



**Figure B.5:** Mass averaged outlet total pressure for TRS1. The contours describe accuracy compared to the manually converged case. The *eval* lengths tested are 50, 150, 250.

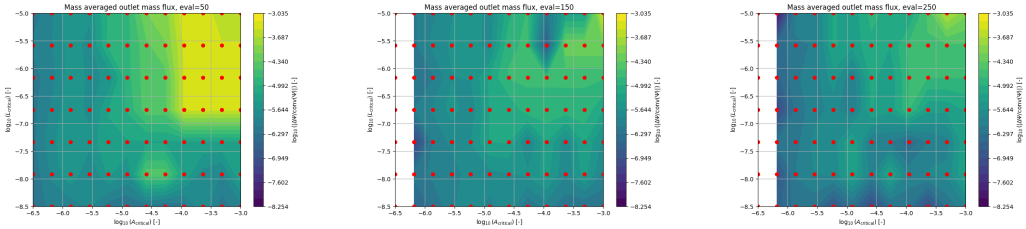



**Figure B.6:** Mass averaged outlet mass flux for TRS1. The contours describe accuracy compared to the manually converged case. The *eval* lengths tested are 50, 150, 250.

**DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY**
**CHALMERS UNIVERSITY OF TECHNOLOGY**
Gothenburg, Sweden
www.chalmers.se


CHALMERS
UNIVERSITY OF TECHNOLOGY