Extension and comparison of the Spalart-Allmaras Turbulence Model to the Spalart-Allmaras negativ Turbulence Model in OpenFOAM

Computational Fluid Dynamics (CFD) (MSK610)

Seminar work from
Gabriele Rinaldi, Urszula Starowicz,
Ursula Westarp, Jette Lorsbach

University of Stavanger

May 7, 2025

Abstract

This study focuses on the implementation and evaluation of the Spalart-Allmaras negative (SAneg) turbulence model in OpenFOAM, comparing its performance to the Spalart-Allmaras (SA) model. The SAneg model extends the original model by allowing the turbulence transport variable to take on negative values, enhancing numerical stability and capturing transitional and separated flows more accurately - particularly on more coarse meshes. The SAneg model was implemented by modifying the existing OpenFOAM SA source code, adjusting vorticity, diffusion terms, and transport equations. Both models were applied to simulate steady to unsteady, incompressible flow around a circular cylinder at low Reynolds numbers (Re = 20–180), a range that is well-studied and allows for exact comparison with existing results in the literature, especially regarding vortex shedding and boundary layer separation. Results showed that while both models produced similar drag and lift coefficients, the SAneg model provided improved predictions of wake behavior, earlier separation points, and more accurate vortex structures in transient regimes. A grid convergence study further validated the numerical setup, ensuring solution reliability. In summary, the SAneg model enables improved modelling of transitional and weakly turbulent flows. This makes the SAneg model a promising option for CFD simulations with complex flow physics or limited mesh resolution. This implementation lays the foundation for the application of SAneg in more advanced simulations.

Contents

List of Figures

List of Tables

1. Introduction

Modeling turbulent flows is one of the key challenges in computational fluid dynamics (CFD), due to the wide range of interacting scales and unsteady behaviors that affect momentum, heat, and mass transport [1]. Since Direct Numerical Simulation (DNS) is computationally expensive - especially for high Reynolds number - turbulence models such as those based on the Reynolds-Averaged Navier-Stokes (RANS) equations are widely used [2]. Among these, the Spalart–Allmaras (SA) model is particularly popular for wall-bounded aerodynamic flows, offering a balance between accuracy and efficiency through a single-equation formulation [3]. However, the standard SA model has known limitations, especially in cases involving separated or transitional flows, or when coarse meshes are used [3, 4]. To address this, the SA-negative (SAneg) model was introduced. It allows the turbulence variable to become negative in specific regions, enabling more realistic modeling of backscatter, weak turbulence, and transitional behavior [5]. It also improves numerical stability on under-resolved grids through modified diffusion and vorticity terms [4, 5]. In this project, the SAneg model is implemented in OpenFOAM by extending the standard SA formulation. Both models are evaluated and compared using the benchmark case of unsteady flow around a circular cylinder, a scenario that features well-defined vortex shedding. While the study focuses on single-phase incompressible flows, the SA-neg approach could also prove useful in more complex situations, such as multiphase flows modeled with the Volume of Fluid (VOF) method [6] or film condensation in microchannels [7, 8, 9, 10, 11, 12].
This report is organized into the following chapters. After giving an introduction, Chapter 2 gives the theoretical background needed for this project. Chapter 3 details the computational setup, including geometry, mesh, solver configurations, and boundary conditions. Chapter 4 presents the simulation results and compares the model performance, focusing on convergence, drag/lift coefficients, and wake behavior. Chapter 5 concludes with a summary of the findings and discusses potential future applications of the SA-neg model in more complex or multiphase flow scenarios.

1.1. Motivation

Turbulent flows are present in nearly all real-world fluid dynamics applications and play a central role in engineering systems like aircraft, turbines, and environmental flows [13]. Accurately modeling turbulence is therefore essential, especially under challenging conditions such as flow separation, transition, or complex geometries [14, 15]. The standard Spalart–Allmaras (SA) model, although efficient, often struggles in these regions—particularly with coarse meshes or adverse pressure gradients [16]. The SAneg model addresses these limitations by allowing negative values of the turbulence transport variable, which improves predictions in weakly turbulent or separated flows. It also improves numerical robustness, making it suitable for industrial simulations where mesh resolution is limited [16]. With CFD increasingly used in iterative design and optimization, such improvements offer a valuable balance between speed and accuracy [17]. From a broader perspective, the need for more flexible and accurate turbulence models is growing. Recent advancements in the SA model, such as the SAneg variant with rotation correction and data-driven modifications via Bayesian inference, show promise for integration into hybrid RANS-LES and transition modeling frameworks. These approaches are especially valuable for sustainability-focused simulations that require accurate and efficient modeling of complex flow environments, including renewable energy systems and pollutant transport [18, 19]. Finally, integrating the SAneg model into different CFD solvers - as demonstrated by its solver-independent implementation and data-driven calibration using high-quality experimental compressor data [19] - supports the broader goals of reproducibility, model transparency, and rigorous verification in turbulence modeling.

2. Theory and Implementation

This chapter introduces the URANS equations to be solved, the Boussinesq approach, the Spalart-Allmaras turbulence model, as well as the Spalart-Allmaras negative turbulence model and the implementation of the Spalart-Allmaras negative turbulence model in OpenFOAM.

2.1. Unsteady Reynolds averaged Navier-Stokes equations

The continuity equation and the momentum equation are required to model the fluid flow. These are the conservation equations for the mass and momentum of the fluid (Navier-Stokes equations) [20]. If additional effects such as heat transfer, compressibility, etc. of the fluid are to be taken into account, the conservation of energy must be included in the system of equations [21]. In the present work, an incompressible flow is considered, and no heat transfer or other phenomena are taken into account, for which the energy equation has to be solved. The Navier-Stokes equations can describe all transport processes in laminar and turbulent flow. However, in most cases the Navier-Stokes equations cannot be solved analytically because the inertia terms are non-linear. The equations can only be solved analytically with extensive simplifications [22]. In order to describe complex fields, the so-called Reynolds-averaged Navier-Stokes equations (RANS) are introduced in the following.
To obtain the RANS equations, the state variables in the Navier-Stokes equations are replaced by

$$\phi(x,y,z,t) = \bar{\phi}(x,y,z,t) + \phi'(x,y,z,t) \tag{2.1}$$

and then a temporal averaging of the Navier-Stokes equations is carried out [23]. Here ϕ is an arbitrary state variable, $\bar{\phi}$ is a time-averaged value and ϕ' is a random fluctuation variable. To obtain the unsteady Reynolds averaged Navier-Stokes (URANS) equations, the state variable $\bar{\phi}$ is determined by ensemble averaging rather than time averaging as for the RANS equations [22]. The rest of the derivation is identical to the derivation of the RANS equations and is shown in [22]. The RANS/URANS equations are formally distinguished from the Navier-Stokes equations by the terms $-\rho\overline{v_i'v_j'}$ in the time-averaged momentum equation. These non-linear terms are called Reynolds stresses, and because of them the RANS/URANS equations are not closed. To solve this problem, a turbulence model is needed to close the set of equations [22].

2.2. Boussinesq hypothesis

To model the Reynolds stresses and close the system of equations, most zero-, one-, and two-equation turbulence models use the Boussinesq hypothesis. The Spalart-Allmaras turbulence model considered in this work also uses this hypothesis to approximate the Reynolds stresses. For this reason, the Boussinesq hypothesis is briefly presented below.
The Boussinesq hypothesis

$$-\rho\overline{v_i'v_j'} = \nu_t\left(\frac{\partial\bar{v}_i}{\partial x_j} + \frac{\partial\bar{v}_j}{\partial x_i}\right) - \frac{2}{3}\left[\rho k + \nu_t\frac{\partial\bar{v}_i}{\partial x_i}\right]I$$
$$with\ k = \frac{1}{2}(\overline{v_x'v_x'} + \overline{v_y'v_y'} + \overline{v_z'v_xz}) \tag{2.2}$$

is based on the assumption that the Reynolds stresses are proportional to the velocity gradients in the flow [22]. In equation 2.2, μ_t describes the eddy viscosity. The eddy viscosity ν_t is not a quantity of material, but a quantity that depends on the structure of the turbulence [3]. The determination of eddy viscosity ν_t differs from turbulence model to turbulence model,

and accordingly the modeling of Reynolds stresses differs between different turbulence models based on the Boussinesq hypothesis [22].

2.3. Spalart-Allmaras turbulence model

As mentioned above, a turbulence model is required to solve the URANS equations. In the present work, the one-equation Spalart-Allmaras (SA) turbulence model is used in its original formulation and in a modified formulation. The original formulation is given below.
In the SA turbulence model, the eddy viscosity ν_t is given by the equation

$$\nu_t = \tilde{\nu}\frac{\chi^3}{\chi^3 + c_{v1}^3},\ \chi = \frac{\tilde{\nu}}{\nu} \tag{2.3}$$

where c_{v1} is a constant and ν is the kinematic viscosity [3]. In the equation 2.3, $\tilde{\nu}$ is the working variable for which the SA turbulence model introduces a transport equation as follows [3]:

$$\frac{D\tilde{\nu}}{Dt} = \underbrace{c_{b1}\tilde{S}\tilde{\nu}}_{Production} \underbrace{- c_{w1}f_w\left[\frac{\tilde{\nu}}{d}\right]^2}_{Destruction} + \frac{1}{\sigma}\left[\nabla\cdot((\nu+\tilde{\nu})\nabla\tilde{\nu}) + c_{b2}(\nabla\tilde{\nu})^2\right] \tag{2.4}$$

Here c_{b1}, σ, c_{b2} and c_{w1} are constants that are defined in [3]. d is the distance to the nearest wall, $\tilde{S}$ is the modified vorticity, and f_w is a function. $\tilde{S}$ and f_w are defined in [3]. By solving equation 2.4, the working variable $\tilde{\nu}$ is determined. With $\tilde{\nu}$ the eddy viscosity ν_t is calculated and then the Reynolds stresses are approximated using the Boussinesq hypothesis.
The SA turbulence model is good at predicting stalled flows, which is useful in aerofoil applications. It also has only one extra transport equation to solve and therefore has a lower computational time compared to two-equation models. A disadvantage of the SA turbulence model is the need to define a length scale for predicting the dissipation of the transported quantity. For more complex geometries, this is a challenging task [13].

2.4. Spalart-Allmaras negative turbulence model

In 2012, the Spalart-Allmaras (SA) negative turbulence model is presented by S. R. Allmaras, F. T. Johnson and P. R. Spalart in [5]. Like the SA turbulence model, it is a one-equation model. The modifications made to the SA turbulence model are intended to improve the ability of the turbulence model to handle under-resolved grids and unphysical transient solutions [5]. The formulation of the SA negative model only applies when $\tilde{\nu}$ becomes negative. Otherwise, the formulation of the SA turbulence model is used within the SA negative turbulence model. In the present work, the SA turbulence model already implemented in OpenFOAM is extended to the SA negative turbulence model. In the following, the SA negative turbulence model is presented and the terms added to the OpenFOAM code are introduced.
The new transport equation for the working variable $\tilde{\nu}$ is

$$\frac{D\tilde{\nu}}{Dt} = \underbrace{P_n}_{Production} - \underbrace{D_n}_{Destruction} + \frac{1}{\sigma}\nabla\cdot[(\nu+\tilde{\nu}f_n)\nabla\tilde{\nu}] + \frac{c_{b2}}{\sigma}(\nabla\tilde{\nu})^2\ . \tag{2.5}$$

Equations 2.4 and 2.5 differ in their form only by the term f_n. $\nu+\tilde{\nu}$ in the SA turbulence model is the diffusion coefficient. This value should not be negative in order to maintain the continuity condition C^1 and to avoid negative minima in a stationary solution [5]. To achieve this, the diffusion coefficient in the SA negative turbulence model is modified to $\nu+\tilde{\nu}f_n$ using the term

$$f_n = \frac{c_{n1}+\chi^3}{c_{n1}-\chi^3}\ . \tag{2.6}$$

c_{n1} is a constant and is defined in [5]. In [5], the production and destruction terms in the SA negative turbulence model are re-formulated and also modified for the SA turbulence model. The production and destruction terms in the SA turbulence model are modified with the laminar suppression term $f_{t2} = c_{t3}exp(-c_{t4}\chi^2)$ to

$$P = c_{b1}(1 - f_{t2})\tilde{S}\tilde{\nu},\ D = \left(c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right)\left[\frac{\tilde{\nu}}{d}\right]^2 . \tag{2.7}$$

The reason for the modification is the prevention of unwanted growth of small $\tilde{\nu}$ in the presence of vorticity [5]. If the SA negative turbulence model is used, the terms P_n and D_n are calculated as follows [5]:

$$P_n = c_{b1}(1 - c_{t3})S\tilde{\nu},\ D_n = -c_{w1}\left[\frac{\tilde{\nu}}{d}\right]^2 . \tag{2.8}$$

In the original formulation it is possible that the value of the vorticity $\tilde{S}$ becomes negative. This is not physical and can lead to problems with the formulation of other terms in the model [5]. This is prevented by the following modification of the vorticity

$$\bar{S} = \frac{\tilde{\nu}}{\kappa^2 d^2}f_{v2} \tag{2.9}$$

$$\tilde{S} = \begin{cases} S + \bar{S} & : \quad \bar{S} \geq -c_{v2}S \\ S + \dfrac{S(c_{v2}^2 S + c_{v3}\bar{S})}{(c_{v3} - 2c_{v2})S - \bar{S}} & : \quad \bar{S} < -c_{v2}S \end{cases} \tag{2.10}$$

presented in [5].
The terms presented here for the modification of the SA turbulence model and its extension to the SA negative model have been implemented in OpenFOAM. The implementation is described in the next chapter.

2.5. Implementation of the Spalart-Allmaras negative turbulence model

The aim of this work is to add the SA negative turbulence model as *negSpalartAllmaras* to the existing library. For this purpose, the SA turbulence model already implemented in OpenFOAM is copied into a private directory and renamed to *negSpalartAllmaras*. Furthermore, all classes named *SpalartAllmaras* are renamed to *negSpalartAllmaras*. See [24] for the underlying source code. Appendix A shows the folder structure. The paths for the compiler to the required files are specified within the *lnInclude* folders. The *.H* and *.C* files that implement the new model are located in the *negSpallartAllmaras* folder. Appendix B contains the code for the *negSpalartAllmaras.H* file and Appendix C contains the code for the *negSpalartAllmaras.C* file. The additional model constants are initialised in the *negSpalartAllmaras.H* file. The rest of the code in *negSpalartAllmaras.H* is unchanged from the underlying source code. The first change made to *negSpalartAllamras.C* is to change the modified vorticity $\tilde{S}$ according to equation **??**. See figure C.2 for the implementation. The functions *pos()* and *neg()* can take the value zero or one, thus allowing to distinguish between the two formulations of the modified vorticity $\tilde{S}$. Assigning values to the newly initialised model constants is the next change to the underlying code. See figure C.5 for the implementation. The diffusion coefficient is modified with the function f_n, see equation 2.6. The function is called *DnuTildaEff* in the code and the implementation is shown in figure C.7. Within the *correct()* function, the vorticity S is defined, which is required to implement the SA negative turbulence model. The new transport equation is then added according to equation 2.5. The distinction between the two models is made using the functions *pos()* and *neg()*. The changes described here are all the changes that have been made to the code. In order to compile the new turbulence model, a *files* and an *options* file are required in the *Make*

folder. These are shown in Appendix D. To add the new turbulence model to the library, the file *myTurbulentTransportModels.C* is required. The code is shown in Appendix E. To add the new model to a simulation, the line *libs ("libmyIncompressibleTurbulenceModels.so");* must be added to the end of the *controlDict* file. The model can then be called up as *negSpalartAllmaras* in the *turbulenceProperties* file.

3. Computational Setup

For the simulation, OpenFOAM-v2406 was used as the solver, FileZilla was used to transfer files to and from the server, and Windows PowerShell was used to implement and run the code, while ParaView 5.13 was used for post-processing. We used two preexisting codes as starting points for our implementation: the turbulent flow on a square cylinder[25], which provided the base case, and the laminar flow on a circular cylinder[26], from which we obtained the predefined mesh.

3.1. Choice of the solver

We considered a flow of pure air at standard conditions and defined the inlet velocity as $U = 0.1332m/s$, the diameter of the cylinder $D = 0.02m$ and the kinematic viscosity as $\nu = 1.48 * 10^{-}5m^2/s$. These values indicate that the flow regime is characterized by a Kármán vortex street and that the flow can be assumed incompressible. Since the Kármán vortex region is unsteady, we simulated the flow as transient. For these reasons, we used the solver pimpleFoam, which is based on the PIMPLE algorithm. We set the simulation time of $50s$ with a time step of $\Delta t = 0.005$, which is sufficient to capture the development of the Kármán vortices while keeping the run time low.

3.2. Geometry and Domain

The geometry is a two-dimensional circular cylinder, a common model for turbulent flows, especially for analyzing the behavior of the Kármán vortex street. The width of the domain is 20 cm, and the height is 50 cm. A large flux is needed to accurately model the Kármán vortex street, in particular we need $l/d > 7$ [27], where l is the width of the domain and d the diameter of the cylinder, so we set the diameter of the cylinder d=2cm. The simulation domain is defined by four boundaries, which will be discussed in Section 3.4:

1. inlet (left side);
2. outlet (right side);
3. walls (upper and lower parts);
4. obstacle (cylinder);

Additionally, the front and back boundaries of the control volume are treated as empty since the simulation is in 2-D.

3.3. Mesh and Discretizations

The mesh was generated using the BlockMesh command in OpenFOAM and consists of three main regions:

1. pre-block: located to the left of the cylinder, it represents the inlet section.
2. cylinder-blocks: composed of four radial blocks arranged around the cylinder. To accurately model the boundary layer, a y+ value lower than 1 would be required. In our case, this condition is not always met; however, since the purpose of the analysis is not to evaluate the boundary layer, we kept the cells wider to reduce computational time. Additionally, in the left and right blocks, we increased the number of cells in the y-direction from 50 to 70 to achieve better results.

3. post-block: located to the right of the cylinder. In this region, the Kármán vortex street is well-defined and begins to dissipate before reaching the outlet point.

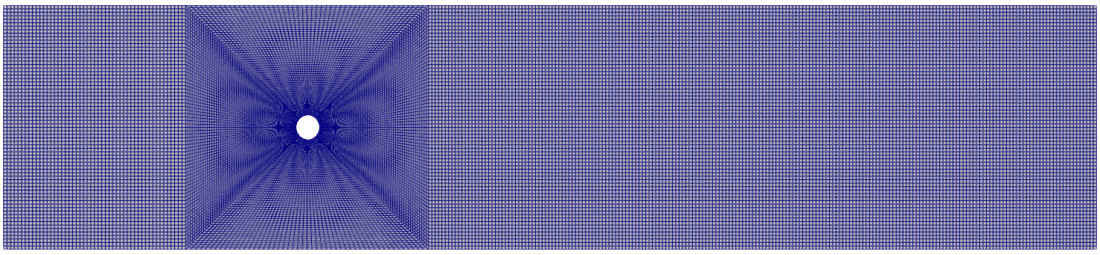

Figure 3.1.: Mesh of the system

Table 3.1.: Mesh dimensions and number of cells for each block

Block	Length (x) [m]	Cells (x)	Height (y) [m]	Cells (y)
Pre-block	15	50	20	70
Cylinder-block left	10	50	20	70
Cylinder-block bottom	20	50	10	50
Cylinder-block right	10	50	20	70
Cylinder-block top	20	50	10	50
Post-block	55	200	20	70

3.4. Grid Convergency Study

The Grid Convergence Index (GCI) study was set in this project based on three points - for the future works it could contain more to achieve more precise results. However, using the methodology presented in [28], which builds upon the foundational work by Roache [29], we decided that three points would be satisfactory.

Table 3.2.: GCI Calculation Data from the CFD Mesh Study

Number	**Mesh**	**N**	**Mean drag coefficient**	**h**
1	Fine	70 000	1.194 84	$1.511\,90 \times 10^{-2}$
2	Medium	17 500	1.219 49	$7.559\,00 \times 10^{-3}$
3	Coarse	4375	1.257 65	$3.779\,60 \times 10^{-3}$

Above in the table 3.2 was shown the values used for the study. The GCI values were then computed for both the coarse-to-medium and medium-to-fine mesh transitions. For the transition from the medium to fine grid, the GCI was approximately (4.99%), which is within an acceptable range for engineering analyses and parametric studies, according to Teschner's instruction [30].

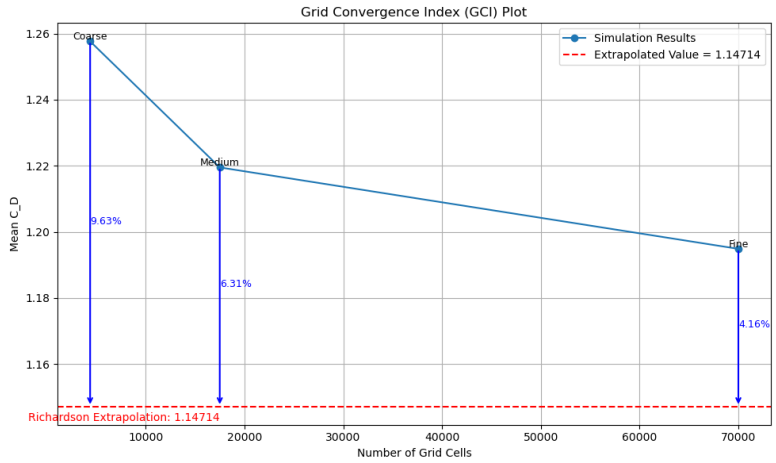

Figure 3.2.: Grid Convergence Index

The figure 3.2 illustrates the mean drag coefficient values at a chosen point with different numbers of grid cells to achieve increasing grid resolution. The Richardson extrapolated value is shown as a red dashed line at $C_{D,\text{extrapolated}} = 1.14714$. The plot also displays the computed relative error values between each grid and the extrapolated value.
In our case, the computed value of the convergence ratio $CR \approx 0.65$ lies within the interval $0 \leq CR < 1$, indicating monotonic convergence. This behavior confirms that the numerical solution exhibits grid independence, thereby validating the reliability of the simulation results.

3.5. Flow properties

Since the flow is Newtonian and incompressible, the cinematic viscosity ν and density ρ are fixed. As initial conditions we defined:

Table 3.3.: Initial conditions

Quantity	Value	Unit
ν	1.48×10^5	m^2/s
$\mathbf{U}$	0.1332	m/s
p	0	Pa
$\tilde{\nu}$	4.0×10^5	m^2/s
ν_t	1.0×10^5	m^2/s

3.6. Boundary conditions

The boundary conditions of U, p, nuTilda and nut are shown in the Table 3.4. At the inlet we defined the value of the velocity $U = 0.1332 m/s$ and the pressure equal to zero. At the outlet we defined the velocity with zeroGradient and the pressure at the same value of the inlet.

Table 3.4.: Boundary conditions for each field at different patches

Field	Inlet	Outlet	Wall
U	`freestreamVelocity`	`zeroGradient`	`slip`
p	`freestreamPressure`	`freestreamPressure`	`zeroGradient`
$\tilde{\nu}$	`freestream`	`freestream`	`fixedValue uniform 0`
ν_t	`freestream`	`freestream`	`fixedValue uniform 0`

Field	Obstacle	Front/Back
U	`noSlip`	`empty`
p	`zeroGradient`	`empty`
$\tilde{\nu}$	`fixedValue uniform 0`	`empty`
ν_t	`nutUSpaldingWallFunction uniform 0`	`empty`

3.7. Numerical Scheme

The numerical schemes used in the simulation are defined in the fvSchemes file. Since the simulation models an unsteady flow using the pimpleFoam solver, the time scheme was set to Euler. Euler is a first-order implicit time integration method suitable for transient simulations. The gradient terms were calculated using the Gauss linear scheme, which employs linear interpolation and the Gauss theorem. For the velocity gradient grad(U), the cellLimited Gauss Linear scheme with a limiter value of 1 was applied. This ensures boundedness and improves the stability especially in areas of the mesh with coarser resolution. Divergence terms were handled individually within the divSchemes category. For the advective terms (e.g. div(phi,U)), the linearUpwindV scheme was used. This is a directionally limited higher-order scheme, well-suited for flows with dominant directional components. For the turbulence-related terms, such as div(phi,nuTilda) and div(phi,k), the upwind scheme was employed to ensure stability. The interpolation scheme was set to linear, the surface normal gradient scheme to corrected, and the Laplacian terms was handled using the Gauss linear corrected scheme, which improves accuracy in non-orthogonal meshes and is generally recommended for OpenFOAM simulations. These choices were made to support numerical stability, especially in under-resolved mesh regions, which is where the SA-neg model is expected to perform better than the standard SA model.

3.8. Solver and Solution Control

In this project, the transient solver pimpleFOAM from OpenFOAM was used, which is based on a combination of the PISO and SIMPLE algorithms. The outer corrector loop (PIMPLE) ensures numerical stability and convergence for each time step. This is particularly helpful in simulations which relatively large time steps or complex turbulence modeling. For the linear system solution, the following solvers were used:
The GAMG solver was chosen for pressure due to its efficiency in solving large, spare systems. For the remaining equations, the smoothSolver was applied in combination with Gauss-Seidel smoothing to enhance convergence. This approach helps reduce residual errors and provides stability, particularly when dealing with under-resolved mesh regions. The under-relaxation factors used in the simulation are:
In addition to the selection of suitable numerical solvers, the control of the solution steps is also crucial for the stability and convergence of the simulation process. A central tool for this is under-relaxation. In general, under-relaxation stabilizes the solution process by restricting how much a variable is allowed to change between iterations. This can be achieved either through adjustments to the solution matrix and source terms before solving, or by directly altering the

Table 3.5.: Field-specific solvers used in the simulation.

Field	Solver
U	`smoothSolver`
p	`GAMG`
$\tilde{\nu}$	`smoothSolver`
ν_t	`smoothSolver`

Table 3.6.: Chosen Relaxation Factors

Relaxation Factor	Value
U	0.7
$\tilde{\nu}$	0.7

field values. An under-relaxation factor $a, 0 < a \leq 1$ specifies the amount of under-relaxation.

- No specified a: no under-relaxation
- a = 1: guaranteed matrix diagonal equality / dominance
- a decreases, under-relaxation increases
- a = 0: solution does not change with successive iterations.

In this simulation, both the velocity field U and nuTilda were assigned a relaxation factor of 0.7. This value represents a moderate level of under-relaxation, which provides a good balance between numerical stability and convergence speed.

3.9. Function Objects

Function objects are used in OpenFOAM to extract specific information during the simulation or to carry out the wanted analysis.

3.9.1. Residuals

Residuals were monitored throughout the simulation using the residuals function object. These values provide insight into the convergence behavior of the solver by tracking the decrease in error between iterations.

3.9.2. yPlus

To assess the quality of the mesh in the near-wall region, the yPlus function object was integrated into the simulation setup. This function calculates the turbulence-related yPlus values adjacent to wall boundaries, which serve as an indicator for whether the mesh resolution is adequate in those areas. In order to tailor the output to the analysis needs, optional parameters were added to the corresponding yPlus configuration file, ensuring that the generated .dat file contained all relevant data. The yPlus value is assumed to be less than or equal to 1, as in the literature [13]. Care was taken during evaluation to ensure that the value yPlus is less than or equal to 1 so that the boundary layer is recognized with sufficient accuracy.

3.9.3. Force Coefficients

The forces function object was used to calculate the forces acting on the cylinder. The drag coefficient and the lift coefficient were derived. These coefficients were monitored over time to observe their periodic behavior. The time-averaged values and fluctuations of these coefficients are also used later for a model comparison in Chapter 4.

3.9.4. Streamlines

In order to visualize streamlines during post-processing, the streamLine funtion object was included in the system/controlDict.The function type was defined as streamLine, and the corresponding library fieldFunctionObjects was specified via libs entry. All other required parameters, such as the selection of fields remained unchanged from the default configuration. This setup enables the automated generation of streamline data for visualization in ParaView, based on the velocity field computed during the simulation.

3.10. Post Processing

The ParaView, Microsoft Excel and Python were used in this project to evaluate and visualize the simulation results. The results were physically interpreted with the help of OpenFOAM. ParaView was primarily used to visualize the flow fields. Among other things, the pressure and velocity distributions and the turbulence variables such as the turbulence viscosity field were visualized. In addition, streamlines isosurfaces and sectional views could be generated with ParaView in order to analyze certain flow phenomena in more detail. Excel was used for the quantitative analysis. In particular for analyzing the residuals and the force coefficients (drag and lift coefficients). For this purpose, the log and CSV files generated from OpenFOAM were further processed and converted into diagrams. A Python script was made to visualize the Grid Convergence Index (GCI).

4. Results and Discussion

Steady-state simulations at low Reynolds numbers offer a controlled setting to evaluate turbulence models. This study uses Re = 20, a well-documented laminar regime where flow around a circular cylinder remains symmetric and free from vortex shedding. This enables direct comparison with benchmark data (e.g., Tritton 1959 [31], Abane 1978 [32]) and avoids complications from unsteady behavior, which typically arises beyond Re ≈ 54 [33].
By assessing the Spalart–Allmaras (SA) and SA-neg models - including results from NASA studies [34] - under steady conditions, we isolate key flow features like separation points and drag, and examine model accuracy, pressure recovery, and near-wall behavior. Since these models are often tuned for high Re, running them at Re = 20 also reveals their limitations in low-Re applications. This setup thus provides a reliable benchmark for analyzing convergence, residuals, and turbulence model performance before addressing more complex flows. In the following residuals, the drag coefficients (C_D) and lift (C_L) coefficients and the separation angle within the transient state are analyzed.

4.1. Analysis of the residuals in steady and transient state

The analysis of residuals is a fundamental step in evaluating the convergence and reliability of solutions in computational fluid dynamics. Residuals represent the imbalance in discretized equations, and their reduction is important to ensure that the numerical solution accurately represents the actual physical behavior of the flow field [13].
Figures 4.1 and 4.2 show the residuals for the laminar and transient flow simulations, respectively. In both cases, the residuals decrease over time, indicating convergence of the numerical solution.

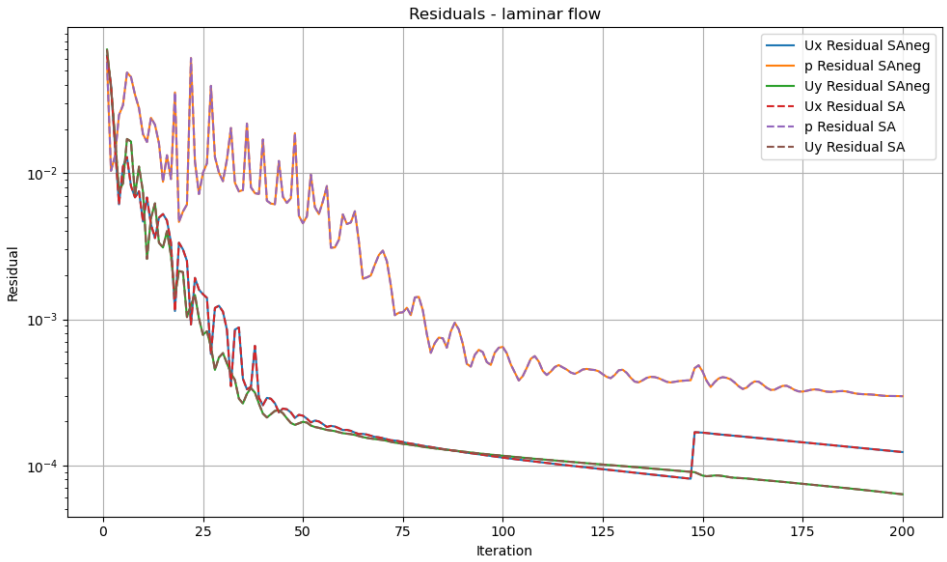

Figure 4.1.: Residuals for laminar flow simulation using SA and SAneg turbulence models.

For the laminar flow, residuals for velocity components (U_x, U_y) and pressure (p) steadily decrease, with oscillations appearing in early iterations, in practicly the same way for the

SAneg model and SA model. After around 50 iterations, a smoother convergence is observed, reaching residuals of the order 10^{-4}.

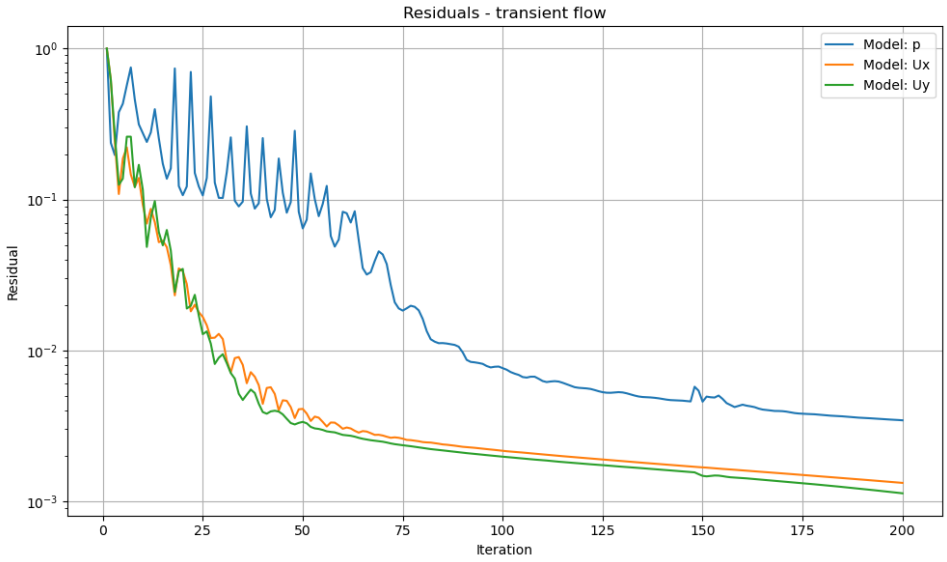

Figure 4.2.: Residuals for transient flow simulation.

In the transient simulation, pressure residuals exhibit larger initial fluctuations and slower convergence compared to velocity components. U_x and U_y residuals show a consistent and smooth decay, achieving residuals below 10^{-3} by 200 time steps.
According to the ANSYS CFX documentation, residual levels larger than 10^{-4} may be sufficient to obtain a qualitative understanding of the flow field, with 10^{-4} considered relatively loose convergence, and 10^{-5} deemed good convergence for most engineering applications [35]. Therefore, in our simulations, residuals reaching levels of 10^{-3} to 10^{-4} are deemed acceptable, provided that key quantities of interest, such as drag and lift coefficients, have stabilized and exhibit minimal variation with further iterations.

4.2. Analysis of drag and lift coefficient

The analyze of the drag (C_D) and lift (C_L) coefficients was carried out and out of it obtained from numerical simulations at low Reynolds numbers. These aerodynamic coefficients provide crucial insights into the flow behavior around the geometry and serve as benchmarks for validating the accuracy and stability of the implemented turbulence models.
Figure 4.3 illustrates the drag and lift coefficients obtained using the standard Spalart-Allmaras (SA) model and its modified also negative version (SAneg). Both models computed a drag coefficient (C_D) close to 1.5, in good agreement with empirical data. The lift coefficient (C_L) remains near zero for both cases, as expected due to the symmetry of the geometry and flow conditions.

4.2.1. Analysis of the drag coefficient

Abane's 1978 [32] study investigated Oseen flow around a circular cylinder at low Reynolds numbers, employing Faxén series and the method of least squares to calculate drag coefficients [32, 36, 37]. Given the result of the drag coefficient 2.33 at Re = 20 using the SA negative model, these values are higher than Abane's calculated drag coefficient of 1.86 at Re = 20 [32].

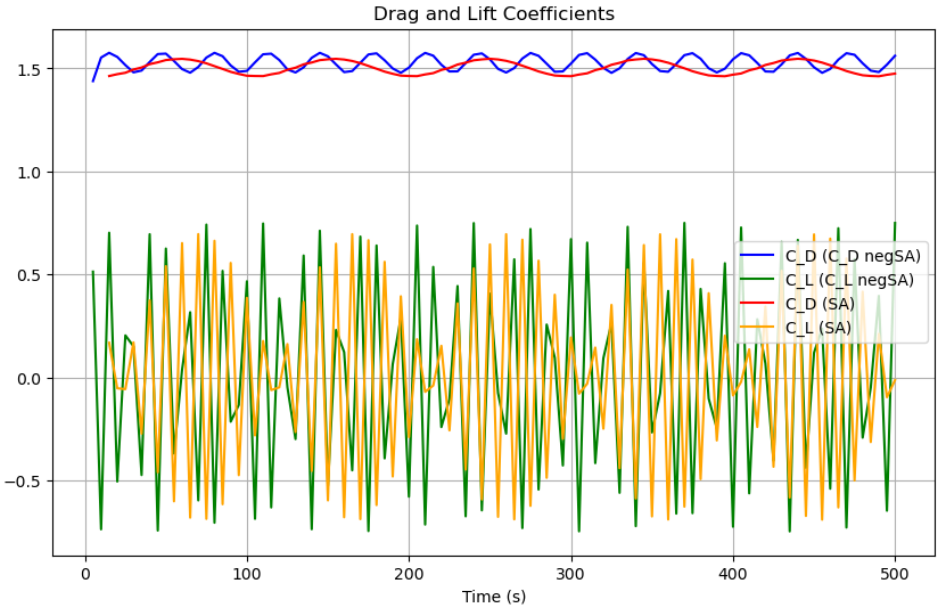

Figure 4.3.: Drag and Lift coefficient

Especially that in Tritton's (1959) [31] experimental study investigated the drag on a circular cylinder across a range of low Reynolds numbers. According to the published data, the drag coefficient at Reynolds number Re = 20 differs between approximately $C_D = 2.23$ [31] and $C_D = 1.32$ [38]. In comparison, our numerical results using the SA-negative turbulence model yielded close value of the drag coefficients. Next study used for the comparison of the results is *Numerical solutions for steady flow past a circular cylinder at Reynolds numbers up to 100* by Dennis [36], where drag coefficient was 2.045. Finally, there was fount the study

Table 4.1.: Calculated drag coefficients C_D at various Reynolds numbers

C_D - Park	C_D - Schlichting	C_D - SA	C_D - SA neg	C_D - Abane	C_D - Dennis	C_D - Tritton
1.32	1.5	1.5	1.53	1.86	2.045	2.23

The calculated drag coefficients show good overall agreement with values reported in the literature, although some discrepancies are observed. The drag coefficient obtained using the Spalart-Allmaras model with negative C_D correction (1.53) and the standard already implemented SA model (1.5) closely match the empirical value reported by Schlichting (1.5), suggesting that the turbulence model performs reasonably well.
The discrepancy may be attributed to multiple factors. The experimental measurements inherently capture additional physical effects such as three-dimensional disturbances, end effects, and slight imperfections in flow symmetry, which are often absent in idealized two-dimensional simulations.
Nonetheless, the achieved values of the drag coefficient with mesh refinement in our simulation demonstrates convergence and good prediction. Moreover is close to the values from the literature.

4.2.2. Analysis of lift coefficient

The lift coefficient obtained from the simulation at Re = 20 was compared to reference values reported in Tritton [31], Abbasi [33] and Park [38]. The lift force is negligible in this low Reynolds number regime due to the symmetric nature of the flow around the cylinder. This is consistent with the present simulation results, which yielded small, values fluctuating near zero. These minor variations are likely due to transient effects or numerical asymmetries, and do not indicate the presence of any sustained lift force. Therefore, the results are in good agreement with the expected behavior.

4.3. Analysis of Wake Behavior and Boundary Layer Detachment

In order to understand the performance of the turbulence models beyond numerical indicators such as residuals and drag coefficients, it is important to investigate the wake evolution and boundary layer behaviour around the selected structure. In this chapter, the Spalart-Allmaras and negative Spalart-Allmaras models are compared in terms of flow separation and vortex shedding. The focus is on how the wake develops over time and how early or late the flow separation takes place.

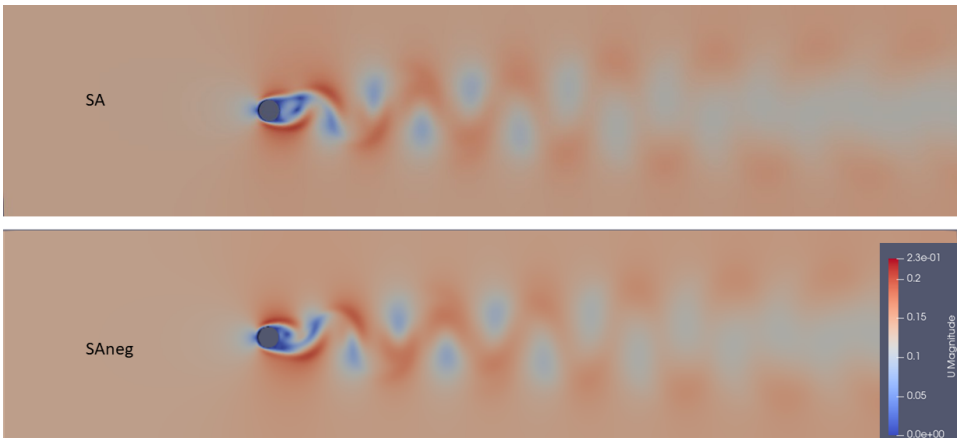

Figure 4.4.: Transient flow field around cylinder, Re=180

The figure 4.4 represents a clearly unsteady wake with classic Kármán vortex shedding, occuring at a Reynolds number within the transitional range (e.g. Re = 80 − 300 [39]). Here the differences between SA and SAneg become more pronounced. The SAneg simulation exhibits well-developed vortex structures with stronger velocity gradients. The blue and red regions alternate clearly in the wake, indicating strong unsteady behavior. The SA model however shows a more smeared wake where the vortex cores quickly diffuse. This supports the findings of Fornberg [40] and Dennis and Chang [36], who both analyzed the limitations of traditional steady-state models in capturing realistic vortex dynamics up to the transitional Re number range. Fornberg [40] demonstrated that at Reynolds numbers beyond 180, the wake becomes unsteady, forming a periodic vortex street with distinct, coherent structures [39], and Dennis and Chang [36] showed that attempting to simulate steady solutions beyond Re $\approx$ 70 results in unrealistic wake characteristics.
When comparing the results of figure 4.4 to the findings by Griffin [41], the sharper vortex defintion and the reduced distance from the cylinder to the onset of vortx shedding correlates well with the here presented SAneg model. Griffin's hot-wire measurements show that accurate resolution of natural shedding leads to a shorter vortex formation length [41]. These results highlight that figure 4.4 demonstrates the SAneg model's ability to represent unsteady vortex shedding and downstream wake development. The improved agreement with literature

underlines that SAneg is better suited for capturing transitional flow behavior around bluff bodies. The flow simulation at $Re = 20$, which was originally intended for comparison, could not be converged with either the SA or the SAneg model and is therefore not considered in this analysis.

To describe the wake behavior better, the wall-shear-stress behind the pile was simulated in order to investigate the separation process in detail, shown in the figures 4.5 and 4.6. This step is particularly important because the location and characteristics of flow separation influence the wake structure, the formation of vortices, and ultimately the forces acting on the cylinder. As shown in previous analyses (Section 4.2), the vortex shedding and drag behavior undergo changes as the Reynolds number increases. To better understand the root cause of these changes, a closer look at the boundary layer detachment becomes necessary.
The figures 4.5 and 4.6 present the distribution of the wall-shear-stress magnitude (τ_m) around the cylinder at Re = 180. This parameter (τ_m) is a direct indicator of boundary layer behavior, especially useful for identifying the separation point. Physically, the τ_m represents the frictional force per unit area exerted by the fluid at the surface of the cylinder [39]. As the boundary layer develops and experiences a negative pressure gradient, the fluid slows down near the surface of the pile. Once the velocity gradient in the flow direction at the wall becomes zero, τ_m drops to a minimum, typically close to zero, indicating the point of flow separation [39]. The τ_m is mathematically defined as follows:

$$\tau_w = \mu \left(\frac{\partial u}{\partial y} \right)_w = 0 \quad \text{(Separation)}. \tag{4.1}$$

The equation contains the wall shear stress τ_w, which depends on the dynamic viscosity μ and the velocity gradient $\partial u/\partial y$ at the wall [39].

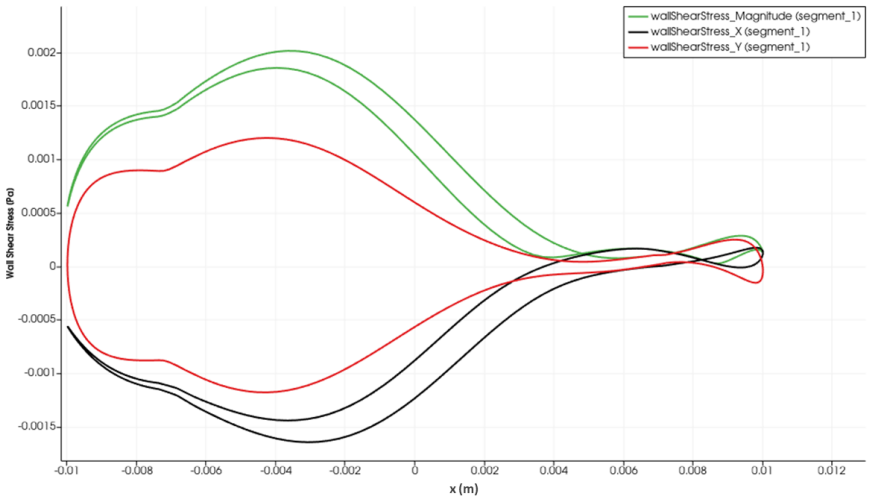

Figure 4.5.: SA model, wall shear stress (τ_m) distribution, Re=180, t=200s.

This numerical behavior is in strong agreement with Dennis and Chang [36], who observed that the separation point corresponds to the angle along the cylinder where τ_m reaches a minimum. In 4.5, the shear stress distribution indicates a single, well-defined separation point on either side of the cylinder, with a therefore symmetric wake, suggesting the presence of one dominant recirculation region. In contrast, 4.6 reveals a more complex pattern: multiple regions of near-zero wall shear stress can be observed downstream of the cylinder, implying the formation of several discrete separation eddies. This aligns with the findings of Dennis and Chang [36], who observed that at higher Reynolds numbers, the flow becomes increasingly

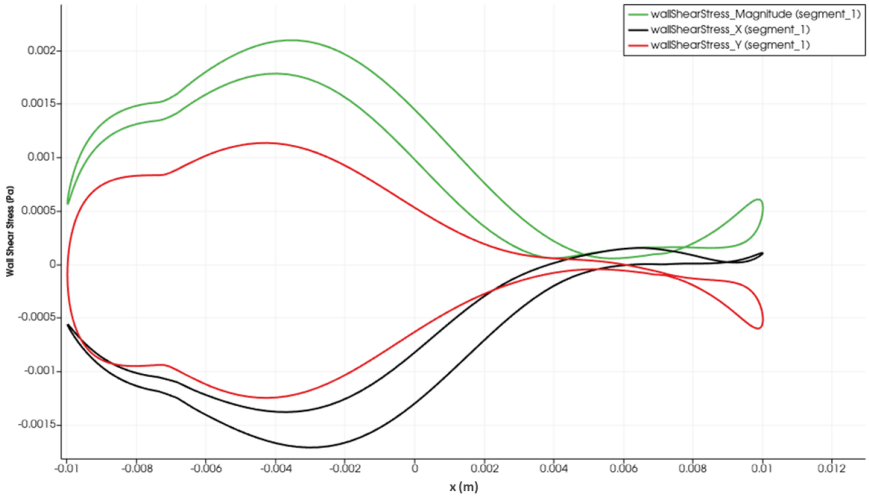

Figure 4.6.: SAneg model, wall shear stress (τ_m) distribution, Re=180, t=500s.

unsteady, and the wake elongates with more spatial complexity-including the appearance of multiple vortical structures. The multiple minima in the τ_m in figure 4.6 mark successive detachment points where additional eddies are shed into the wake, a behavior characteristic of the transition toward vortex street formation.

After the qualitative observation of vortex shedding in the wake, the following will analyze the wake length, defined as the distance from the rear of the cylinder to the reattachment point where streamlines begin to align with the free stream again. In the present simulation (see Figure 4.7), the wake region extends approximately 6 to 7 cylinder diameters downstream before the vortical structures become fully convected and the streamlines regain a mostly parallel alignment with the main flow. The first vortex detaches just behind the cylinder, and the regular alternating pattern of vortices persists with nearly constant spacing. Based on this observed wake structure, we estimate the recirculation zone i.e., the region enclosed by the first pair of symmetric vortices and reverse flow, as approximately 2 to 2.5 diameters in length.

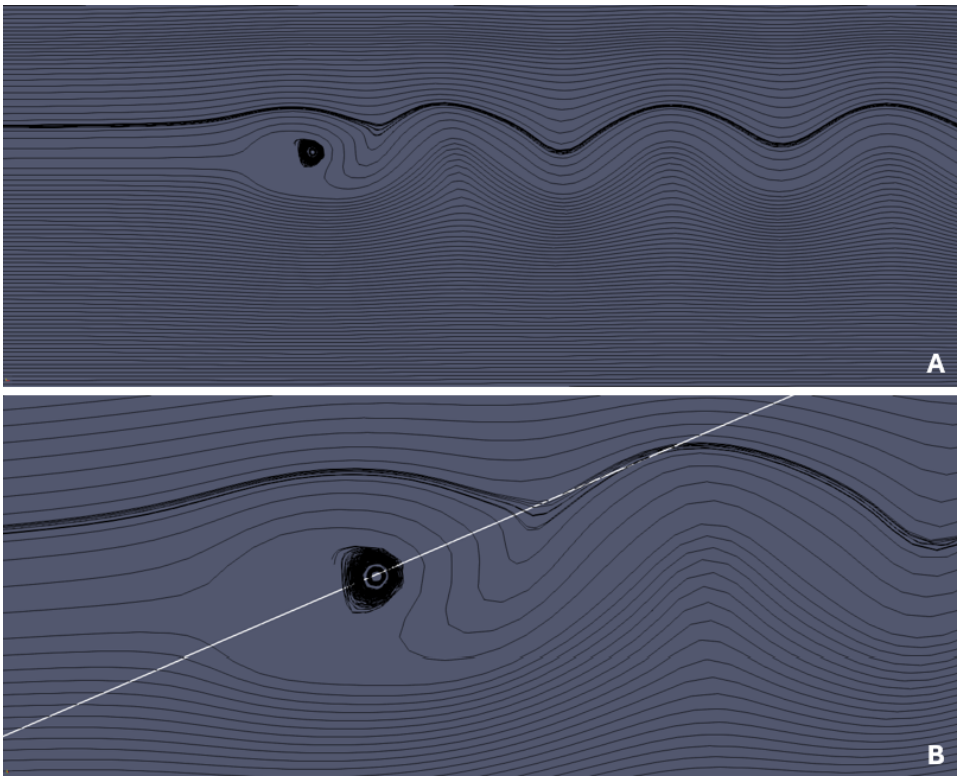

Figure 4.7.: Overview of the unsteady wake flow behind a circular cylinder (A) and corresponding detailed view of the near-wake region (B), Re=180.

This result can be placed in context with a study by Dennis and Chang [36]. They showed that the wake length L_w increases approximately linearly with Reynolds number once flow separation occurs (starting around $Re \approx 7$). Their data indicate a wake length of about:

- $L_w \approx 0.5D$ at $Re = 10$,
- $L_w \approx 2D$ at $Re = 40$,
- $L_w \approx 4D$ at $Re = 80$,
- $L_w \approx 5D$ at $Re = 100$. [36]

Compared to these values, the wake length observed in the current simulation suggests a Reynolds number in the range of 100-150. This is further supported by the presence of a periodic vortex street downstream of the recirculation zone, which typically emerges near $Re \approx 47$ and becomes fully developed and periodic at higher Reynolds numbers. Moreover,

the wake shape and flow reattachment behavior in the present results exhibit strong qualitative agreement with the flow fields presented in Dennis and Chang's figures, particularly their steady symmetric solutions up to $Re = 100$ (figures a to g) [36].
In unsteady flows like shown in 4.7, there's no single point where the wake "ends" like in steady flows. Instead, we measure the wake length based on how far we can observe vortices downstream of the cylinder. If defined that way, the total observable wake - including alternating vortices - extends about 6 to 7 diameters, again consistent with flows in the range $Re = 100$–200, as also discussed by Fornberg [40].

5. Conclusion and Outlook

In this project, steady and unsteady cylinder-flow simulations at low Reynolds numbers were carried out using the standard Spalart–Allmaras (SA) model and its modified negative-production variant (SAneg). At $Re = 20$, both models exhibited smooth convergence in the laminar regime, yielding drag coefficients $C_D \approx 1.50$ (SA) and 1.53 (SAneg) in close agreement with classical data (Schlichting 1.53). Lift coefficients remained near zero, reflecting the geometric symmetry.

Wall-shear-stress distributions confirmed that SAneg captured multiple separation and reattachment zones, in line with Dennis and Chang's observations of increasing wake complexity as Re rises. Wake-length measurements (6–7 D downstream, with a 2–2.5 D recirculation zone) further corroborate a transitional-flow behavior consistent with literature. Overall, the SAneg formulation demonstrates improved fidelity for both low-Re laminar benchmarks and transitional wake dynamics, while the standard SA model remains adequate for purely laminar cases.

Moving forward, our research will broaden to include a denser grid-convergence investigation by incorporating additional grid levels into the GCI analysis and systematically testing a wider span of Reynolds numbers to capture the full spectrum from creeping to transitional flows. We will refine our assessment of grid-independence by evaluating convergence indices across multiple mesh resolutions and conducting complementary y^+ studies to ensure near-wall accuracy. Concurrently, we plan to extend simulations beyond $Re = 20$ up to at least $Re = 500$, identifying key thresholds for unsteady behavior and validating turbulence-model performance across laminar, transitional, and early turbulent regimes. These enhancements will not only strengthen the quantitative robustness of our drag and lift predictions but also deepen insights into wake evolution and separation phenomena across different flow regimes.

Bibliography

[1] Launder B.E.: Review of *Turbulence Modeling for CFD* by D. C. Wilcox. *Journal of Fluid Mechanics* 289, 406–407, 1995.

[2] Cant S.: Book Review: *Turbulent Flows* by S. B. Pope. *Combustion and Flame* 125, 1361–1362, 2001. doi:10.1016/S0010-2180(01)00244-9.

[3] Spalart P.R. and Allmaras S.R.: A One-Equation Turbulence Model for Aerodynamic Flows. 30th Aerospace Sciences Meeting and Exhibit. AIAA, Reno, NV, 1992. doi: 10.2514/6.1992-439. AIAA Paper 1992-0439.

[4] Spalart P.R.: Strategies for turbulence modelling and simulations. *International Journal of Heat and Fluid Flow* 21(3), 252–263, 2000. doi:10.1016/S0142-727X(00)00007-2.

[5] S. R. Allmaras F. T. Johnson P.R.S.: Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model. *Seventh International Conference on Computational Fluid Dynamic* 2012.

[6] Hirt C.W. and Nichols B.D.: Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. *Journal of Computational Physics* 39(1), 201–225, 1981. doi: 10.1016/0021-9991(81)90145-5.

[7] Wang H. and Rose J.W.: Film Condensation in Horizontal Microchannels: Effect of Channel Shape. Proceedings of the 3rd International Conference on Microchannels and Minichannels (ICMM2005). ASME, Toronto, Ontario, Canada, 2005.

[8] Zhao T.S. and Liao Q.: Theoretical analysis of film condensation heat transfer inside vertical mini triangular channels. *International Journal of Heat and Mass Transfer* 45(13), 2829–2842, 2002. doi:10.1016/S0017-9310(01)00354-4.

[9] Cavallini A., Doretti L., Matkovic M. and Rossetto L.: Update on Condensation Heat Transfer and Pressure Drop inside Minichannels. *Heat Transfer Engineering* 27(4), 74–87, 2006. doi:10.1080/01457630500523907.

[10] Coleman J.W. and Garimella S.: Two-phase flow regimes in round, square and rectangular tubes during condensation of refrigerant R134a. *International Journal of Refrigeration* 26(1), 117–128, 2003. doi:10.1016/S0140-7007(02)00013-0.

[11] Garimella S., Killion J.D. and Coleman J.W.: An Experimentally Validated Model for Two-Phase Pressure Drop in the Intermittent Flow Regime for Circular Microchannels. *Journal of Fluids Engineering* 124(1), 205–215, 2002. doi:10.1115/1.1428327.

[12] Brackbill J.U., Kothe D.B. and Zemach C.: A Continuum Method for Modeling Surface Tension. *Journal of Computational Physics* 100, 335–354, 1992. doi:10.1016/0021-9991(92)90240-Y.

[13] H. K. Versteeg, W. Malalasekera: An Introduction to Computational Fluid Dynamics. Pearson Education Limited, 2007. ISBN 78-0-13-127498-3.

[14] Duquesne P., Maciel Y. and Deschênes C.: Investigation of Flow Separation in a Diffuser of a Bulb Turbine. *Journal of Fluids Engineering* 138(1), 011102–1–011102–9, 2016. doi:10.1115/1.4031254.
URL `https://doi.org/10.1115/1.4031254`

[15] Kubo N., Bhandari S., Tanaka M., Nonomura T. and Kawabata H.: Experimental Parametric Study on Flow Separation Control Mechanisms around NACA0015 Airfoil Using a Plasma Actuator with Burst Actuation over Reynolds Numbers of 10–10. *Applied Sciences* 14(11), 4652, 2024. doi:10.3390/app14114652.
URL `https://doi.org/10.3390/app14114652`

[16] Anderson W.K., Wood S.L. and Allmaras S.R.: An Initial Exploration of Improved Numerics within the Guidelines of the Negative Spalart-Allmaras Turbulence Model. Technical Memorandum NASA/TM–2019–220429, NASA Langley Research Center, Hampton, VA, USA, 2019.
URL `https://ntrs.nasa.gov/citations/20190033167`

[17] Aulakh D.J.S., Yang X. and Maulik R.: Robust Experimental Data Assimilation for the Spalart-Allmaras Turbulence Model. *Physical Review Fluids* 9(8), 084608, 2024. doi:10.1103/PhysRevFluids.9.084608.

[18] Diskin B., Liu Y. and Galbraith M.C.: High-Fidelity CFD Verification Workshop 2024: Spalart-Allmaras QCR2000-R Turbulence Model. AIAA SciTech Forum 2023 Special Session: High-Fidelity CFD Verification Preworkshop. American Institute of Aeronautics and Astronautics, National Harbor, MD, USA, 2024. Invited paper.
URL `https://github.com/HighFidelityCFDVerificationWorkshop`

[19] He X., Zhao F. and Vahdati M.: A Turbo-Oriented Data-Driven Modification to the Spalart–Allmaras Turbulence Model. *Journal of Turbomachinery* 144(12), 121007, 2022. doi:10.1115/1.4055333.

[20] V.D Narasimhamurthy: Unsteady-RANS Simulation of Turbulent Trailing-Edge Flow. Dissertation, Chalmers University of Technology, 2004.

[21] C. D. Munz, T. Westermann: Numerische Behandlung gewöhnlicher und partieller Differenzialgleichungen: Ein anwendungsorientiertes Lehrbuch für Ingenieure [German]. Springer, 2019. ISBN 978-3-662-55886-7.

[22] F. Moukalled, L. Mangani, M. Darwish: The Finite Volume Method in Computational Fluid Dynamics. Springer, 2016. ISBN 978-3-319-16873-9.

[23] Alfonsi G.: Reynolds-Averaged Navier–Stokes Equations for Turbulence Modeling. *CESIC–Supercomputing Center for Computational Engineering, Fluid Dynamics Division* 2009.

[24] OpenFOAM: API Guide v2112, 2025. Online.
URL `https://www.openfoam.com/documentation/guides/latest/api/classFoam_1_1RASModels_1_1SpalartAllmaras.html`

[25] OpenFOAM-pimpleFOAM-Rectangular˙Cylinder, 2021. Online.
URL `https://github.com/Interfluo/OpenFOAM-pimpleFOAM-Rectangular_Cylinder.git`

[26] Youtube-Tutorials, 2021. Online.
URL `https://github.com/AsmaaHADANE/Youtube-Tutorials.git`

[27] S OKAMOTO M.Y.: Flow past circular cylinder of finite length placed normal to ground plane in uniform shear flow. *Bulletin of JSME* 1984.

[28] Slater J.W.: Examining Spatial (Grid) Convergence, 2021.
URL `https://www.grc.nasa.gov/WWW/wind/valid/tutorial/spatconv.html`

[29] Roache P.J.: Perspective: A method for uniform reporting of grid refinement studies. *Journal of fluids engineering* 1994.

[30] Teschner T.R.: How to manage uncertainty in CFD: the grid convergence index, 2025.
URL `https://cfd.university/blog/how-to-manage-uncertainty-in-cfd-the-grid-convergence-#aioseo-tools-to-calculate-the-grid-convergence-index`

[31] Tritton D.J.: Experiments on the flow past a circular cylinder at low Reynolds numbers. *Journal of Fluid Mechanics* 6, 547–567, 1959.

[32] Abane S.M.: Calculation of Oseen flows past a circular cylinder at low Reynolds numbers. *Applied Scientific Research* 1978.

[33] W. S. Abbasi S.U.I.: Transition from steady to unsteady state flow around two inline cylinders under the effect of Reynolds numbers. *Journal of the Brazilian Society of Mechanical Sciences and Engineering* 40, 2018.

[34] W. K. Anderson S. L. Wood S.R.A.: An Initial Exploration of Improved Numerics within the Guidelines of the Negative Spalart-Allmaras Turbulence Model 2019.

[35] Residuals, 2025. Online.
URL `https://ansyshelp.ansys.com/public/account/secured?returnurl=/Views/Secured/corp/v242/en/cfx_mod/i1323887.html`

[36] S. C. R. Dennis G.C.: Numerical solutions for steady flow past a circular cylinder at Reynolds numbers up to 100. *Journal of Fluid Mechanics* 42, 471–489, 1970.

[37] M. Nishioka H.S.: Measurements of velocity distributions in the wake of a circular cylinder at low Reynolds numbers. *Journal of Fluid Mechanics* 65, 97–112, 1974.

[38] J. Park K. Kwon H.C.: Numerical Solutions of Flow Past a Circular Cylinder at Reynolds Numbers up to 160. *KSME International Journal* 12, 1200–1205, 1998.

[39] Schlichting H. and Gersten K.: Boundary-Layer Theory. Springer, Berlin, Heidelberg, 9th edition, 2017. ISBN 978-3-662-52917-1. doi:10.1007/978-3-662-52919-5.

[40] Fornberg B.: A numerical study of steady viscous flow past a circular cylinder. *Journal of Fluid Mechanics* 98(4), 819–855, 1980. doi:10.1017/S0022112080000302.

[41] Griffin E.: The unsteady wake of an oscillating cylinder at low Reynolds number. *Journal of Applied Mechanics* 38(4), 729–738, 1971. doi:10.1115/1.3408948.

A. Folder structure SA negative turbulence model

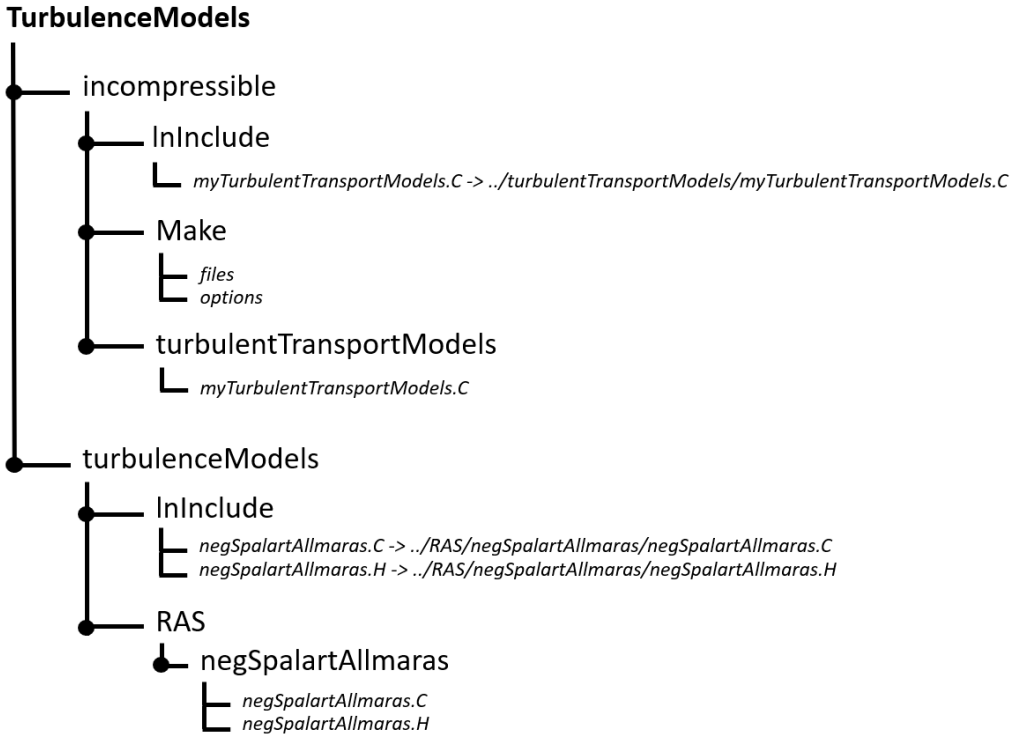

Figure A.1.: Folder structure of the SA negative turbulence model.

B. negSpalartAllmaras.H

```
#ifndef negSpalartAllmaras_H
#define negSpalartAllmaras_H

#include "RASModel.H"
#include "eddyViscosity.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace RASModels
{

/*---------------------------------------------------------------------------*\
                    Class negSpalartAllmaras Declaration
\*---------------------------------------------------------------------------*/

template<class BasicTurbulenceModel>
class negSpalartAllmaras
:
    public eddyViscosity<RASModel<BasicTurbulenceModel>>
{
    // Private Member Functions

        //- No copy construct
        negSpalartAllmaras(const negSpalartAllmaras&) = delete;

        //- No copy assignment
        void operator=(const negSpalartAllmaras&) = delete;
```

Figure B.1.: Definition of the class *negSpalartAllmaras*.

```
protected:

    // Protected Data

        // Model coefficients

            dimensionedScalar sigmaNut_;
            dimensionedScalar kappa_;
            dimensionedScalar Cb1_;
            dimensionedScalar Cb2_;
            dimensionedScalar Cw1_;
            dimensionedScalar Cw2_;
            dimensionedScalar Cw3_;
            dimensionedScalar Cv1_;
            dimensionedScalar Cs_;

    // Additional model coefficients

        dimensionedScalar Cn1_;
        dimensionedScalar Cn2_;
        dimensionedScalar Cn3_;
        dimensionedScalar Ct3_;
        dimensionedScalar Ct4_;

        // Fields

            //- Modified kinematic viscosity [m2/s]
            volScalarField nuTilda_;

            //- Wall distance
            //  Note: different to wall distance in parent RASModel
            //  which is for near-wall cells only
            const volScalarField::Internal& y_;


    // Protected Member Functions

        tmp<volScalarField> chi() const;

        tmp<volScalarField> fv1(const volScalarField& chi) const;

        tmp<volScalarField::Internal> fv2
        (
            const volScalarField::Internal& chi,
            const volScalarField::Internal& fv1
        ) const;

        tmp<volScalarField::Internal> Stilda() const;

        tmp<volScalarField::Internal> fw
        (
            const volScalarField::Internal& Stilda
        ) const;


        //- Update nut with the latest available nuTilda
        virtual void correctNut();
```

Figure B.2.: Definition of the model and additional model coefficients and definition of the fields and of the protected member functions.

```
public:

    typedef typename BasicTurbulenceModel::alphaField alphaField;
    typedef typename BasicTurbulenceModel::rhoField rhoField;
    typedef typename BasicTurbulenceModel::transportModel transportModel;


    //- Runtime type information
    TypeName("negSpalartAllmaras");


    // Constructors

        //- Construct from components
        negSpalartAllmaras
        (
            const alphaField& alpha,
            const rhoField& rho,
            const volVectorField& U,
            const surfaceScalarField& alphaRhoPhi,
            const surfaceScalarField& phi,
            const transportModel& transport,
            const word& propertiesName = turbulenceModel::propertiesName,
            const word& type = typeName
        );


    //- Destructor
    virtual ~negSpalartAllmaras() = default;


    // Member Functions

        //- Re-read model coefficients if they have changed
        virtual bool read();

        //- Return the effective diffusivity for nuTilda
        tmp<volScalarField> DnuTildaEff() const;

        //- Return the (estimated) turbulent kinetic energy
        virtual tmp<volScalarField> k() const;

        //- Return the (estimated) turbulent kinetic energy dissipation rate
        virtual tmp<volScalarField> epsilon() const;

        //- Return the (estimated) specific dissipation rate
        virtual tmp<volScalarField> omega() const;

        //- Solve the turbulence equations and correct the turbulent viscosity
        virtual void correct();
};
```

Figure B.3.: Definition of the Constructors, Destructor and member functions.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace RASModels
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#ifdef NoRepository
    #include "negSpalartAllmaras.C"
#endif

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

// ************************************************************************* //
```

Figure B.4.: End of the *negSpalartAllmaras.H* file.

C. negSpalartAllmaras.C

```
#include "negSpalartAllmaras.H"
#include "fvOptions.H"
#include "bound.H"
#include "wallDist.H"
#include <math.h>

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace RASModels
{

// * * * * * * * * * * * * * Protected Member Functions  * * * * * * * * * * * * * //

template<class BasicTurbulenceModel>
tmp<volScalarField> negSpalartAllmaras<BasicTurbulenceModel>::chi() const
{
    return nuTilda_/this->nu();
}


template<class BasicTurbulenceModel>
tmp<volScalarField> negSpalartAllmaras<BasicTurbulenceModel>::fv1
(
    const volScalarField& chi
) const
{
    const volScalarField chi3(pow3(chi));

    return chi3/(chi3 + pow3(Cv1_));
}


template<class BasicTurbulenceModel>
tmp<volScalarField::Internal> negSpalartAllmaras<BasicTurbulenceModel>::fv2
(
    const volScalarField::Internal& chi,
    const volScalarField::Internal& fv1
) const
{
    return scalar(1) - chi/(scalar(1) + chi*fv1);
}
```

Figure C.1.: Definition of χ, f_{v1} and f_{v2}.

```
// Modified Stilda
template<class BasicTurbulenceModel>
tmp<volScalarField::Internal> negSpalartAllmaras<BasicTurbulenceModel>::Stilda()
const
{
    const volScalarField chi(this->chi());

    const volScalarField fv1(this->fv1(chi));

    const volScalarField::Internal Omega
    (
        ::sqrt(scalar(2))*mag(skew(fvc::grad(this->U_)().v()))
    );

    return
    (
        Omega
        + pos(Cn2_*Omega + (fv2(chi(), fv1())*nuTilda_()/sqr(kappa_*y_)))*  (fv2(chi(), fv1())*nuTilda_()/sqr(kappa_*y_))
        +neg(Cn2_*Omega + (fv2(chi(), fv1())*nuTilda_()/sqr(kappa_*y_)))
        *(Omega*(sqr(Cn2_)*Omega + Cn3_*(fv2(chi(), fv1())*nuTilda_()/
        sqr(kappa_*y_))))/((Cn3_ - scalar(2)*Cn2_)*Omega - (fv2(chi(), fv1())*  nuTilda_()/sqr(kappa_*y_)))
      );
}


template<class BasicTurbulenceModel>
tmp<volScalarField::Internal> negSpalartAllmaras<BasicTurbulenceModel>::fw
(
    const volScalarField::Internal& Stilda
) const
{
    const volScalarField::Internal r
    (
        min
        (
            nuTilda_
           /(
                max
                (
                    Stilda,
                    dimensionedScalar(Stilda.dimensions(), SMALL)
                )
              *sqr(kappa_*y_)
            ),
            scalar(10)
        )
    );

    const volScalarField::Internal g(r + Cw2_*(pow6(r) - r));

    return
        g*pow
        (
            (scalar(1) + pow6(Cw3_))/(pow6(g) + pow6(Cw3_)),
            scalar(1)/scalar(6)
        );
}
```

Figure C.2.: Definition of the modified vorticity $\tilde{S}$ and f_w.

```
template<class BasicTurbulenceModel>
void negSpalartAllmaras<BasicTurbulenceModel>::correctNut()
{
    this->nut_ = nuTilda_*this->fv1(this->chi());
    this->nut_.correctBoundaryConditions();
    fv::options::New(this->mesh_).correct(this->nut_);

    BasicTurbulenceModel::correctNut();
}


// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //

template<class BasicTurbulenceModel>
negSpalartAllmaras<BasicTurbulenceModel>::negSpalartAllmaras
(
    const alphaField& alpha,
    const rhoField& rho,
    const volVectorField& U,
    const surfaceScalarField& alphaRhoPhi,
    const surfaceScalarField& phi,
    const transportModel& transport,
    const word& propertiesName,
    const word& type
)
:
    eddyViscosity<RASModel<BasicTurbulenceModel>>
    (
        type,
        alpha,
        rho,
        U,
        alphaRhoPhi,
        phi,
        transport,
        propertiesName
    ),

    sigmaNut_
    (
        dimensioned<scalar>::getOrAddToDict
        (
            "sigmaNut",
            this->coeffDict_,
            scalar(2)/scalar(3)
        )
    ),
    kappa_
    (
        dimensioned<scalar>::getOrAddToDict
        (
            "kappa",
            this->coeffDict_,
            0.41
        )
    ),
```

Figure C.3.: Definition of the function *correctNut()* and start of the definition of the constants.

```
Cb1_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cb1",
        this->coeffDict_,
        0.1355
    )
),
Cb2_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cb2",
        this->coeffDict_,
        0.622
    )
),
Cw1_(Cb1_/sqr(kappa_) + (scalar(1) + Cb2_)/sigmaNut_),
Cw2_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cw2",
        this->coeffDict_,
        0.3
    )
),
Cw3_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cw3",
        this->coeffDict_,
        2.0
    )
),
Cv1_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cv1",
        this->coeffDict_,
        7.1
    )
),

Cs_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cs",
        this->coeffDict_,
        0.3
    )
),
```

Figure C.4.: Definition of constants.

```
// Additional model coefficients
Cn1_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cn1",
        this->coeffDict_,
        16.0
    )
),

Cn2_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cn2",
        this->coeffDict_,
        0.7
    )
),
Cn3_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Cn3",
        this->coeffDict_,
        0.9
    )
),
Ct3_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Ct3",
        this->coeffDict_,
        1.2
    )
),
Ct4_
(
    dimensioned<scalar>::getOrAddToDict
    (
        "Ct4",
        this->coeffDict_,
        0.5
    )
),
```

Figure C.5.: Definition of the additional model coefficients.

```
    nuTilda_
    (
        IOobject
        (
            "nuTilda",
            this->runTime_.timeName(),
        this->mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        this->mesh_
    ),

    y_(wallDist::New(this->mesh_).y())
{
    if (type == typeName)
    {
        this->printCoeffs(type);
    }
}

// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * //

template<class BasicTurbulenceModel>
bool negSpalartAllmaras<BasicTurbulenceModel>::read()
{
    if (eddyViscosity<RASModel<BasicTurbulenceModel>>::read())
    {
        sigmaNut_.readIfPresent(this->coeffDict());
        kappa_.readIfPresent(this->coeffDict());

        Cb1_.readIfPresent(this->coeffDict());
        Cb2_.readIfPresent(this->coeffDict());
        Cw1_ = Cb1_/sqr(kappa_) + (scalar(1) + Cb2_)/sigmaNut_;
        Cw2_.readIfPresent(this->coeffDict());
        Cw3_.readIfPresent(this->coeffDict());
        Cv1_.readIfPresent(this->coeffDict());
        Cs_.readIfPresent(this->coeffDict());
    Cn1_.readIfPresent(this->coeffDict());
        Cn2_.readIfPresent(this->coeffDict());
    Cn3_.readIfPresent(this->coeffDict());
    Ct3_.readIfPresent(this->coeffDict());
    Ct4_.readIfPresent(this->coeffDict());

        return true;
    }

    return false;
}
```

Figure C.6.: Definition of *nuTilda˙*, *y˙* and of the member Functions.

```
// Modified DnuTildaEff
template<class BasicTurbulenceModel>
tmp<volScalarField> negSpalartAllmaras<BasicTurbulenceModel>::DnuTildaEff() const
{
    volScalarField pow3chi = pow(this->chi(), 3);
    volScalarField fn = pos(this->chi()) + neg(this->chi())*(Cn1_ + pow3chi)/(Cn1_ - pow3chi);
    return tmp<volScalarField>::New
    (
        "DnuTildaEff",
        (fn*nuTilda_ + this->nu())/sigmaNut_
    );
}


template<class BasicTurbulenceModel>
tmp<volScalarField> negSpalartAllmaras<BasicTurbulenceModel>::k() const
{
    // (B:Eq. 4.50)
    const scalar Cmu = 0.09;

    return tmp<volScalarField>::New
    (
        IOobject
        (
            IOobject::groupName("k", this->alphaRhoPhi_.group()),
            this->runTime_.timeName(),
            this->mesh_
        ),
        cbrt(this->fv1(this->chi()))
        *nuTilda_
        *::sqrt(scalar(2)/Cmu)
        *mag(symm(fvc::grad(this->U_))),
        this->nut_.boundaryField().types()
    );
}
```

Figure C.7.: Definition of the modified diffusion coefficient *DnuTildaEff* and of the member function *k()*.

```
template<class BasicTurbulenceModel>
tmp<volScalarField> negSpalartAllmaras<BasicTurbulenceModel>::epsilon() const
{
    // (B:Eq. 4.50)
    const scalar Cmu = 0.09;
    const dimensionedScalar nutSMALL(sqr(dimLength)/dimTime, SMALL);

    return tmp<volScalarField>::New
    (
        IOobject
        (
            IOobject::groupName("epsilon", this->alphaRhoPhi_.group()),
            this->runTime_.timeName(),
            this->mesh_
        ),
        pow(this->fv1(this->chi()), 0.5)
        *pow(::sqrt(Cmu)*this->k(), 2)
        /(nuTilda_ + this->nut_ + nutSMALL),
        this->nut_.boundaryField().types()
    );
}


template<class BasicTurbulenceModel>
tmp<volScalarField> negSpalartAllmaras<BasicTurbulenceModel>::omega() const
{
    // (P:p. 384)
    const scalar betaStar = 0.09;
   const dimensionedScalar k0(sqr(dimLength/dimTime), SMALL);

    return tmp<volScalarField>::New
    (
        IOobject
        (
            IOobject::groupName("omega", this->alphaRhoPhi_.group()),
            this->runTime_.timeName(),
            this->mesh_
        ),
        this->epsilon()/(betaStar*(this->k() + k0)),
        this->nut_.boundaryField().types()
    );
}
```

Figure C.8.: Definition of the member functions *epsilon()* and *omega()*.

```
template<class BasicTurbulenceModel>
void negSpalartAllmaras<BasicTurbulenceModel>::correct()
{
    if (!this->turbulence_)
    {
        return;
    }

    {
        // Construct local convenience references
        const alphaField& alpha = this->alpha_;
        const rhoField& rho = this->rho_;
        const surfaceScalarField& alphaRhoPhi = this->alphaRhoPhi_;
        fv::options& fvOptions(fv::options::New(this->mesh_));

        eddyViscosity<RASModel<BasicTurbulenceModel>>::correct();

        const volScalarField::Internal Stilda(this->Stilda());
    const volScalarField::Internal chi(this->chi());

    // Definiton of the vorticity
        const volScalarField::Internal Omega
        (
            ::sqrt(scalar(2))*mag(skew(fvc::grad(this->U_)().v()))
        );


        tmp<fvScalarMatrix> nuTildaEqn
        (
            fvm::ddt(alpha, rho, nuTilda_)
          + fvm::div(alphaRhoPhi, nuTilda_)
          - fvm::laplacian(alpha*rho*DnuTildaEff(), nuTilda_)
          - Cb2_/sigmaNut_*alpha*rho*magSqr(fvc::grad(nuTilda_))
         ==
     pos(nuTilda_)*(Cb1_*(scalar(1)-Ct3_*exp(-Ct4_*pow(chi,2)))*alpha()*rho()*Stilda*nuTilda_()
         - fvm::Sp(Cw1_*alpha()*rho()*fw(Stilda)*nuTilda_()/sqr(y_), nuTilda_)
     + fvm::Sp(Cb1_/sqr(kappa_)*alpha()*rho()*Ct3_*exp(-Ct4_*pow(chi,2))*nuTilda_()/sqr(y_), nuTilda_))
     + neg(nuTilda_)*(Cb1_*(scalar(1)-Ct3_)*alpha()*rho()*Omega*nuTilda_()
         + fvm::Sp(Cw1_*alpha()*rho()*nuTilda_()/sqr(y_),        nuTilda_))


        );


        nuTildaEqn.ref().relax();
        fvOptions.constrain(nuTildaEqn.ref());
        solve(nuTildaEqn);
        fvOptions.correct(nuTilda_);
    nuTilda_.correctBoundaryConditions();
    }

    // Update nut with latest available nuTilda
    correctNut();
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace RASModels
} // End namespace Foam
```

Figure C.9.: Definition of the transport equation.

D. files and options file

```
turbulentTransportModels/myTurbulentTransportModels.C
LIB = $(FOAM_USER_LIBBIN)/libmyIncompressibleTurbulenceModels
```

Figure D.1.: File *files*.

```
EXE_INC = \
    -I../turbulenceModels/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude
LIB_LIBS = \
    -lturbulenceModels \
    -lfiniteVolume \
    -lmeshTools \
    -lincompressibleTransportModels \
    -lincompressibleTurbulenceModels
```

Figure D.2.: File *options*.

E. myTurbulentTransportModels.C file

```
#include "turbulentTransportModels.H"

// ------------------------------------------------------------------------- //
// RAS models
// ------------------------------------------------------------------------- //

#include "negSpalartAllmaras.H"
makeRASModel(negSpalartAllmaras);

// ************************************************************************* //
```

Figure E.1.: File *options.*