Winter 12-15-2016

Development of a One-Equation Eddy Viscosity Turbulence Model for Application to Complex Turbulent Flows

Timothy Wray
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering & Applied Science
Department of Mechanical Engineering & Material Science

Dissertation Examination Committee:
Dr. Ramesh Agarwal, Chair
Dr. Mark Jakiela
Dr. Kenneth Jerina
Dr. Mori Mani
Dr. David Peters
Dr. Palghat Ramachandran

Development of a One-Equation Eddy Viscosity Turbulence Model for Application to Complex Turbulent Flows

by
Timothy J. Wray

A dissertation presented to
The Graduate School
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

December 2016
St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Dr. Ramesh Agarwal for his guidance and advice throughout this research. Your support of my ideas and confidence in my abilities is what made this work possible. You have set an example of excellence as a researcher, mentor, and role model.

I would like to extend my gratefulness to my committee members and the staff of the MEMS department for their time and contributions to my development while at Washington University.

I would like to thank my family for their love support, and constant encouragement. A special thank you to all my colleagues in the CFD lab over the years for their camaraderie and collaboration.

The financial support for this work was provided by a NASA Space Grant and a NASA EPSCoR Grant. It is gratefully acknowledged.

Tim Wray

Washington University in St. Louis

December 2016

ABSTRACT OF THE DISSERTATION

Development of a One-Equation Eddy Viscosity Turbulence Model for Application to Complex Turbulent Flows

by

Timothy J. Wray

Doctor of Philosophy in Mechanical Engineering

Washington University in St. Louis, 2017

Research Advisor: Ramesh Agarwal

Computational fluid dynamics (CFD) is routinely used in performance prediction and design of aircraft, turbomachinery, automobiles, and in many other industrial applications. Despite its wide range of use, deficiencies in its prediction accuracy still exist. One critical weakness is the accurate simulation of complex turbulent flows using the Reynolds-Averaged Navier-Stokes equations in conjunction with a turbulence model. The goal of this research has been to develop an eddy viscosity type turbulence model to increase the accuracy of flow simulations for mildly separated flows, flows with rotation and curvature effects, and flows with surface roughness. It is accomplished by developing a new zonal one-equation turbulence model which relies heavily on the flow physics; it is now known in the literature as the Wray-Agarwal one-equation turbulence model. The effectiveness of the new model is demonstrated by comparing its results with those obtained by the industry standard one-equation Spalart-Allmaras model and two-equation Shear-Stress-Transport $k - \omega$ model and experimental data. Results for subsonic, transonic, and supersonic flows in and about complex geometries are presented. It is demonstrated that the Wray-Agarwal model can provide the industry and CFD researchers an accurate, efficient, and reliable turbulence model for the computation of a large class of complex turbulent flows.

Chapter 1: Introduction

1.1 Background and Motivation

Computational fluid dynamics (CFD) is routinely used in the design and performance prediction of aircraft, turbomachinery, automobiles, and may other industrial applications. Over the last four decades, a great deal of progress has been made in the accurate CFD prediction of a wide variety of turbulent flows. Turbulence modeling, however, remains a critical item in the accurate prediction of complex turbulent flows. Modeling approaches can be broadly labeled as full modeling using the Reynolds-Averaged Navier-Stokes (RANS) equations, partial modeling using Large Eddy Simulation (LES), and zero modeling using Direct Numerical Simulation (DNS). DNS resolves the complete turbulence structure, producing accurate simulations. However, even optimistic predictions believe DNS will be unavailable for common engineering problems until 2080 due to its large computational requirements [1]. LES reduces the computational cost of DNS by only resolving large scale turbulence and modeling sub-grid turbulence. For 3D high Reynolds number flows, LES is still cost prohibitive and expected to remain unavailable for routine use until 2045 [1]. Hybrid RANS/LES approaches further reduce the computational cost by using LES away from solid boundaries and reserving the RANS equations for modeling the near-wall turbulence. Hybrid methods introduce the additional complication of blending the RANS/LES frameworks and the computational cost remains large for massively separated 3D flows. Also the near wall accuracy of the simulation is still dependent on the underlying RANS modeling. Complete modeling of the turbulent stresses in the RANS equations remains by far the most widely used approach for the prediction of turbulent flows due to its ease of use and low computational

cost. For these reasons accurate RANS modeling of the turbulent stresses is essential to current and next generation CFD.

Many turbulence models have been developed in the RANS framework. Despite their wide range of use, deficiencies in the prediction accuracy of the models still exist. The purpose of this work is to develop an eddy viscosity turbulence model to increase the accuracy of flow simulations for mildly separated flows, flows with rotation and curvature effects, and flows with surface roughness. This is accomplished by building a zonal one-equation turbulence model reliant on flow physics, termed the Wray-Agarwal (WA) model. The effectiveness of the new model is demonstrated by comparing it against the industry standard Spalart-Allmaras (SA) and Shear-Stress-Transport (SST) k-ω models and experimental data. Results for subsonic, transonic, and supersonic flows of varying geometrical difficulty are presented. It is anticipated that the Wray-Agarwal model will provide industry and researchers the means for accurate and reliable computation of a large class of complex turbulent flows.

1.3 Objectives

The overall objective of this work is to develop a new one-equation eddy viscosity turbulence model and to evaluate its performance for a broad range of turbulent flows. The principal tasks to be accomplished are:

1) Correct the implementation of the SA and SST *k-ω* turbulence models in the flow solver OpenFOAM so that they are in agreement with previous published results.
2) Derive a new one-equation turbulence model based on *k-ω* closure.

3) Define a blending function to control the near-wall and away from the wall behavior of the WA model.
4) Validate the WA, SA, and SST *k-ω* models for a wide range of subsonic, transonic, and supersonic flows.
5) Implement rotation/curvature corrections and surface roughness corrections in the WA, SA, and SST k-ω models.

1.4 Outline

The overall goal of this dissertation is introduce a new one-equation eddy viscosity turbulence model and establish its accuracy for a broad range of fluid flows. Each Chapter will explain in sufficient detail the following topics:

Chapter 2: Introduction to Turbulence Modeling: This chapter provides an introduction to computational fluid dynamics (CFD) and turbulence modeling. First the governing equations of fluid dynamics are introduced, along with their transformation into the Reynolds-Averaged Navier-Stokes (RANS) equations. Eddy viscosity turbulence models are explained and the equations of the SA and SST *k-ω* turbulence models are provided.

Chapter 3: OpenFOAM Verification: In order to build confidence in the capabilities of the OpenFOAM solver, several basic flows are investigated. Implementation of the SA model and SST *k-ω* model will be investigated and the models are modified to become consistent with the NASA CFL3D and FUN3D solvers.

Chapter 4: The Wray-Agarwal (WA) Turbulence Model: This chapter introduces the WA turbulence model. A historical perspective of one-equation eddy viscosity turbulence models is

provided. The derivation of the WA turbulence model and its relation to other models is given. Results of the WA model for several self-similar free shear flows are presented.

Chapter 5: Validation Cases: In this chapter a direct comparison among the WA, SA, SST k-ω models, and experimental data for a wide range of flows are made. The goal of this chapter is to establish the ability of the WA model to predict a broad range of flow physics. The flow conditions, geometry, and grid for each case are given.

Chapter 6: Rotation/Curvature and Surface Roughness Corrections: Two special flow regimes for which regular eddy viscosity turbulence models cannot capture the flow physics are flows with system rotation / geometry curvature and flows with surface roughness. An introduction to these flow regimes is provided. Corrections for the WA model are developed. Corrections for the SA and SST k-ω models are taken from the literature. Results for the baseline and corrected models for several simple cases are presented.

Chapter 7: Summary and Future Work: This chapter summarizes the results obtained with the WA model. Contributions made in the dissertation and possible future work are discussed.

Chapter 2: Introduction to Turbulence Modeling

2.1 Introduction

The Navier-Stokes (NS) equations completely describe the motion of continuum fluid flow. Analytical solutions of these equations exist only for laminar flows for very simple geometries and flow conditions. For large Reynolds numbers, the flow becomes turbulent and the analytical solutions to the NS equations cannot be obtained. Thus, at high Reynolds numbers a numerical approach becomes necessary for solution of turbulent flow fields. The methodology of numerically solving the governing equations of fluid dynamics is known as computational fluid dynamics (CFD).

The computational cost of exactly solving the NS equations at high Reynolds numbers is extremely high. Turbulence is inherently unsteady with a wide range of time and length scales. Capturing all the features of turbulence for an industrial flow is not feasible in terms of computation time and storage. Estimates put the first direct numerical simulation (DNS) of a complete aircraft to be possible around 2080 [1]. Therefore for CFD to be useful today, a computationally simpler set of equations for computation of turbulent flows is required. The most popular approach for reducing the NS equations is based on statistical modeling of the turbulent stresses, leading to the Reynolds-Averaged Navier-Stokes (RANS) equations. In the RANS equations turbulent stresses are unknown which results in the "closure problem." The closure problem is addressed by modeling the turbulent stresses using the empirically known behavior of turbulence quantities such as turbulent kinetic energy, turbulent dissipation, etc. The modeling of turbulent stresses in the RANS

equation is known as "turbulence modeling." The most accurate possible representation of the turbulent stresses is the goal of turbulence modeling.

In the following sections the RANS equations are introduced. Eddy viscosity turbulence models are discussed with special emphasis on the industry standard Spalart-Allmaras and Shear-Stress-Transport k-ω turbulence models. The numerical considerations for solving the RANS equations are described. Finally the numerical solver OpenFOAM and a general CFD procedure are discussed.

2.2 Reynolds-Averaged Navier-Stokes Equations.

The Navier-Stokes equations completely describe the motion of continuum fluid. The equations are derived from conservation of mass, momentum, and energy, Stoke's hypothesis, and an equation of state. For simplicity the incompressible NS equations are considered. The governing equations for an incompressible fluid are given in Eq (1).

$$\frac{\partial u_i}{\partial x_i} = 0$$
$$\rho \frac{\partial u_i}{\partial t} + \rho u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left(2\mu s_{ij}\right) \tag{1}$$

where u_i and x_i are velocity and position vectors, t is time, p is pressure, ρ is density, μ is dynamic viscosity, and s_{ij} is the strain rate tensor. All these quantities are instantaneous values. In Reynolds decomposition an instantaneous quantity is decomposed into the sum of its mean and fluctuating component. For example, instantaneous velocity, u, can be written as the sum of the mean velocity U and velocity fluctuation u'. The mean velocity is the time-averaged instantaneous velocity. By applying this operation to the incompressible NS equations the Reynolds-Averaged Navier-Stokes equation are obtained as:

$$\frac{\partial U_i}{\partial x_i} = 0$$

$$\rho \frac{\partial U_i}{\partial t} + \rho U_j \frac{\partial U_i}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}\left(2\mu S_{ij} - \rho\overline{u_i' u_j'}\right) \tag{2}$$

where U_i, P, and S_{ij} are the mean velocity, pressure, and strain rate tensor, respectively. Comparing Eqs. 1 and 2, it can be seen that the time averaged continuity equation is identical to the instantaneous continuity equation with mean velocity replacing the instantaneous velocity. Similarly time-averaged quantities replace the instantaneous quantities in the Navier-Stokes momentum equation. The only difference between Eqs. (1) and (2), aside from these replacements, is the additional term $\rho\overline{u_i' u_j'}$ found in the time averaged NS equation. This term represents the rate of momentum transfer due to turbulence and is known as the Reynolds-stress tensor. The Reynolds stress tensor must be defined in order to close the system of equations and compute the mean flow quantities.

2.3 Eddy Viscosity Turbulence Models

In an analogy to the molecular momentum transport process, the Boussineq approximation given in Eq. 3 can be used to model the Reynolds-stress tensor by introduction a scalar, the eddy viscosity. The modeling problem then becomes that of determining the eddy viscosity. The class of turbulence models that define ν_t, are known as eddy viscosity turbulence models. Eddy viscosity models are the most common and are the easiest to implement turbulence models.

$$\overline{u_i' u_j'} = -\nu_t S_{ij} + \frac{2}{3} k \delta_{ij} \tag{3}$$

To model the eddy viscosity one can either use some combination of turbulent length scale, time scale, and kinetic energy or introduce directly an equation for eddy viscosity. The former method has the advantage of using well defined quantities, such as the turbulent kinetic energy, while the

latter is generally simpler. Details of two of the most widely used eddy viscosity models are given in the next sections.

2.3.1 Spalart-Allmaras Turbulence Model

The Spalart-Allmaras (SA) turbulence model is a single equation eddy-viscosity model developed for the prediction of aerodynamic flows [2]. The SA model's transport quantity, $\tilde{\nu}$, defines the eddy-viscosity everywhere except very close to the wall. A clever formulation allows $\tilde{\nu}$ to vary linearly as $\tilde{\nu}=uky$ not only through the log-layer but also down to the wall. This linearity makes the model numerically attractive because second order discretization methods can capture linear functions very accurately. The model was derived using empiricism and arguments of dimensional analysis (having no link to the *k-ε* turbulence model). It is a low-Reynolds number model, allowing for accuracy all the way to the wall assuming that the mesh is properly refined. The transport equation for the standard SA model and its definition of eddy viscosity are given below:

$$\frac{D\tilde{\nu}}{Dt} = c_{b1}[1-f_{t2}]\,\tilde{S}\,\tilde{\nu} + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left((\nu+\tilde{\nu})\,\frac{\partial\tilde{\nu}}{\partial x_j}\right) + c_{b2}\left(\frac{\partial\tilde{\nu}}{\partial x_j}\right)^2\right] - \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left[\frac{\tilde{\nu}}{d}\right]^2 \tag{4}$$

The turbulent eddy-viscosity is given by the equation:

$$\nu_t = \tilde{\nu}\, f_{v1}. \tag{5}$$

Near wall blocking is accounted for by the damping function f_{v1}.

$$f_{v1} = \frac{\chi^3}{\chi^3 + c^3{}_{v1}}, \qquad \chi = \frac{\tilde{\nu}}{\nu}. \tag{6}$$

The remaining function definitions are given by the following equations:

$$\tilde{S} = \Omega + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \qquad f_{v2} = 1 - \frac{\chi}{1 - \chi f_{v1}} \tag{7}$$

$$f_w = g\left[\frac{1 + c^6{}_{w3}}{g^6 + c^6{}_{w3}}\right]^{1/6}, \tag{8}$$

$$g = r + c_{w2}(r^6 - r), \tag{9}$$

$$r = \frac{\tilde{\nu}}{\tilde{S}\,\kappa^2 d^2}, \tag{10}$$

$$f_{t2} = C_{t3} exp(-C_{t4}\chi^2) \tag{11}$$

The closure coefficients for the SA model are given in Table 2.1.

Table 2.1: Spalart-Allmaras closure coefficients.

Closure Coefficient	Value
c_{b1}	1.355
c_{b2}	0.622
c_{v1}	7.1
c_{t3}	1.2
c_{t4}	0.5
c_{w2}	0.3
c_{w3}	2.0
σ	2/3
κ	0.41

2.3.2 Shear-Stress-Transport *k-ω* Turbulence Model

The Shear-Stress-Transport (SST) *k-ω* [3] model combines the desirable characteristics of the *k-ε* and *k-ω* models. This is accomplished by blending the *k-ω* and *k-ε* formulations. Wilcox's *k-ω* [4] model remains active only near the solid boundaries and the standard *k-ε* model is used at the boundary layer edge and other shear regions. By confining the *k-ω* model to the inner part of the boundary layer, its sensitivity to the free-stream conditions is avoided. Additionally, *k-ε* models have historically been more accurate in predicting free-shear flows than *k-ω* models [6]. The near wall behavior of *k-ε* models is poor and requires additional damping functions to resolve the viscous sublayer, while *k-ω* models are integrable to the wall without corrective functions. The

shear-stress-transport aspect of the model limits the eddy viscosity as a function of the turbulent kinetic energy. This modification improves the prediction of flows with strong pressure gradients and separation. The equations of the SST *k*-ω model are given below. The transport equations for *k* and ω are given by:

$$\frac{D\rho k}{Dt} = \tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta^*\rho\omega\kappa + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_k\mu_t)\frac{\partial k}{\partial x_j}\right] \tag{12}$$

$$\frac{D\rho\omega}{Dt} = \frac{\gamma}{\nu_t}\tau_{ij}\frac{\partial u_i}{\partial x_j} - \beta^*\rho\omega^2 + \frac{\partial}{\partial x_j}\left[(\mu + \sigma_\omega\mu_t)\frac{\partial \omega}{\partial x_j}\right] + 2(1-F_1)\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x} \tag{13}$$

The turbulent eddy-viscosity is computed from:

$$\nu_t = \frac{a_1 k}{\max(a_1\omega; \Omega F_2)}\ ,\ \ \Omega = \sqrt{2W_{ij}W_{ij}}\ ,\ \ W_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \tag{14}$$

Each model constant is blended between an inner and outer constant by:

$$\varphi = F_1\varphi_1 + (1-F_1)\varphi_2 \tag{15}$$

The remaining function definitions are given by the following equations:

$$F_1 = tanh(arg_1^4) \tag{16}$$

$$arg_1 = \min\left[\max\left(\frac{\sqrt{k}}{\beta^*\omega d}, \frac{500\nu}{d^2\omega}\right), \frac{4\rho\sigma_{\omega 2}k}{CD_{k\omega}d^2}\right] \tag{17}$$

$$CD_k = \max\left(2\rho\sigma_{\omega 2}\frac{1}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial \omega}{\partial x_j}, 10^{-20}\right) \tag{18}$$

$$F_2 = tanh(arg_2^2) \tag{19}$$

$$arg_2 = \max\left(2\frac{\sqrt{k}}{\beta^*\omega d}, \frac{500\nu}{d^2\omega}\right) \tag{20}$$

The closure coefficients for the SST *k*-ω model are given in

Table 2.2.

Table 2.2: SST *k-ω* closure coefficients.

Closure Coefficient	Standard Value
β^*_{inf}	0.09
a_1	0.31
β_1	0.075
β_2	0.0828
σ_{k1}	0.85
σ_{k2}	1.0
σ_{w1}	0.5
σ_{w2}	0.856
κ	0.41

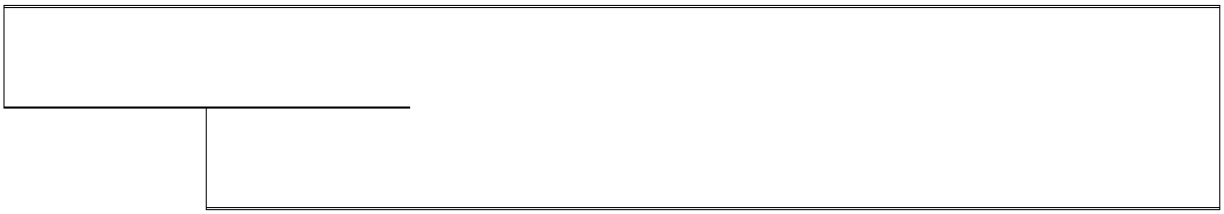
Chapter 3: OpenFOAM Verification

Before using OpenFOAM in the present study, the implementations of the SA and SST *k-ω* models in OpenFOAM were investigated. Examination of the source code of these models showed that they did not match the standard turbulence model definitions found on the NASA Langley Turbulence Model Resource (TMR) website [5]. The NASA TMR website provides a central location for the documentation of turbulence models and provides verification cases for testing turbulence model implementations. Corrections were made to the original code of the SA and SST *k-ω* models to match the definitions of the SA-noft2 [6] and SST-2003 [7] models from the TMR website. In what follows, the original implementations are denoted as SA-OF-Baseline and SST-OF-Baseline. The corrected implementations are denoted as SA-OF-Corrected and SSTkω-OF-Corrected. The results from the baseline and corrected models are compared to results from the NASA codes CFL3D [8] and FUN3D [9] to verify the newly implemented models.

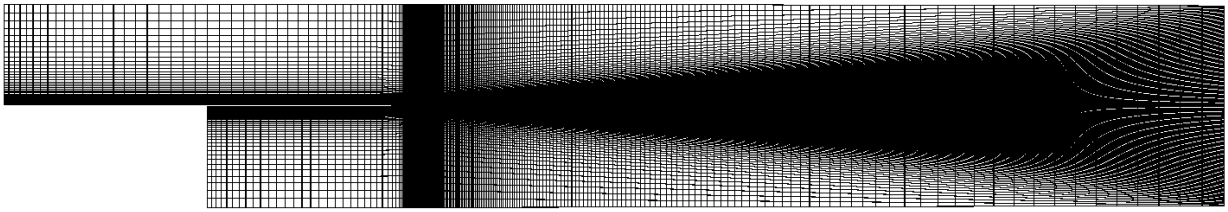
Three cases selected from the TMR website were run to verify the corrected models. These cases were the 2D mixing layer, the 2D wake, and the 2D flat plate boundary layer. Details of these cases are given below with results for the SA and SST *k-ω* models shown in the next sections. The three cases were run using a steady-state incompressible solver and second order discretization schemes.

- **2D Mixing Layer** - In this case, the upper higher-velocity stream and lower slower velocity stream mix to form a freeshear mixing layer that eventually achieves a self-similar solution. The reference Mach number of the upper stream is $M = 0.121108$ and Reynolds number $Re_L = 2900$. The reference static pressure P_{ref} for both streams was $101325\ Pa$. The geometry is shown in Figure 3.1(a). A family of computational grids was provided by the TMR website. Results presented below were achieved with the second-finest grid shown

in Figure 3.1(b). The quantities of interest for this case are the non-dimensional velocity at *x = 200, 650,* and *950 mm*. The baseline and corrected SA and SST *k-ω* models are compared to the CFL3D results taken from the TMR website.



(a)



(b)

Figure 3.1: 2D mixing layer (a) geometry and (b) mesh.

- **2D Wake** - In this case wake characteristics behind a non-symmetric 10% thick Model A-airfoil at angle of attack $\alpha = 0°$ were measured. Both the upper and lower surfaces of the airfoil were tripped. The definition of the airfoil was slightly altered to achieve a sharp trailing edge with chord length $c = 1$. The freestream Mach number was $M = 0.088$ and Reynolds number per chord was $Re_C = 1.2x10^6$. A family of computational grids is provided by the TMR website. Results presented below were achieved with the second-finest grid. The computational grid is shown in Figure 3.2(b). The quantities of interest for this case are the non-dimensional velocity profiles at *x/c = 1.05, 1.40,* and *2.19*. The baseline and corrected SA and SST *k-ω* models are compared to the CFL3D results taken from the TMR website.

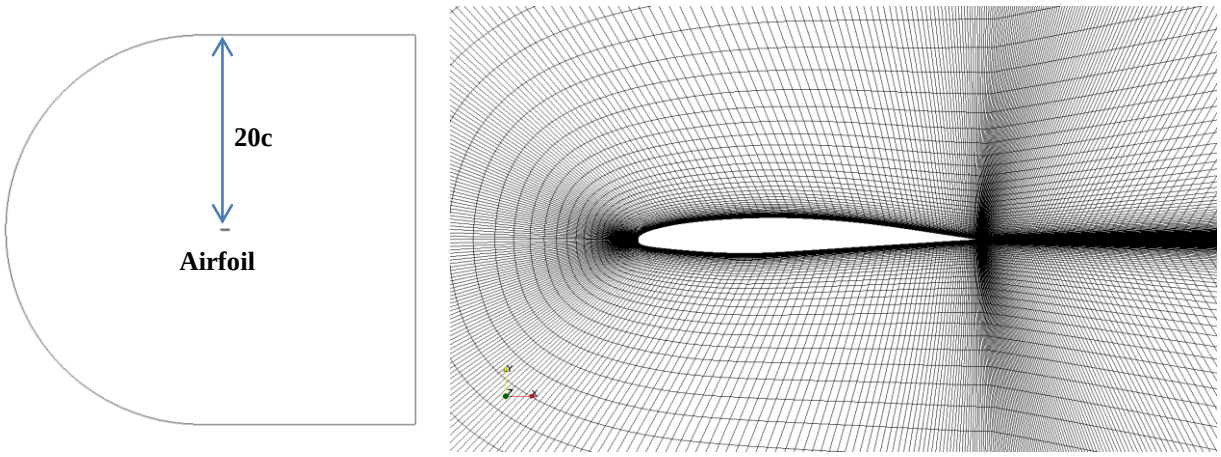


Figure 3.2: 2D wake (a) geometry and (b) mesh.

- **2D Flat Plate -** In this case, characteristics of a flat plate turbulent boundary layer were measured. The case had a Mach number of $M = 0.2$ and Reynolds number $Re = 5x10^6$. A family of computational grids is provided by the TMR website. The skin friction coefficient along the plate and the normalized velocity profile at $x = 0.97$ was given. Skin friction and velocity results presented below were achieved with the second-finest grid. The computational grid is shown in Figure 3.4. Additionally the convergence behavior of the skin friction coefficient is also presented.

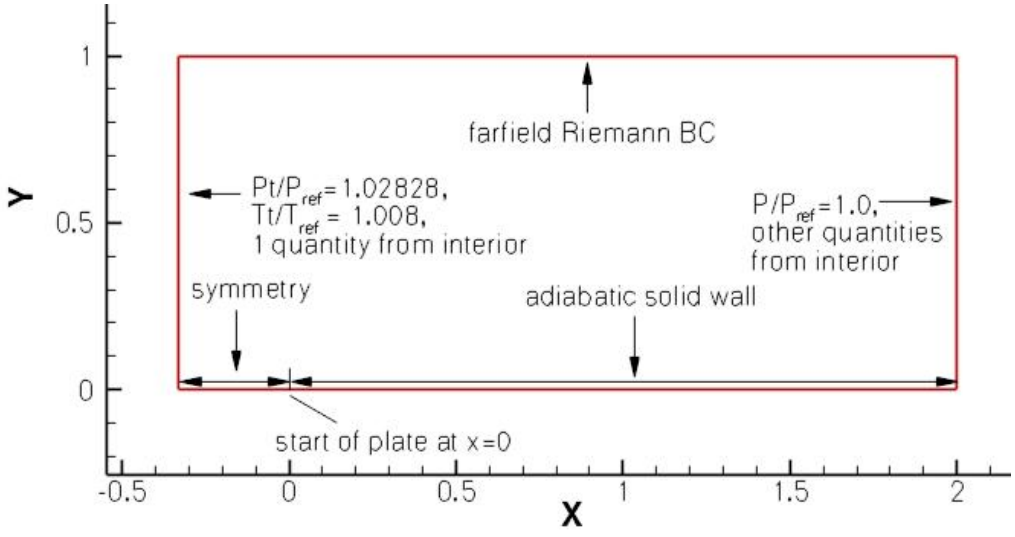


Figure 3.3: 2D flat plate geometry and boundary conditions [5].

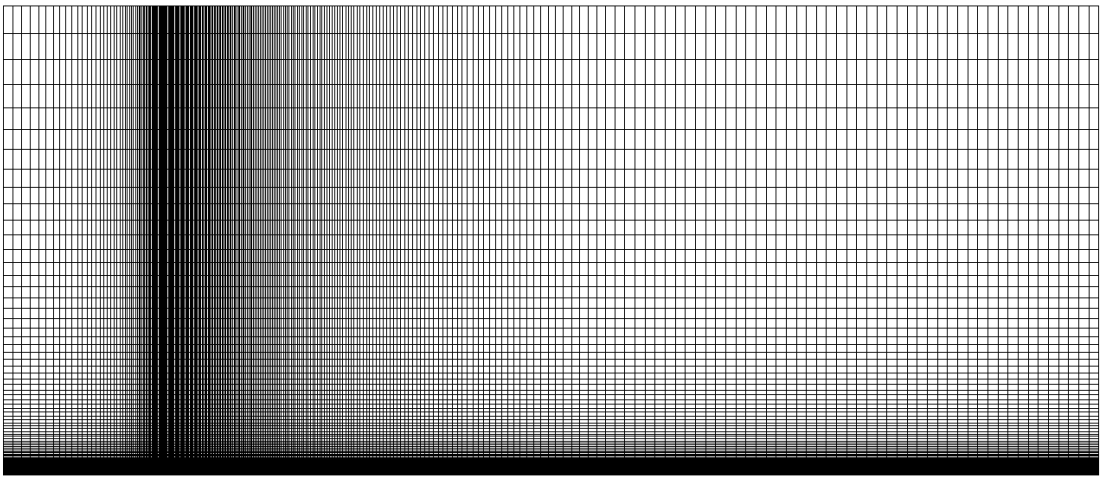


Figure 3.4: 2D flat plate mesh.

3.1 Spalart-Allmaras Model Refinements and Verification

The main difference between the SA-OF-Baseline and SA-OF-Corrected model is the inclusion of the f_{v3} term in the SA-OF-Baseline model. This form of the SA model is "not recommended because of unusual transition behavior at low Reynolds numbers. Unfortunately, coding of this version still persists"[10]. After removing this term, the f_{v2} and $\tilde{S}$ definitions were corrected. The f_{t2} term found in the SA-Standard model was not added but is not expected to influence the solutions for the high Reynolds number cases examined [11]. The SA-noft2 code is available in Appendix A of this dissertation.

The SA-OF-Corrected model was first tested on the 2D mixing layer and 2D wake cases. The non-dimensionalized velocity at *x = 200, 650,* and *950 mm* are shown in Figure 3.5, Figure 3.6, and Figure 3.7 respectively. It can be seen that the both the SA-OF-Baseline and SA-OF-Corrected results agree very well with the SA-CFL3D results. Some discrepancy in the core of the predicted 2D wake flow is seen in Figure 3.8. This difference in the predicted velocity is most pronounced near the airfoil trailing edge. Overall the models are in very good agreement for these two free shear cases.

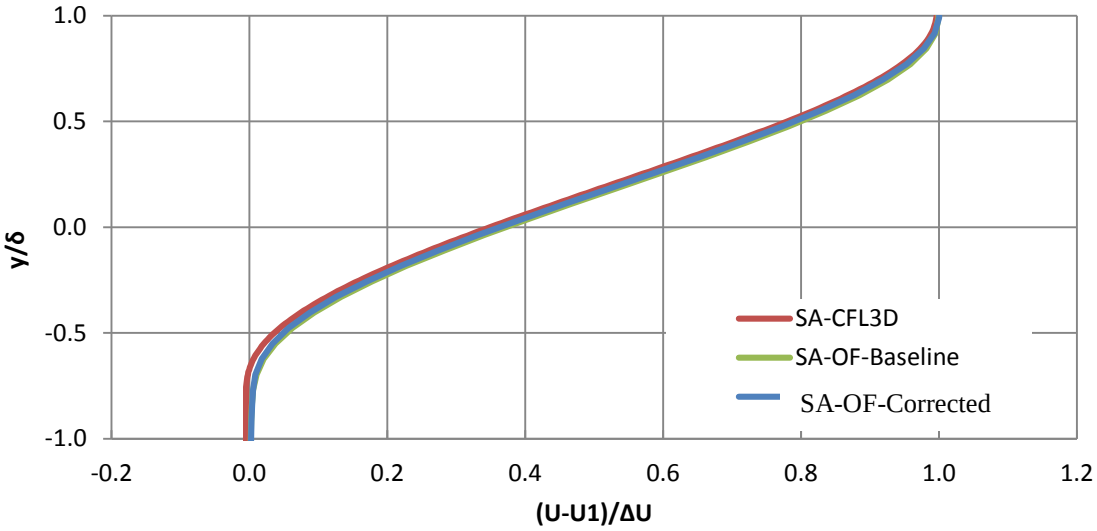


Figure 3.5: SA model verification for the 2D mixing layer at x = 200 mm.

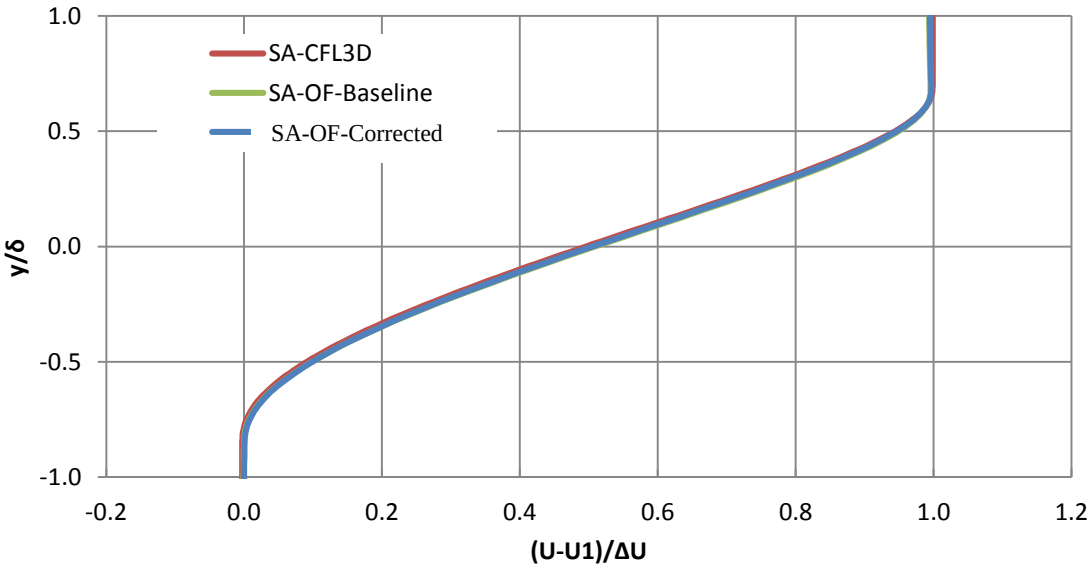


Figure 3.6: SA model verification for the 2D mixing layer at x = 650 mm.

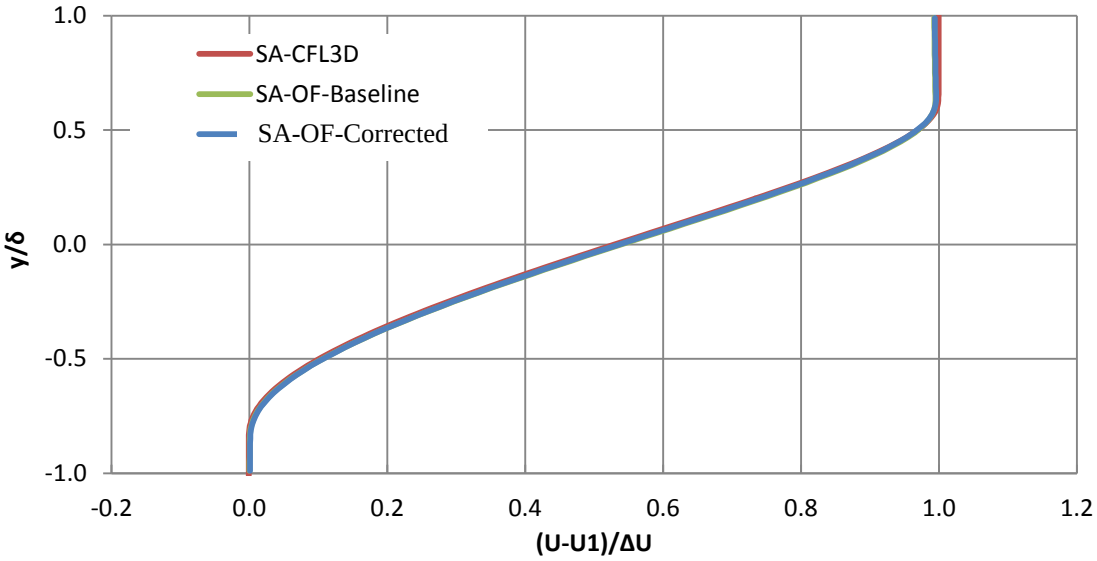


Figure 3.7: SA model verification for the 2D mixing layer at x = 950 mm.

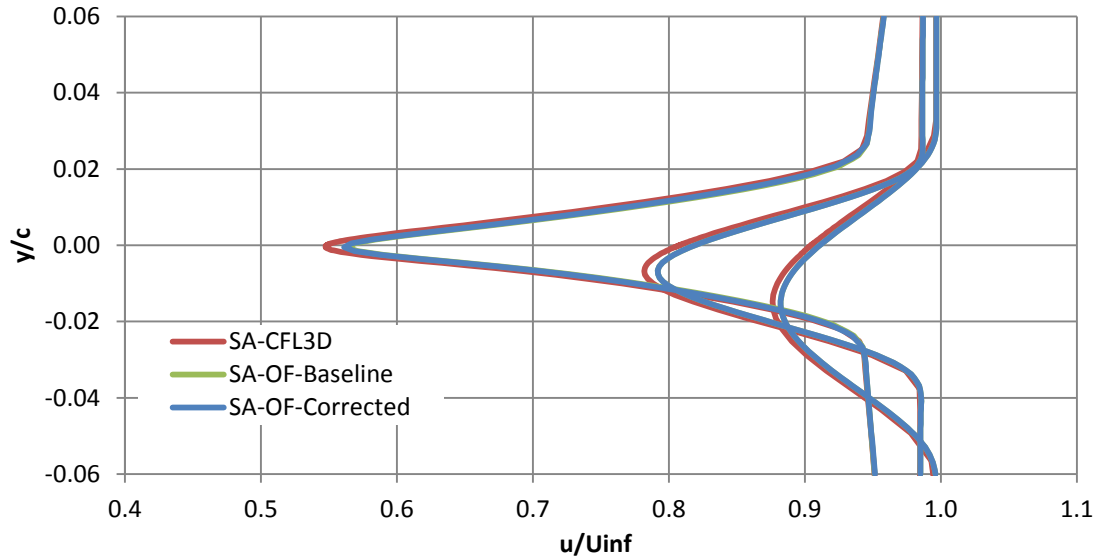


Figure 3.8: SA model verification for the 2D wake verification at x/c = 1.05, 1.40, and 2.19.

The next case used to verify the implementation of the SA-OF-Corrected model was the 2D flat plate turbulent boundary layer. Comparisons of the velocity profile at $x = 0.97$ and the skin friction coefficient along the plate are shown in Figure 3.9 (a) and (b) respectively. It can be seen that the corrected SA model shows some improvement in the velocity and skin friction predictions. Also the baseline SA model predicts some transitional behavior near the leading edge of the plate which

is not seen in the corrected SA model and FUN3D model. The skin friction coefficient of the corrected SA model was also improved as can be seen in Figure 3.10

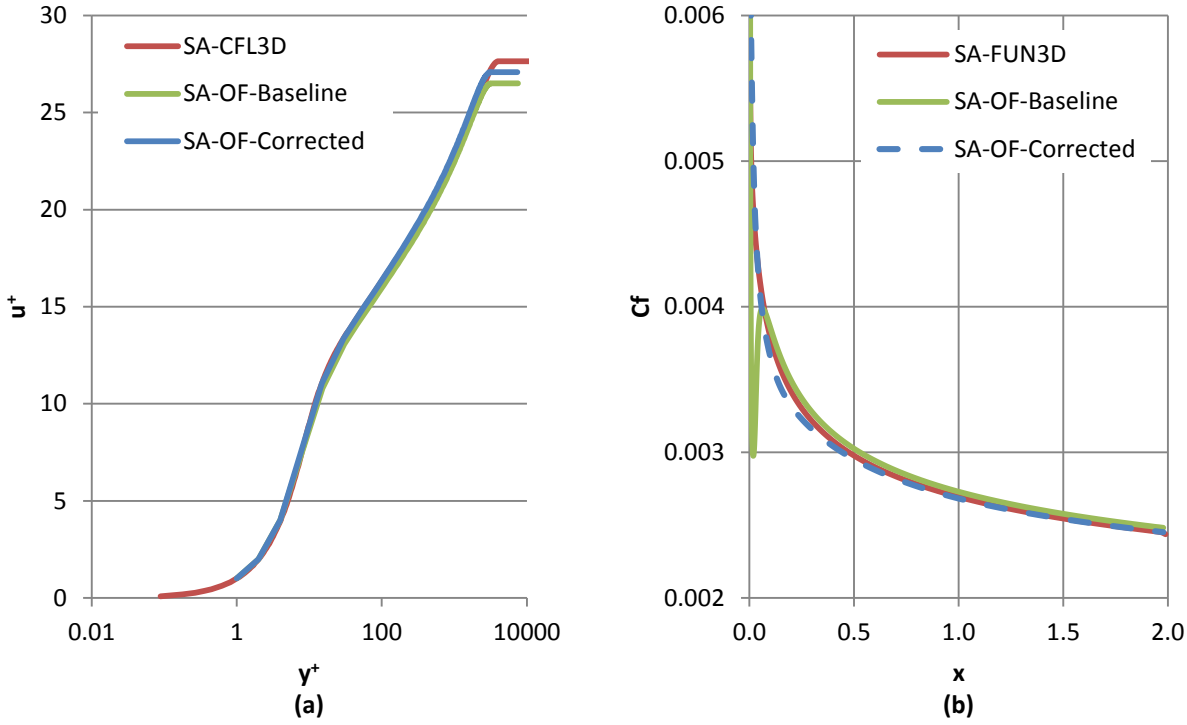


Figure 3.9: SA model verification of the (a) velocity and (b) skin friction coefficient.

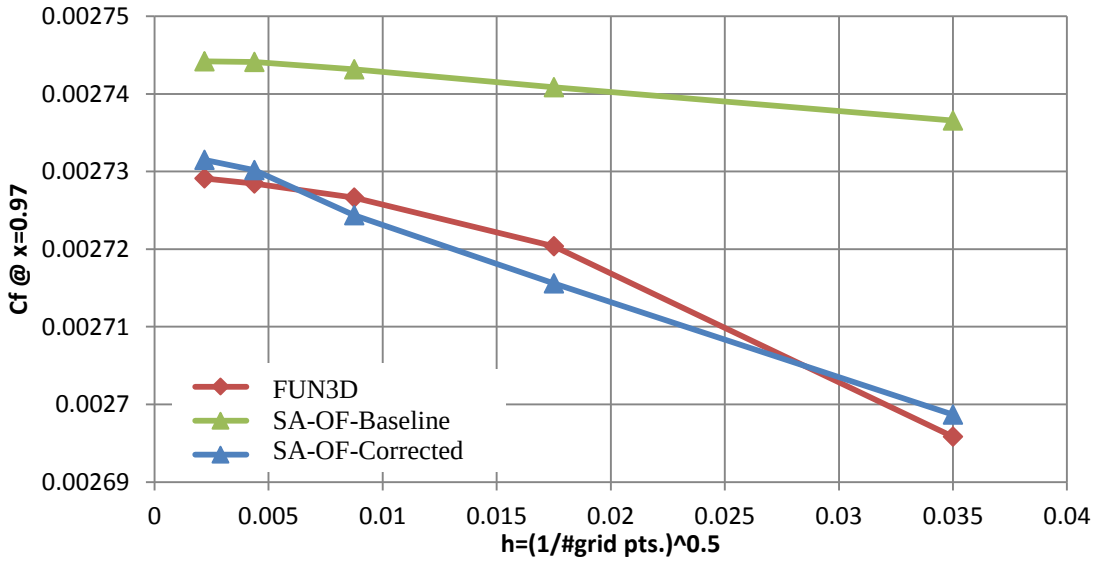


Figure 3.10: SA verification of the skin friction coefficient convergence.

3.2 SST *k-ω* Model Refinements and Verification

The SSTkω-OF-Baseline model includes an extra function, F_3, not found in any of the versions present on the TMR website. After removing this function, changes were made in the definition of the eddy-viscosity and $CD_{k\omega}$. Also a limiter was placed on both the k and ω production terms. The biggest change between the SSTkω-OF-Baseline and SSTkω-OF-Corrected model was the removal of wall functions. While this change is not specifically outlined on the TMR website, it was done to ensure that the proper wall boundary conditions were being applied. The SST *k-ω* model is integrable down to the wall but the use of wall functions risks overwriting the computed near wall flow. Wall functions were disabled in OpenFOAM and the TMR recommended ω_{wall} and k_{wall} boundary conditions were applied in the SSTkω-OF-Corrected model. The SST-2003 code is available in Appendix A of this dissertation.

The free shear flows of the 2D mixing layer and 2D wake were first used to verify the SSTkω-OF-Corrected model. The 2D wake non-dimensionalized velocity at *x = 200, 650,* and *950 mm* are shown in Figure 3.11, Figure 3.12, and Figure 3.13 respectively. It can be seen that the SST-OF-Baseline model gives very poor results compared to SSTkω-CFL3D. The SSTkω-OF-Corrected model is in very good agreement with SSTkω-CFL3D except at x=200mm. The results of the 2D wake are shown in Figure 3.14. Again it is evident that some errors exist in the SSTkω-OF-Baseline model. The implementation of the SSTkω-OF-Corrected model greatly improves the accuracy of OpenFOAM.

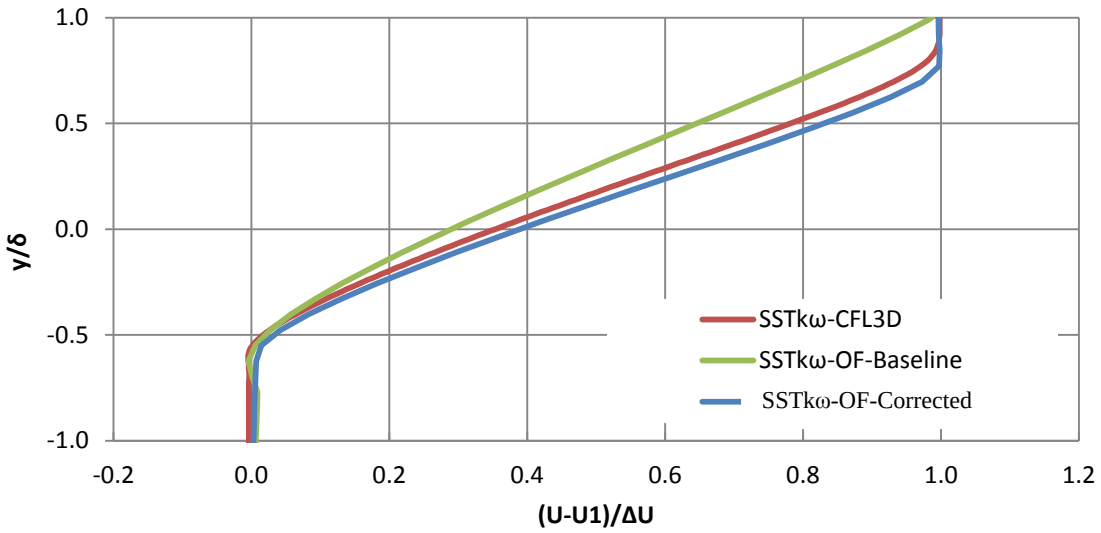


Figure 3.11: SST *k-ω* model verification of the 2D wake at x = 200mm.

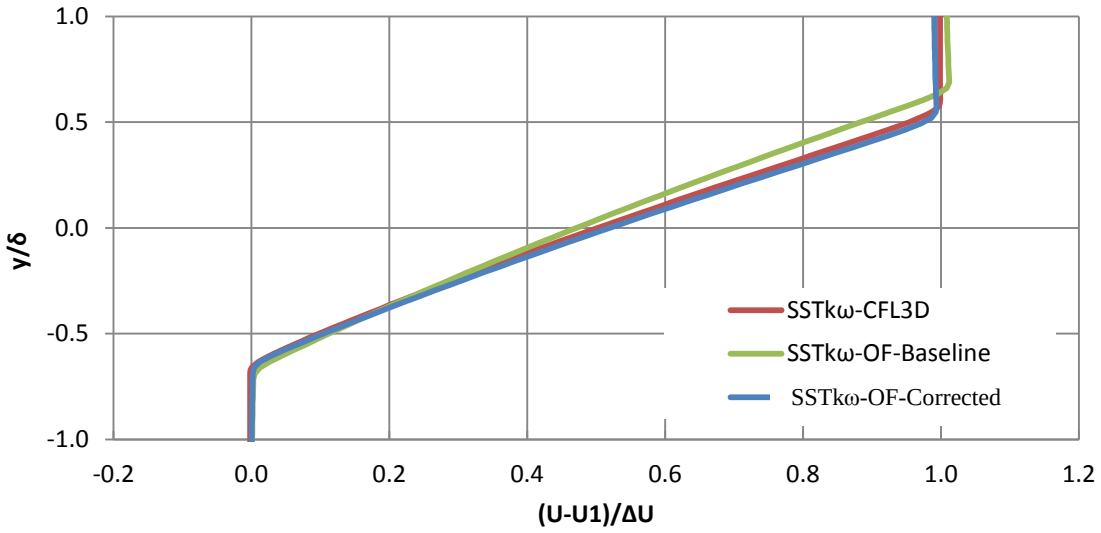


Figure 3.12 SST *k-ω* model verification of the 2D wake at x = 650 mm.

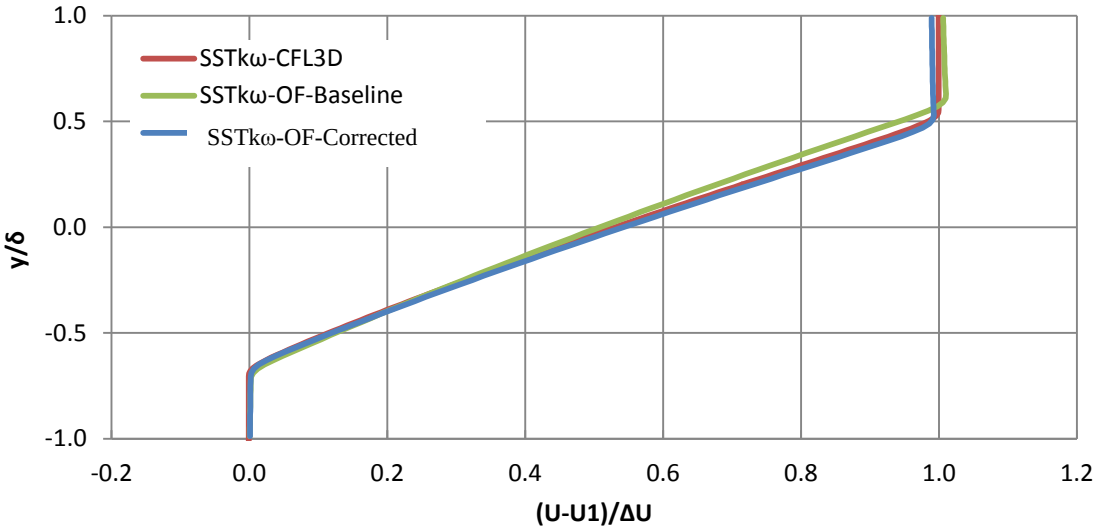


Figure 3.13: SST *k-ω* model verification of the 2D wake at x = 950 mm.

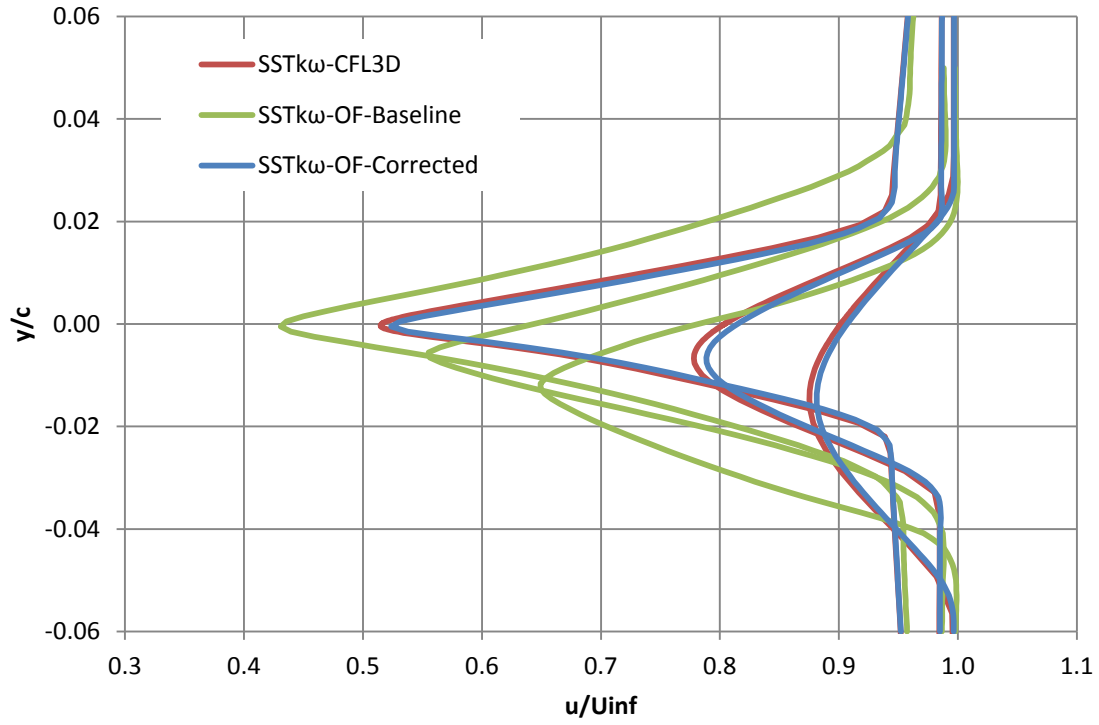


Figure 3.14 SST *k-ω* model verification of the 2D wake at x = 1.05, 1.40, and 2.19.

The 2D flat plate turbulent boundary was used next to verify the implementation of the SSTkω-OF-Corrected model. Comparisons of the velocity profile at $x = 0.97$ and the skin friction coefficient along the plate are shown in Figure 3.15 (a) and (b) respectively. It can be seen that the

corrected model is in much better agreement with FUN3D results than the baseline model. The SSTkω-OF-Baseline model relies on wall functions to correctly predict the skin friction coefficient. With wall functions disabled the baseline model cannot accurately simulate turbulent boundary layers.

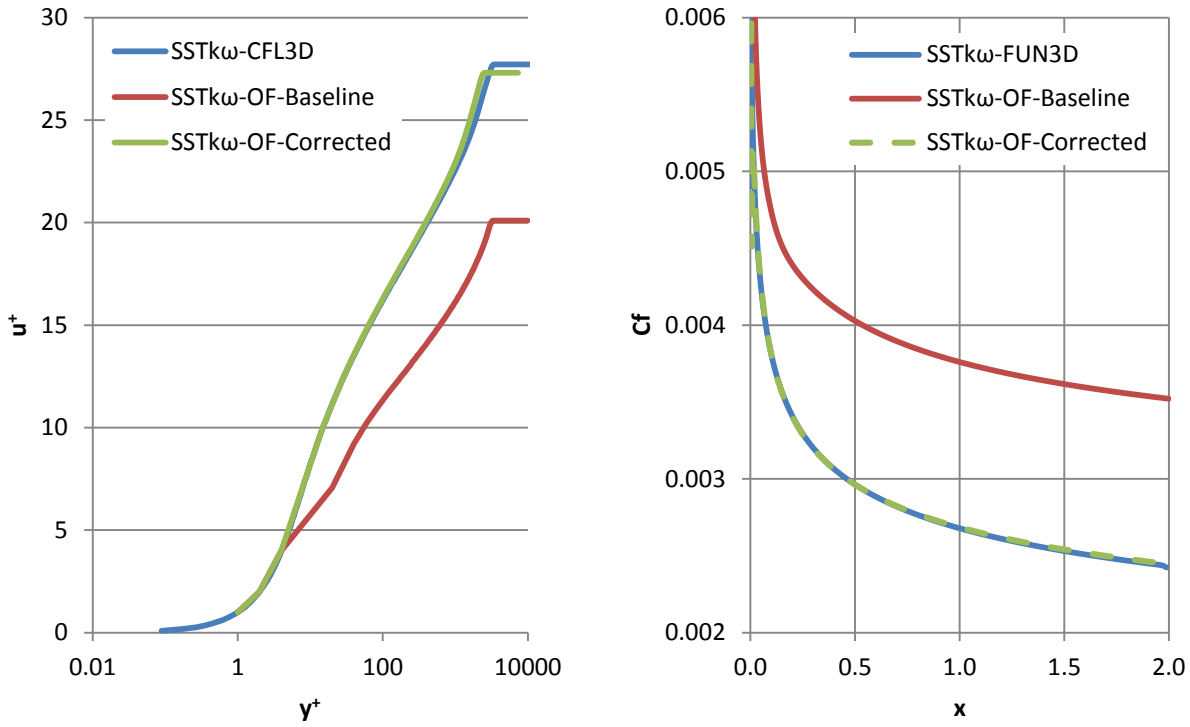


Figure 3.15: SST *k-ω* verification of the (a) velocity and (b) skin friction coefficient.

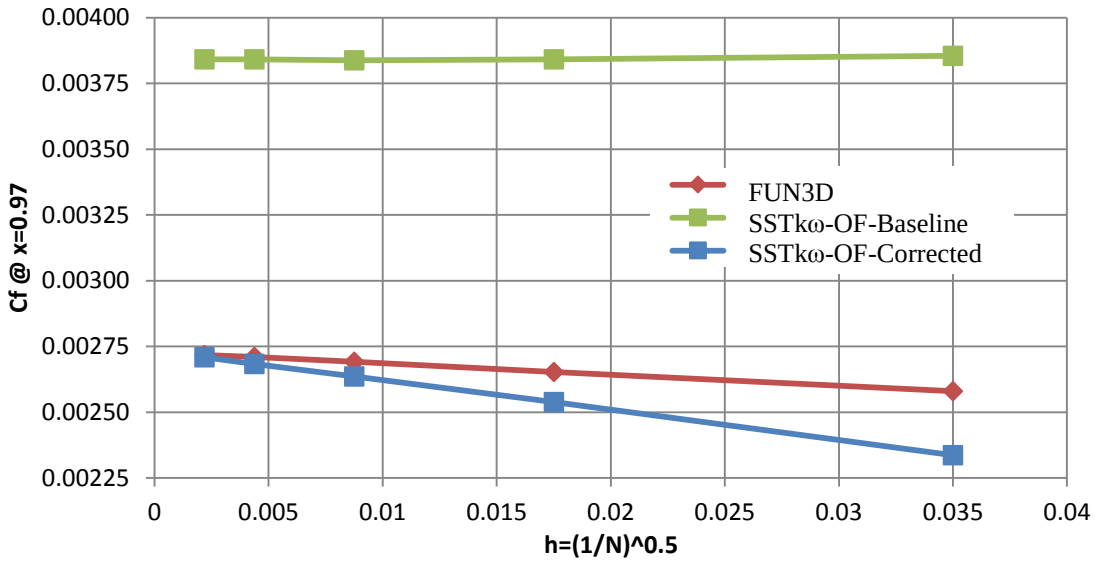


Figure 3.16: SST *k-ω* verification of the skin friction coefficient convergence.

The results of these three cases demonstrate that the newly implemented SA and SST *k-ω* models are in excellent agreement with NASA's CFL3D and FUN3D implementations. The minor discrepancies that remain may be due to running OpenFOAM's incompressible solver while CFL3D/FUN3D run compressible solvers or it could be due to the selection of different discretization schemes in OpenFOAM and CFL3D/FUN3D. With the SA and the SST *k-ω* models verified, results from OpenFOAM can be directly compared with confidence to results from other CFD codes and results from the literature.

Chapter 4: The Wray-Agarwal (WA) Turbulence Model

4.1 Introduction

Baldwin and Barth [12] were among the first to derive a one-equation turbulence model by transforming and simplifying a two-equation model. However, the assumptions made in their derivation resulted in the model having a very different behavior then the parent k-ε model. Additionally the model was ill-conditioned near shear layer regions leading to extreme numerical difficulties. Menter [13] refined this transformation methodology and used it to propose his own one-equation model also based on k-ε closure. Menter's one-equation model showed very similar and even improved results compared to its parent k-ε model [13]. Since then several one-equation models based on k-ε and k-ω closure have been proposed but have found limited adoption by the scientific community due to their poor numerical behavior, difficulty in implementation, or poor accuracy compared to the industry standard models. Goldberg [14] developed a one-equation model functionally similar to the Baldwin-Barth model, except that a switch function was used to 'turn off' the problem term when needed. The switch function was non-Galilean-invariant and the model did not see wide adoption. Nagano et. al [15], who have also done extensive work on two-equation models, derived their one-equation model by transforming their k-ε model. However, the complexity of their two-equation model led to many non-linear and singular terms in their one-equation model. Fares and Schröder [16] proposed a one-equation model based on k-ω closure. Their model showed good results for free shear flows and some wall bounded adverse pressure gradient flows but was later shown to be sensitive to freestream boundary conditions [17]. Elkhoury [18] refined Menter's one-equation model by limiting the production term and later adding a switch to the destruction term. Rahman et. al [19] took experience gained with their

previous algebraic stress model and two-equation model to propose a model similar to Baldwin-Barth with additional functions to capture non-equilibrium effects and enforce realizability constraints. The model showed good results for several canonical cases but is complex and difficult to implement. Rahman et. al [20,21] have also recently proposed a one-equation *k*-model and one-equation *ε*-model that have shown accurate prediction of near wall *k* and ε profiles compared to DNS. Remarkably, despite being one of the first one-equation models, the Spalart-Allmaras model has remained the benchmark one-equation model due to its simplicity, ease of use, and excellent numerical characteristics. The goal of this research has been to develop a one-equation model that combines the most desirable characteristics of the one-equation *k-ε* models and one-equation *k-ω* models, analogous to the SST *k-ω* model's combination of the two-equation *k-ε* and *k-ω* models. It is shown that the new model is generally more accurate than the SA model and is competitive with the SST *k-ω* model for a wide range of flows.

4.2 Derivation of the WA model

The Wilcox's 2006 *k-ω* model [4] in the boundary layer coordinates can be written as:

$$\frac{Dk}{Dt} = \frac{\partial}{\partial y}\left(\sigma_k \frac{k}{\omega}\frac{\partial k}{\partial y}\right) + \nu_t \left(\frac{\partial u}{\partial y}\right)^2 - \beta^* k\omega \tag{21}$$

$$\frac{D\omega}{Dt} = \frac{\partial}{\partial y}\left(\sigma_\omega \frac{k}{\omega}\frac{\partial \omega}{\partial y}\right) + \alpha \frac{\omega}{k}\nu_t \left(\frac{\partial u}{\partial y}\right)^2 - \beta\omega^2 + \frac{\sigma_d}{\omega}\frac{\partial k}{\partial y}\frac{\partial \omega}{\partial y} \tag{22}$$

where the material derivative $D/Dt = \partial/\partial t + u_j\left(\partial/\partial x_j\right)$. Boundary layer coordinates are used for simplicity and the model coefficients are not defined to emphasize the generality of this approach. It should also be noted that the last term of the *ω*-equation was not present when Fares and Schröder derived their one equation model. This cross diffusion term will be responsible for additional terms in the resulting one-equation. With *R* defined as *k/ω*, the material derivative of *R* can be obtained as:

$$\frac{DR}{Dt} = \frac{1}{\omega}\frac{Dk}{Dt} - \frac{k}{\omega^2}\frac{D\omega}{Dt} \tag{23}$$

Substitution of Eq. 21 and Eq. 22 in Eq. 23 defines the new transport equation for *R*. However, two independent equations are still necessary to remove *k* and ω from the new transport equation for *R*. The first of these has already been defined with $R = k/\omega$. The closure is completed using the Bradshaw's relation [22, 23] given in Eq. 24, where a_1 is the Bradshaw's constant and $\left|\overline{-\overline{u'v'}}\right|$ is the turbulent shear stress.

$$|-\overline{u'v'}| = \nu_t \left|\frac{\partial u}{\partial y}\right| = a_1 k \tag{24}$$

After substitution and the additional assumption that $\sigma_k = \sigma_\omega = \sigma_R$, the *R*-Equation can be obtained as:

$$\frac{DR}{Dt} = \frac{\partial}{\partial y}\left(\sigma_R R \frac{\partial R}{\partial y}\right) + C_1 R \left|\frac{\partial u}{\partial y}\right| + C_2 \frac{R}{\left|\frac{\partial u}{\partial y}\right|}\frac{\partial R}{\partial y}\frac{\partial \left|\frac{\partial u}{\partial y}\right|}{\partial y} - C_3 R^2 \left(\frac{\frac{\partial \left|\frac{\partial u}{\partial y}\right|}{\partial y}\frac{\partial \left|\frac{\partial u}{\partial y}\right|}{\partial y}}{\left|\frac{\partial u}{\partial y}\right|^2}\right) \tag{25}$$

After comparison with previous one-equation models, it can be noted that the C_2 term is identical to the destruction term of one-equation *k*-ω models (Fares and Schröder) while the C_3 term is identical to the destruction term in one-equation *k*-ε models (Menter). Therefore it is possible to control the behavior of the new model by controlling which destruction term is active. The two-equation *k*-ω model has a strong dependency on the free stream values which has been shown to carry over into the one-equation *k*-ω model. To minimize this effect, the C_2 destruction term is limited to be employed near solid boundaries and away from shear layer edges. The aim is to design a blending function f_1 which has a value of one in the viscous sub-layer and in the majority of the log layer. Then, near the outer edge of the log-layer, it decreases to zero allowing the C_3

term to dominate. This approach is analogous to the F_1 blending function of the SST k-ω model. The F_1 blending function of the SST k-ω has the form as follows

$$f_1 = \tanh(arg_1^4) \tag{26}$$

Hyperbolic tangent is a desirable function because it is bounded and smooth. Use of it in this model gives better numerical behavior than the "if-then" and "min-max" switch functions of Goldberg and Elkhoury. The remaining problem is to find a formula for arg_1 such that arg_1=1.0 in the log-layer and zero away from the wall. Noting that in the log-layer $R = u_t \kappa d = S\kappa^2 d^2$, the following definition of arg_1 is suggested. The coefficient C_b is used to control how fast the transition from one to zero occurs.

$$arg_1 = C_b \frac{\nu + R}{S\kappa^2 d^2} \tag{27}$$

The coefficients of the new one-equation model were calibrated by applying the model to free shear flows. Similarity solutions were derived for four standard free shear cases, namely the plane wake, the planer jet, the round jet, and the radial jet. The equations were cast into self-similar solutions following the procedure of Wilcox [24] and numerically solved. The spreading rates for these four free shear flows are shown in Table 4.1.

Table 4.1: Free shear flow spreading rates

Flow	R-Eqn.	SA [24]	SST k-ω [25]	Experiment
Far Wake	0.305	0.341	0.258	0.32-0.40 [26]
Plane Jet	0.108	0.157	0.112	0.10-0.11 [27]
Round Jet	0.119	0.248	0.127	0.086-0.096 [28]
Radial Jet	0.093	0.166	---	0.096-0.110 [29]

It can be seen from Table 4.1 that the predicted spreading rates using the *R*-Equation are in much better agreement than the SA model for the plane, round, and radial jets. For the far wake flow,

the SA model is in best agreement with the experimental value followed by the *R*-Equation model and lastly by SST *k-ω* model. The model coefficients are given in Eqs. (32) – (36). The law of the wall was also used as a constraint when determining the model coefficients.

For appropriate model behavior in the viscous sub-layer and buffer layer, further modification to the model is required. The wall blocking effect is accounted for by the damping function of Mellor and Herring [30] shown in Eq. (31). The value of C_w was determined by calibrating the model to a simple flat plate flow. A value of C_w=*13.0* is used and ν has the usual definition of dynamic viscosity.

The final form of the new zonal one-equation model, known as the Wray-Agarwal (WA) model is as follows:

$$\frac{\partial R}{\partial t}+\frac{\partial u_j R}{\partial x_j}=\frac{\partial}{\partial x_j}\left[(\sigma_R R+\nu)\frac{\partial R}{\partial x_j}\right]+C_1 RS+f_1 C_{2k\omega}\frac{R}{S}\frac{\partial R}{\partial x_j}\frac{\partial S}{\partial x_j} - (1-f_1)C_{2k\varepsilon}R^2\left(\frac{\frac{\partial S}{\partial x_j}\frac{\partial S}{\partial x_j}}{S^2}\right) \tag{28}$$

$$\nu_T=f_\mu R \tag{29}$$

$$f_1=tanh(arg_1^4), \qquad arg_1=C_b\frac{\nu+R}{S\kappa^2 d^2} \tag{30}$$

$$f_\mu=\frac{\chi^3}{\chi^3+C_w^3}, \qquad \chi=\frac{R}{\nu} \tag{31}$$

$$C_{1k\omega}=0.0833, \qquad C_{1k\varepsilon}=0.16 \tag{32}$$

$$C_1=f_1(C_{1k\omega}-C_{1k\varepsilon})+C_{1k\varepsilon} \tag{33}$$

$$C_{2k\omega}=1.22, \qquad C_{2k\varepsilon}=1.95 \tag{34}$$

$$\sigma_R = f_1(\sigma_{k\omega} - \sigma_{k\varepsilon}) + \sigma_{k\varepsilon} \tag{35}$$

$$\sigma_{k\omega} = 0.72, \qquad \sigma_{k\varepsilon} = 1.0 \tag{36}$$

$$C_w = 8.54, \qquad C_b = 1.66 \tag{37}$$

$$S = \sqrt{2S_{ij}S_{ij}}\,, \qquad S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{38}$$

4.3 Characteristics of the WA model

The WA model was first applied to compute free shear flows. Since these flows are inexpensive and quick to solve numerically, they can be used to easily investigate some properties of the new model. Solution sensitivity to free stream boundary conditions is an issue that plagues many two-equation models and especially the *k-ω* models [31]. Since a *k-ω* closure was used in the derivation of the WA model, it must be determined whether this behavior is also present in the new one-equation model. Figure 4.1 shows the plane jet spreading rates for the WA and SA models for varying free stream non-dimensional eddy viscosity, *N*, on two grids. It can be seen that both one-equation models have no free stream dependency. The WA model however is more sensitive to the number of grid points than the SA model but it is much more accurate than the SA model in calculation of the spreading rate when compared to the experimental data as shown in Table 4.1.

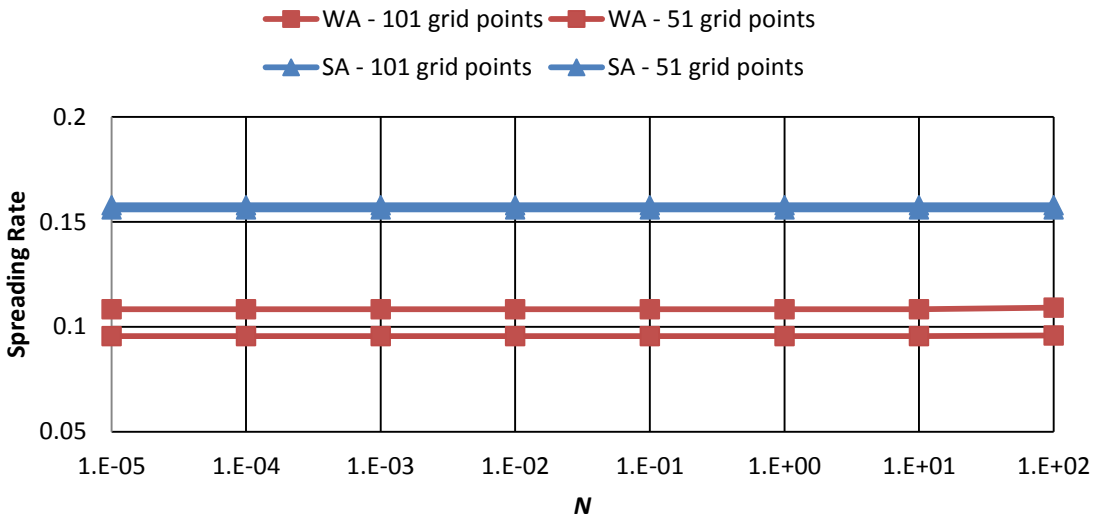


Figure 4.1: Plane jet spreading rates as a function of free stream eddy viscosity *N* and number of grid points.

One-equation models are attractive in part due to their numerical efficiency. A simple 2D subsonic boundary layer flow past a flat plate is used to investigate the convergence behavior of the models. As can be seen from Figure 4.2, the SA model converges quite well to its final value showing very little mesh sensitivity. This is somewhat expected since special consideration was given to the numerical behavior during its formulation. The WA and SST k-ω models have a similar order of convergence. The computational cost per iteration of the three models was also measured for the finest mesh of computation of boundary flow past flat plate on a single processor. The SA model ran the fastest with 0.090 seconds per iteration, followed by the WA model with a speed of 0.125 seconds per iteration. The SST k-ω model expectedly was the slowest with a CPU time of 0.185 seconds per iteration. All models were run in serial on the same desktop computer. All models converged in nearly the same number of iterations, making the SA model the least computationally intensive followed by the WA model and lastly by the SST k-ω model. This trend in CPU time and convergence was observed for all the computed cases in the following chapters.

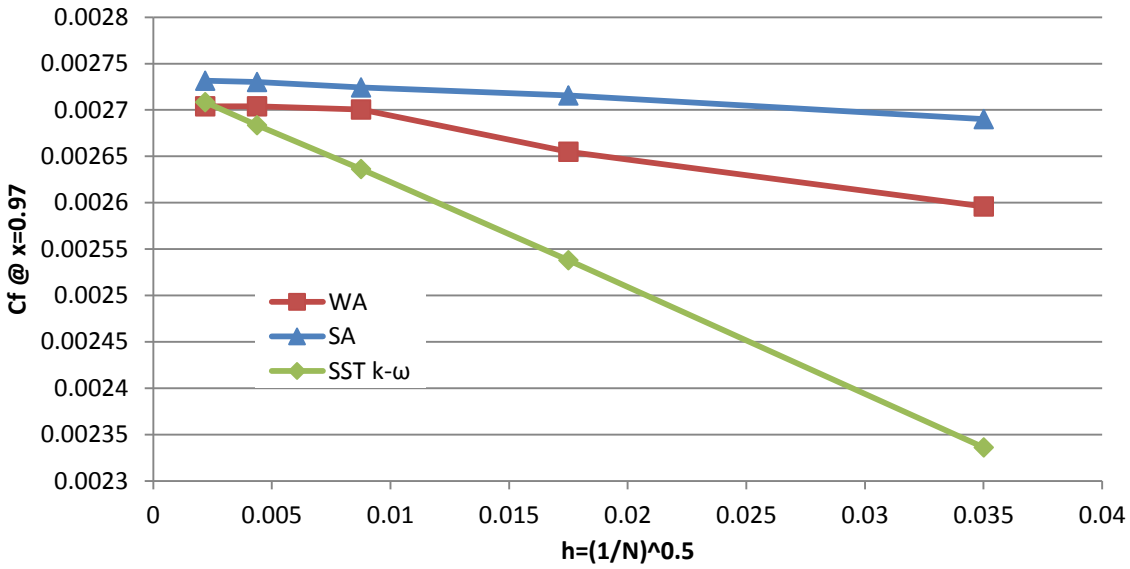


Figure 4.2: Convergence of skin friction coefficient at x = 0.97.

Chapter 5: Validation Cases

5.1 Introduction

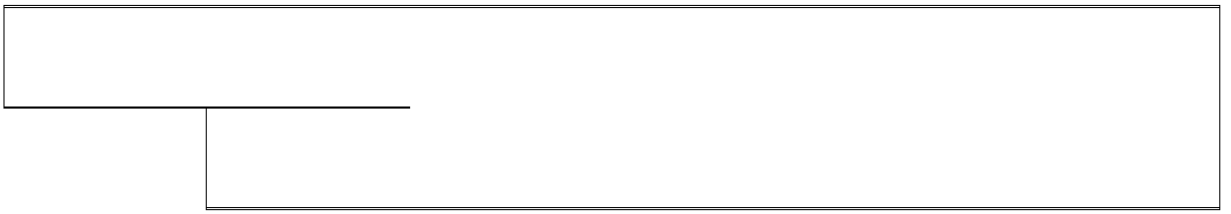
In Chapter 3, the correct implementations of the SA and SST k-ω models were verified for a few canonical flows. In this chapter comparison of the WA, SA, and SST k-ω models to experimental data are made. A wide variety of validation cases are calculated. The majority of the cases are selected from the NASA Langley Research Center Turbulence Modeling Resource (TMR) [5] or the National Project for Application-oriented Research in CFD (NPARC) Alliance [32]. A few additional canonical cases from literature are also computed. Cases are broken into the following sections: Subsonic Flows, Transonic Flows, and Supersonic Flows. Details of the flow conditions, geometry, and grid are given for each case. Whenever possible, TMR-provided grids are used so that a direct comparison to TMR results can be made. The variety of cases considered demonstrates the WA model's ability to accurately reproduce a wide range of flows.

5.2 Subsonic Flows

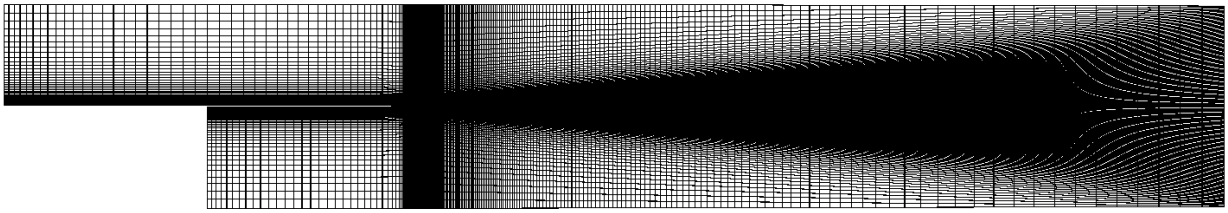
5.2.1 2D Mixing Layer

In this section results for the TMR 2D Mixing Layer are presented. The experimental data is taken from Deville et. al [33]. In this case, the upper higher-velocity stream and lower slower velocity stream mix to form a freeshear mixing layer that eventually achieves a self-similar solution. The reference Mach number of the upper stream is $M = 0.121108$ and Reynolds number $Re_L = 2900$. The reference static pressure P_{ref} for both streams is $101325\ Pa$. The geometry is shown in Figure 5.1(a). A family of computational grids is provided by the TMR website. Results presented below were obtained with the second-finest grid. The computational grid is shown in Figure 5.1(b).

The quantities of interest for this case are: the non-dimensional velocity profiles at *x = 200, 650,* and *900* mm. The vorticity thickness δ_w from the experiment was used to nondimensionalize the y-coordinate. It can be seen in Figure 5.2-5.4 that the velocity profiles from all three models are in good agreement with the experimental data.



(a)



(b)

Figure 5.1: 2D mixing layer (a) geometry and (b) mesh.

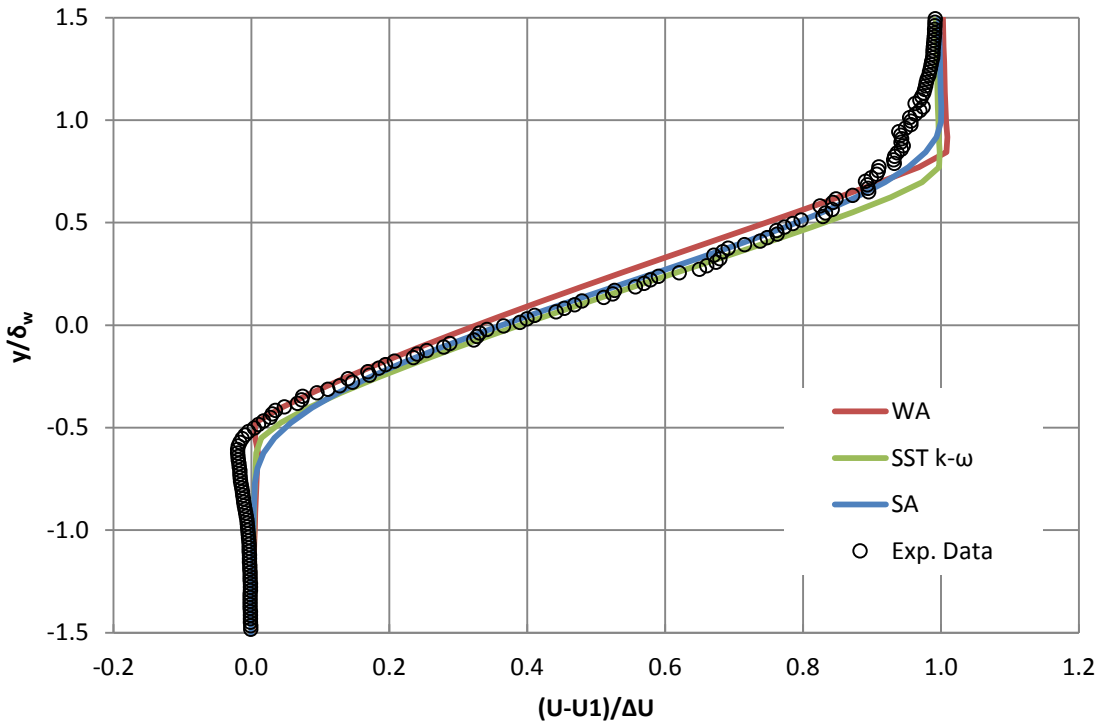


Figure 5.2: Normalized velocity profile comparisons at x = 200mm.

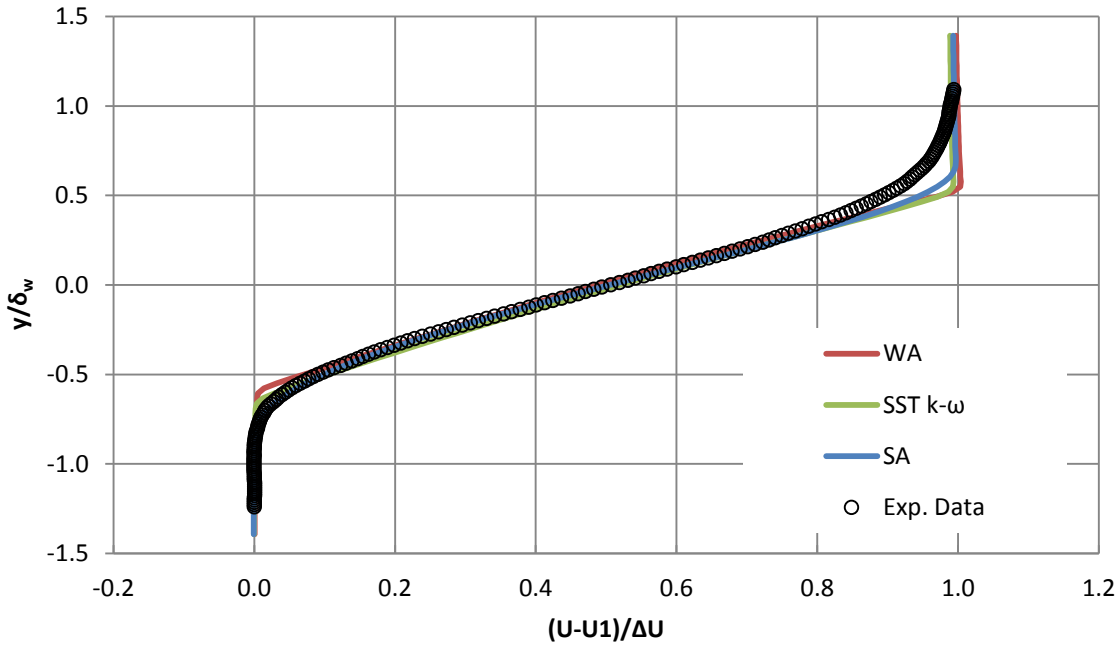


Figure 5.3: Normalized velocity profile comparisons at x = 650mm.

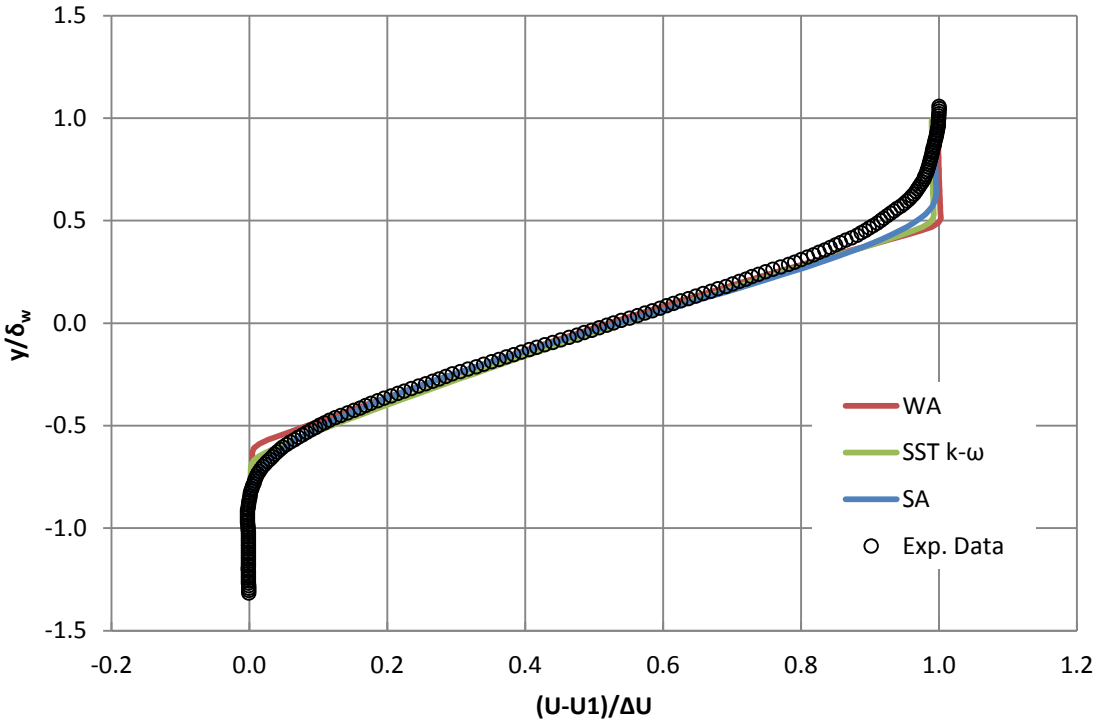


Figure 5.4: Normalized velocity profile comparisons at x = 950mm.

5.2.2 2D Wake

In this section results for the TMR 2D Airfoil Near-Wake are presented. In this case wake characteristics behind a non-symmetric 10% thick Model A-airfoil at angle of attack $\alpha = 0°$ were measured. Both the upper and lower surfaces of the airfoil were tripped. Experimental test conditions for this case are given by Nakayama. [34] The definition of the airfoil was slightly altered to achieve a sharp trailing edge with chord length $c = 1$. The freestream Mach number is $M = 0.088$ and Reynolds number per chord is $Re_C = 1.2x10^6$. A family of computational grids is provided by the TMR website. Results presented below were achieved with the second-finest grid. The computational grid is shown in Figure 5.5:. The quantities of interest for this case are: the non-dimensional velocity profiles at $x/c = 1.05$, 1.40, and 2.19. As seen in Figure 5.6-5.7 all three models are in general agreement with the experimental data.

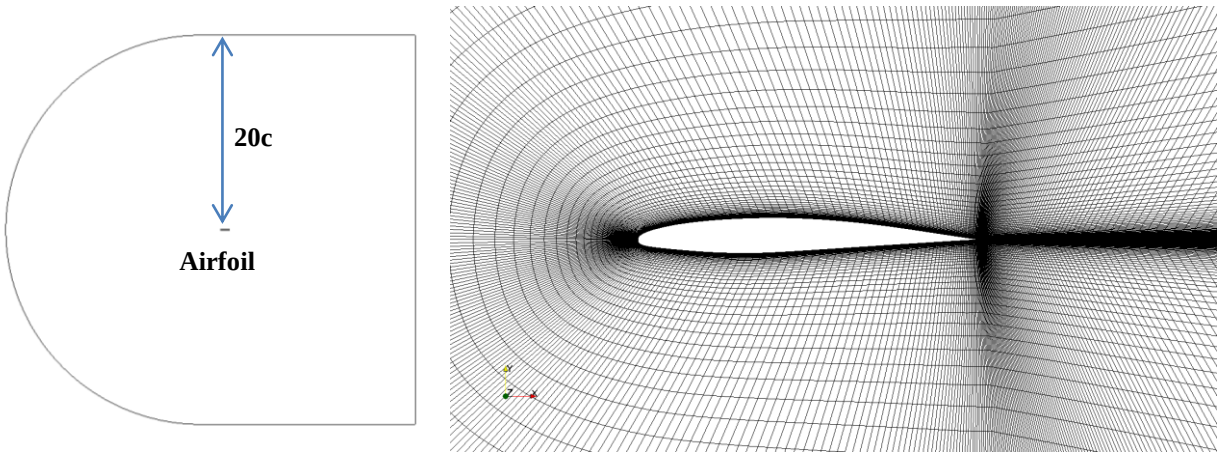


Figure 5.5: 2D wake (a) geometry and (b) mesh.

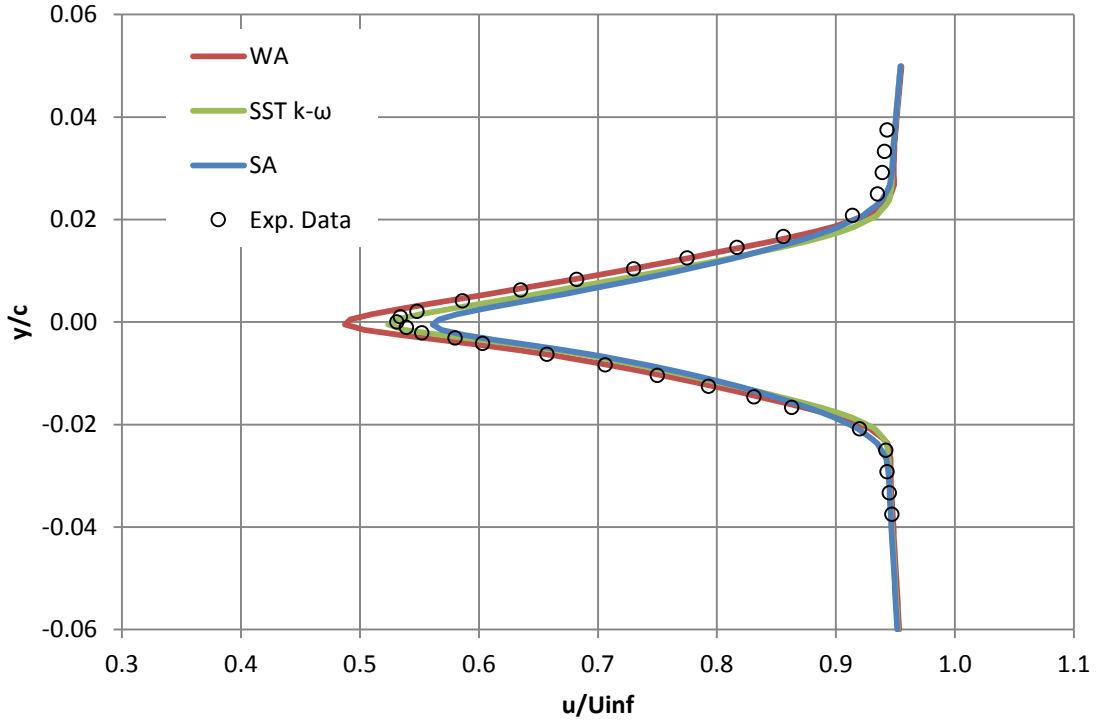


Figure 5.6: Normalized velocity profile comparisons at x=1.05.

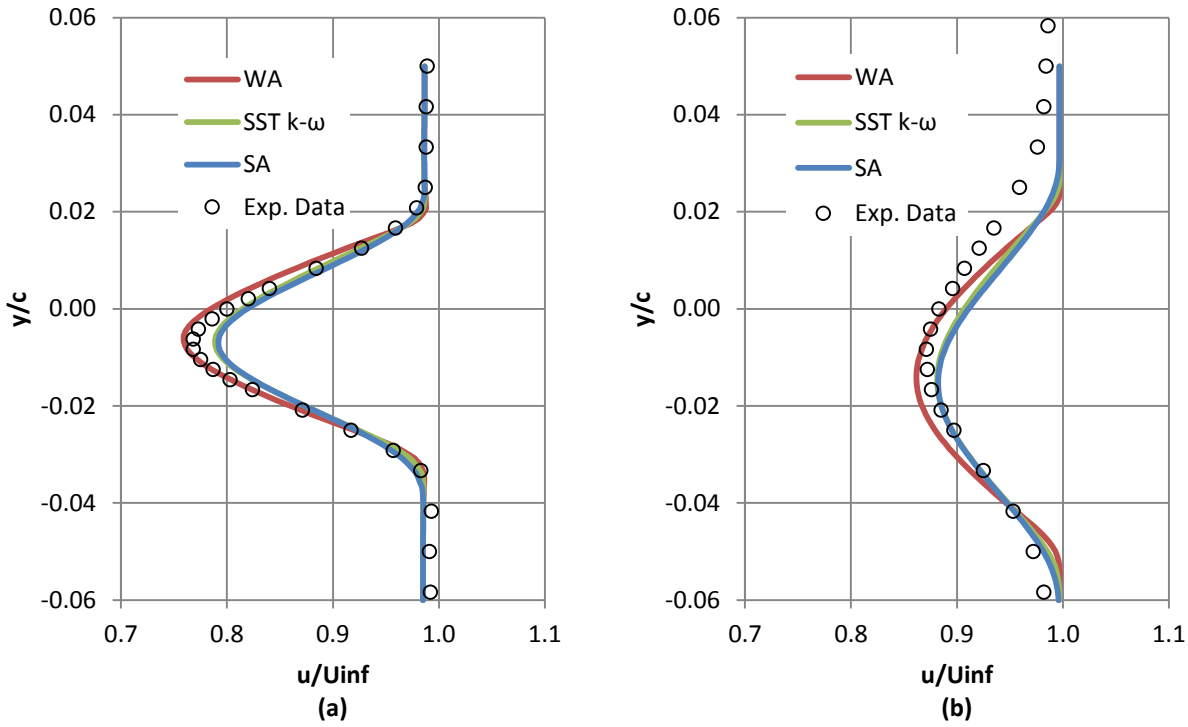


Figure 5.7: Normalized velocity profile comparisons at (a)x=1.40 and (b)x=2.19.

5.2.3 2D Flat Plate

In this section results for the TMR 2D Zero Pressure Gradient Flat Plate are presented. This case has a Mach number $M = 0.2$ and Reynolds number $Re = 5x10^6$. A family of computational grids is provided by the TMR website. Results presented below were obtained with the second-finest grid. The computational grid is shown in Figure 5.9.

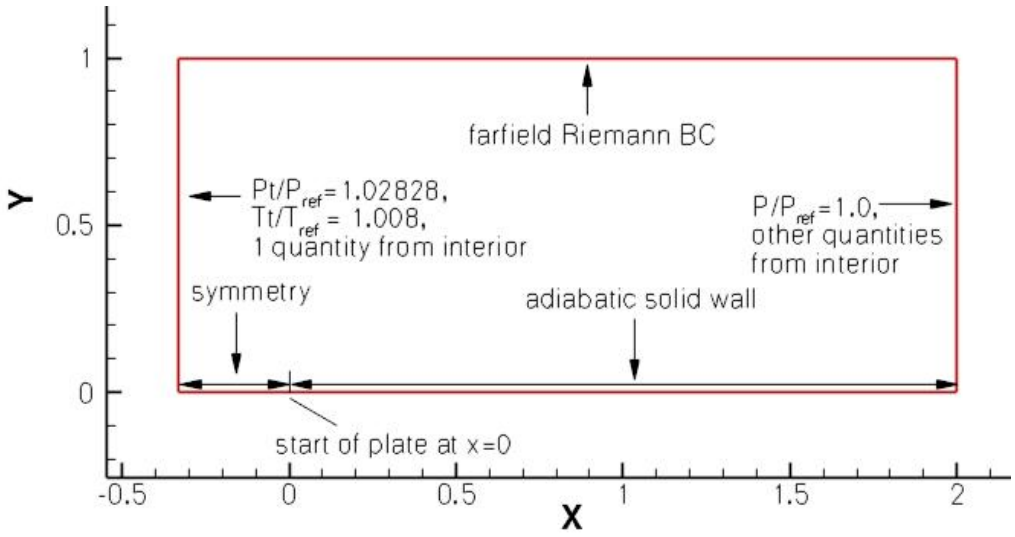


Figure 5.8: 2D flat plate geometry and boundary conditions [5].

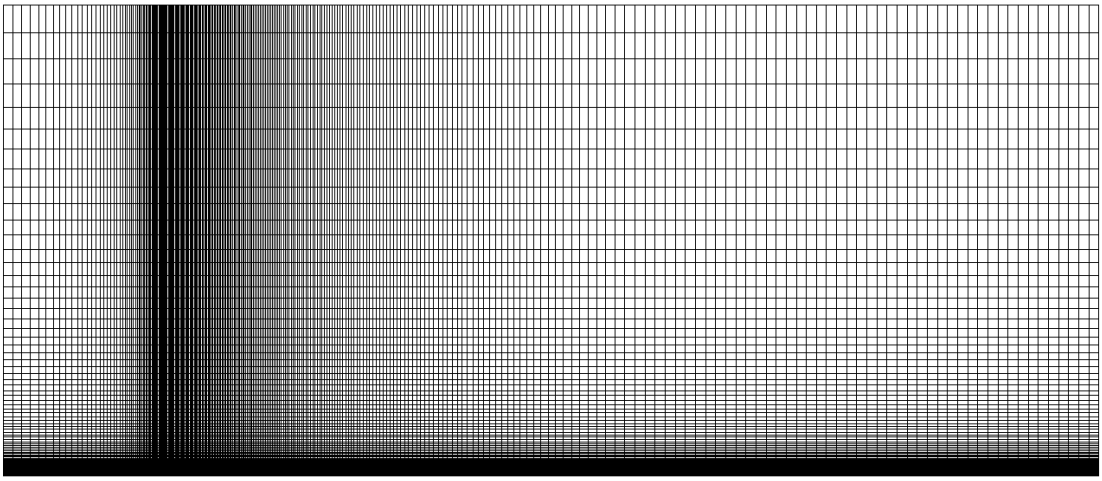


Figure 5.9: 2D flat plate mesh.

The quantities of interest to compare are: the skin friction coefficient C_f along the plate and the non-dimensional velocity u^+ at $Re_\theta = 5{,}000$ and $10{,}000$. Experimental data from Weighardt[35] as well as empirical formulas from Coles[36] are used for comparison. Calculation of the momentum thickness θ is necessary for determining Re_θ. This calculation required numerical integration of the boundary layer. Evaluation of the integral for θ was carried out by an additional post-processing code not included in OpenFOAM.

Figure 5.10 compares the skin friction coefficient, C_f, computed using the WA, SA, and SST k-ω models. It can be seen that all the models predict the correct profile of skin friction coefficient along the plate. Figure 5.11 compares the computed velocity profiles using the three models. As expected for this simple case, the results from all the models compare extremely well with the experimental data and the empirical formula.

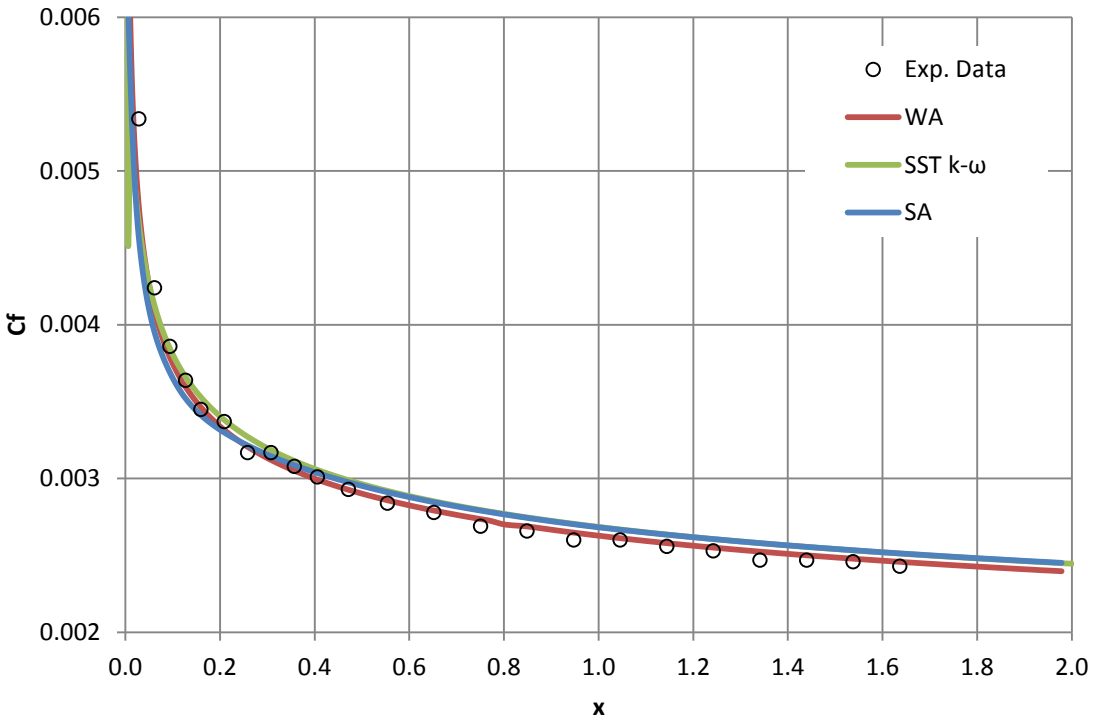


Figure 5.10: Skin friction coefficient for flow over a 2D flat plate.

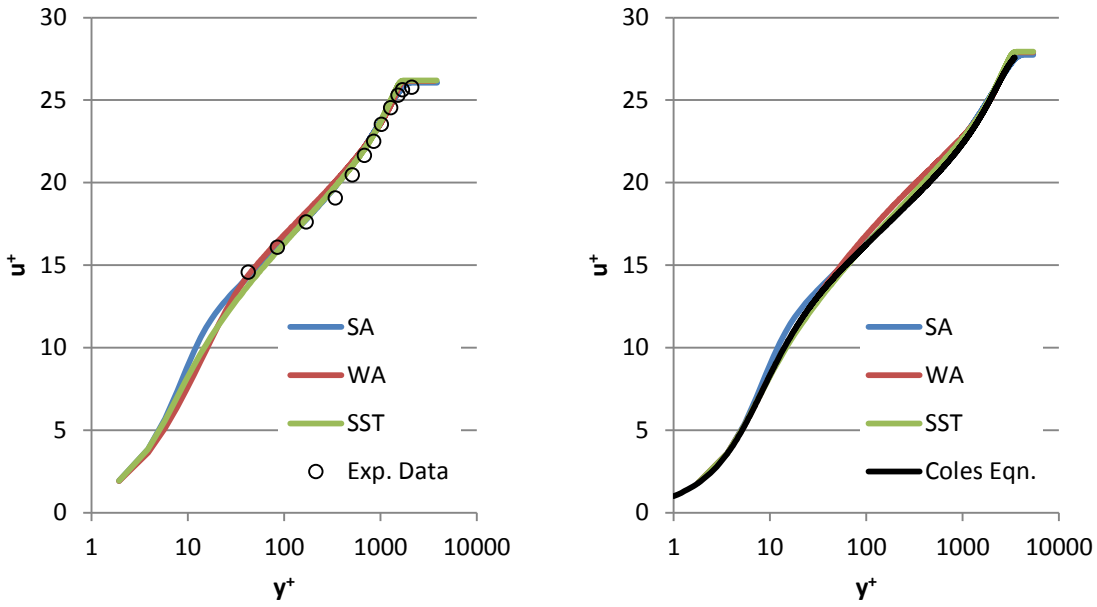


Figure 5.11: Velocity profiles at (a) Re_0=5000 and (b) Re_0=10,000.

5.2.4 2D Channel

In this section results for a fully developed turbulent channel flow at $Re_\tau = 550$ and $Re_\tau = 1000$ are presented. The calculation is conducted in a half-width channel, imposing periodic boundary conditions. A pressure gradient source is applied to achieve the necessary Re_τ. The grid was refined for the $Re_\tau = 1000$ case until mesh independence was achieved. This mesh was then used for the $Re_\tau = 550$ case. Figure 5.12 shows the comparison of predicted velocity profiles with the DNS results of Lee and Moser [37]. For both Reynolds numbers, excellent agreement between the predicted and DNS results can be seen.

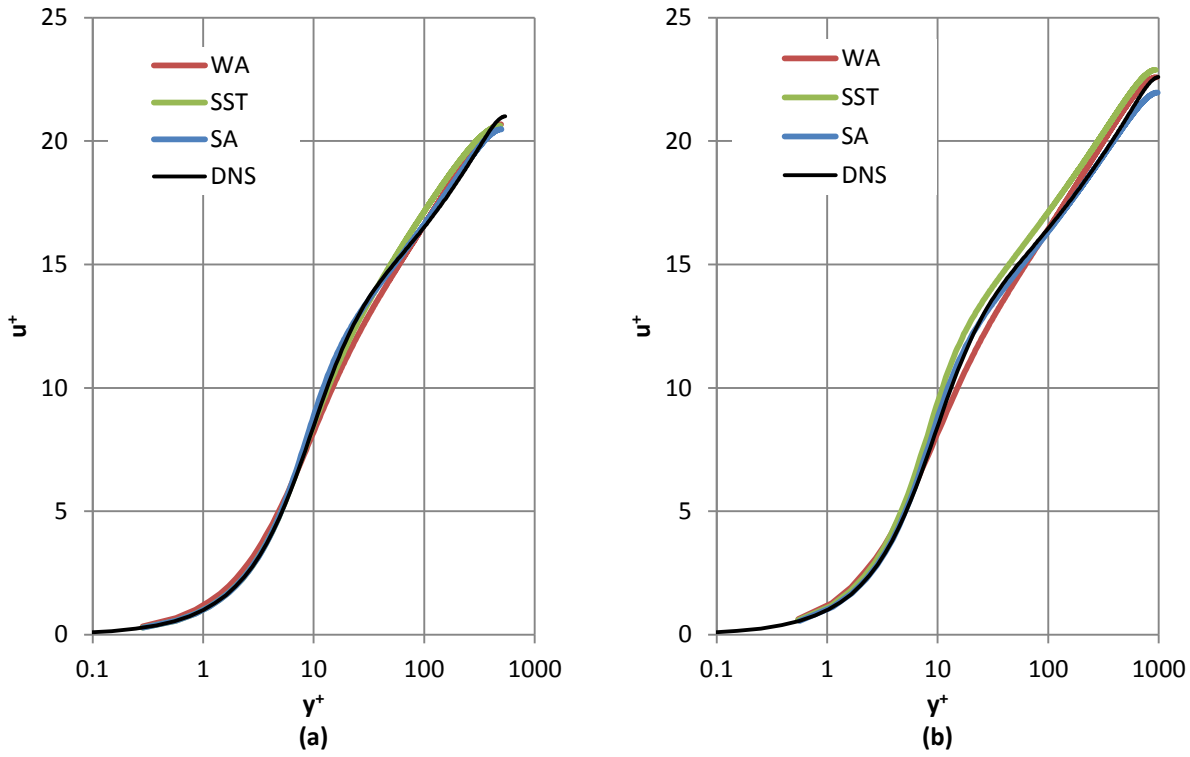


Figure 5.12: Comparison of nondimensional velocity profiles for fully developed channel flow at Re_τ = (a) 550, and (b) 1000.

5.2.6 2D NACA0012 Airfoil

In this section results for the TMR 2D NACA0012 Airfoil are presented. Experimental test conditions for this case are given by Ladson. [38] Both the upper and lower surfaces of the airfoil were tripped. The definition of the airfoil was slightly altered to achieve a sharp trailing edge with chord length $c = 1$. The freestream Mach number is $M = 0.15$ and the Reynolds number per chord is $Re_C = 6x10^6$. A family of computational grids is provided by the TMR website. Results presented below were achieved with the second-finest grid. The quantities of interest to compare are: the lift coefficient C_l, the drag coefficient C_d, the surface pressure coefficient C_p and the surface skin friction coefficient C_f at angles of attack $\alpha = 5°$, $10°$, and $15°$. Surface pressure coefficients are compared to the data of Gregory [39] which better captured the 2D effects, while lift and drag coefficients are compared to the experiment of Ladson[38].

The predicted lift and drag coefficients are compared to experimental values in Table 5.1. It can be seen that the WA model under predicts the lift coefficient and over predicts the drag at each angle of attack. The SA and SST models are both reasonably close to the experimental value.

Table 5.1: Lift and drag predictions for the NACA0012 airfoil.

	α=5°		*α=10°*		*α=15°*	
Model	C_l	C_d	C_l	C_d	C_l	C_d
WA	0.544134	0.009680	1.069435	0.013018	1.473077	0.023764
SST *k-ω*	0.549202	0.008485	1.079623	0.010850	1.537168	0.017703
SA	0.551979	0.009023	1.082537	0.011731	1.537900	0.020150
Exp.	--	--	~1.076	~0.0118	~1.519	~0.0185

Figure 5.13-5.15 show the comparison of pressure coefficient and skin friction coefficient at α = *5°*, *10°*, and *15°* respectively obtained using the three models.

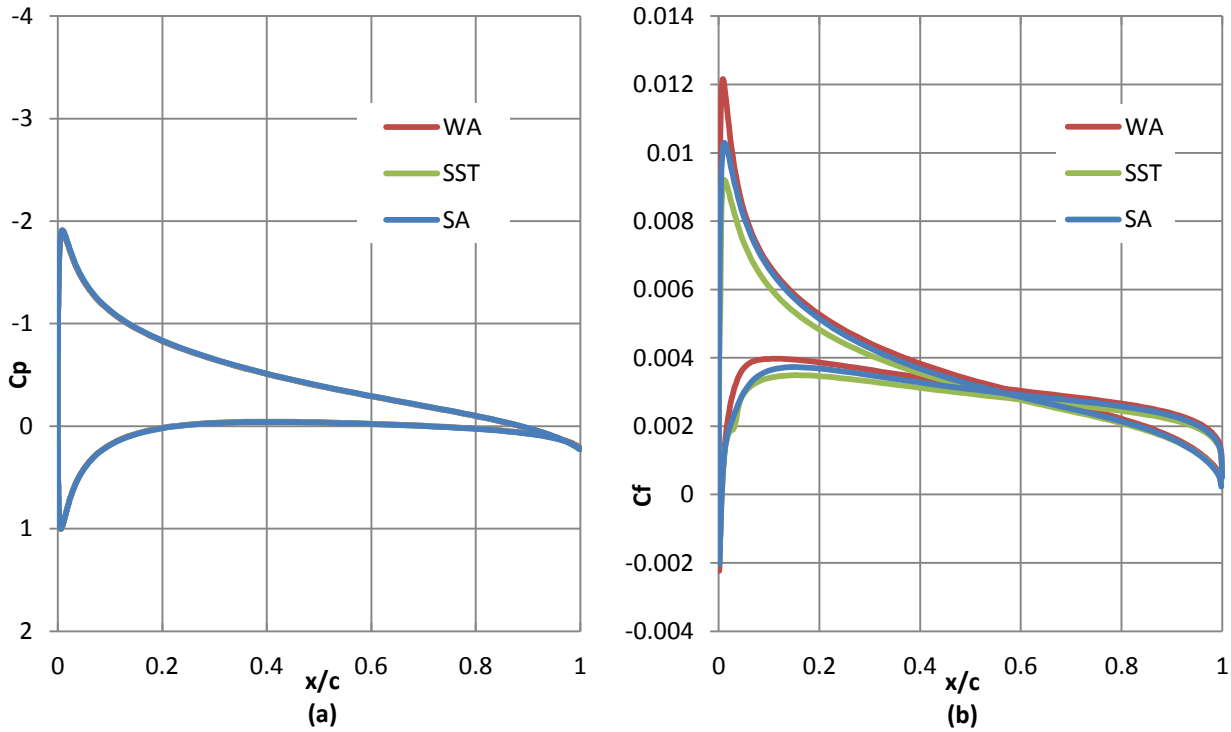


Figure 5.13: (a)Pressure coefficient and (b)skin friction coefficient comparisons for the NACA0012 at *α*=5°.

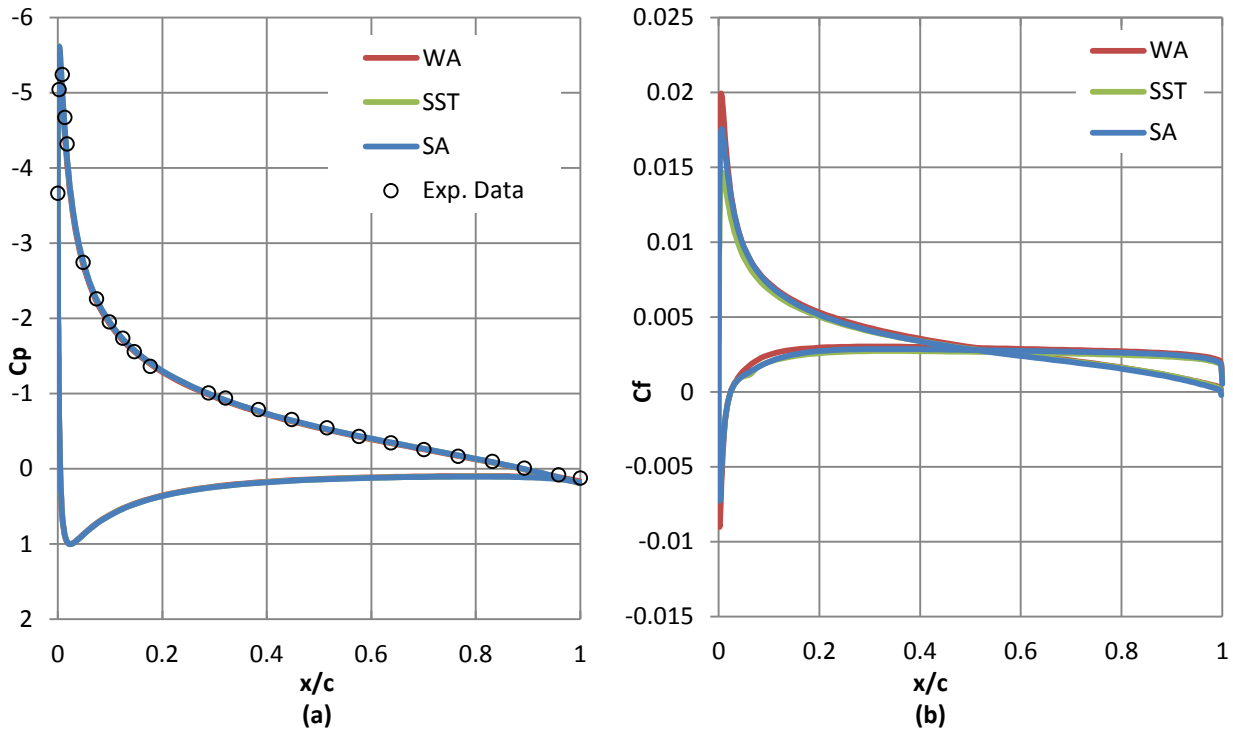


Figure 5.14: (a)Pressure coefficient and (b)skin friction coefficient comparisons for the NACA0012 at α=10°.

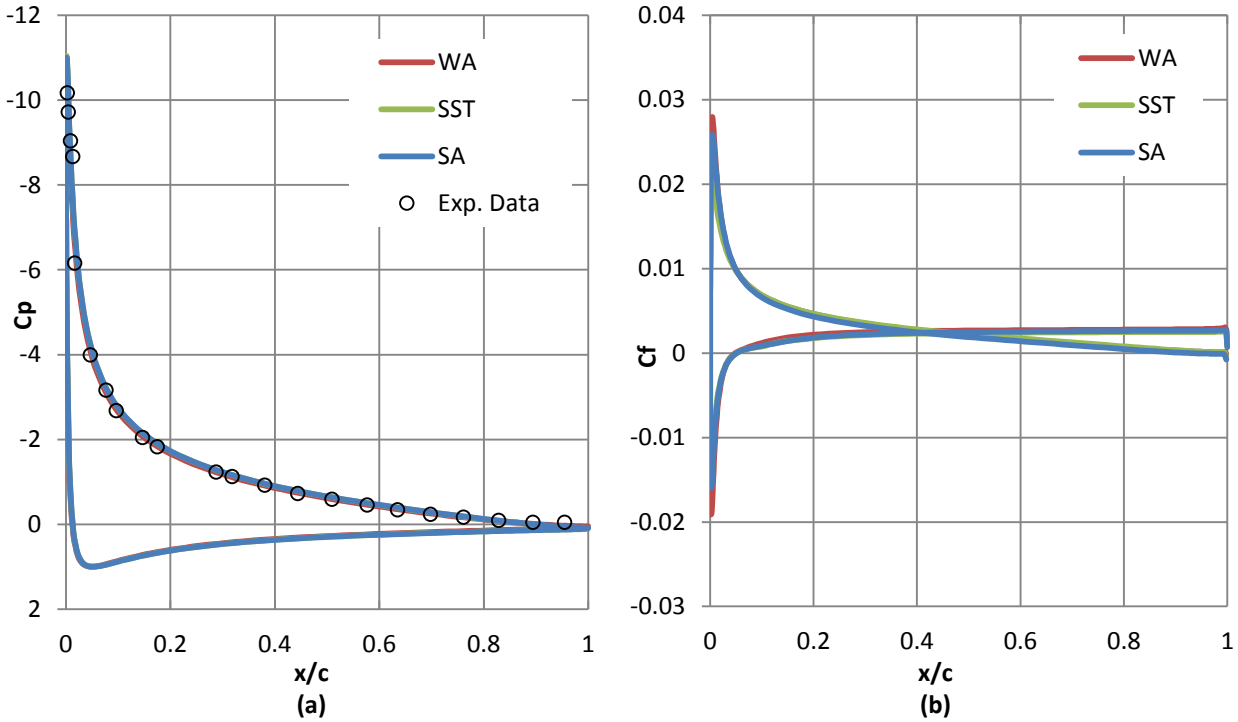


Figure 5.15: (a)Pressure coefficient and (b)skin friction coefficient comparisons for the NACA0012 at α=15°.

5.2.8 2D Backward Facing Step

In this section results for the TMR 2D Backward Facing Step are presented. The computations for this case correspond to the experiment investigated by Driver and Seegmiller [40]. The reference Mach number, $M = 0.128$, is taken at $x/H = -4$. This case has $Re_H = 36{,}000$ based on step height. The wall opposite the step has zero deflection and is treated as a viscous wall. The ratio between the channel height and the step is 9. Figure 5.16 shows the geometry of this case. A family of computational grids is provided by the TMR website. Results presented below were obtained with the second-finest grid. The computational grid is shown in Figure 5.17. The quantities of interest to compare are: the flow reattachment point, the pressure coefficient C_p and skin friction coefficient C_f of the lower wall, and normalized u-velocity profiles at $x/H = 1, 4, 6,$ and 10.

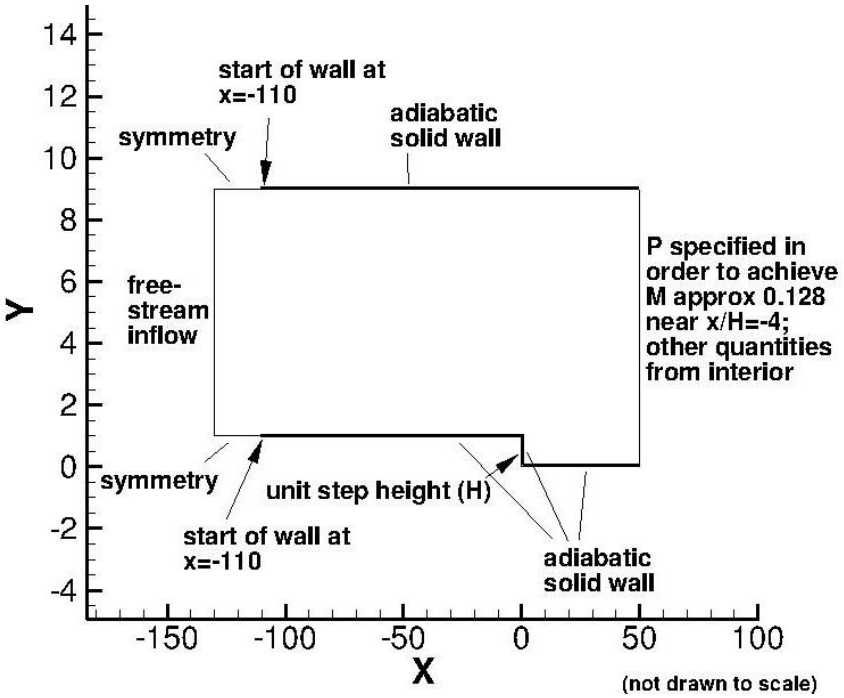


Figure 5.16: 2D backward facing step geometry and boundary conditions. [5]

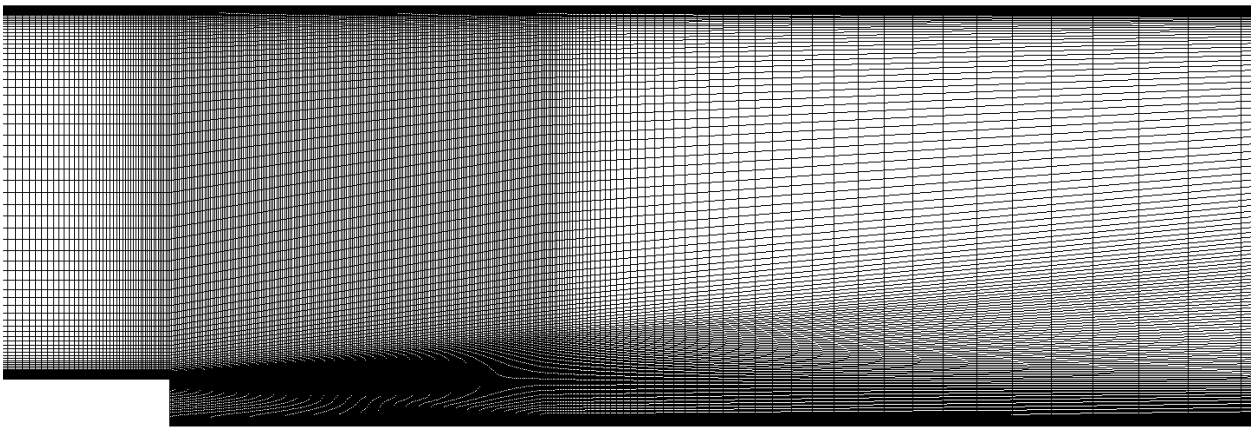


Figure 5.17: 2D backward facing step mesh.

The pressure coefficient and skin friction coefficients along the lower wall are given in Figure 5.18 and Figure 5.19 respectively. The predicted pressure coefficients of the WA and SST *k-ω* models are very similar and are in general agreement with the experimental data. The SA model pressure coefficient prediction is furthest from the experimental values, especially in the separation region and in the recovery region after reattachment. The reattachment length given in the experiment is *x/H = 6.26±0.1*. The WA and SA models predict reattachment slightly before this range at *5.95* and *6.08* respectively. The SST *k-ω* model overpredicts the length of the separation bubble with a reattachment point at *x/H = 6.60*. Skin friction through the separation region and downstream of the reattachment is best predicted by the WA model.

Velocity profiles sampled at four stations are shown in Figure 5.20(a)-(d). The results of the WA and SST *k-ω* model are very similar at *x/H = 1* and *4*. At *x/H = 6* the SST *k-ω* model has yet to reattach. Despite being the first model to reattach, the WA model shows delayed recovery evident at *x/H = 6* and *10*. The SA model recovers well and is able to predict the velocity profiles accurately downstream of reattachment.

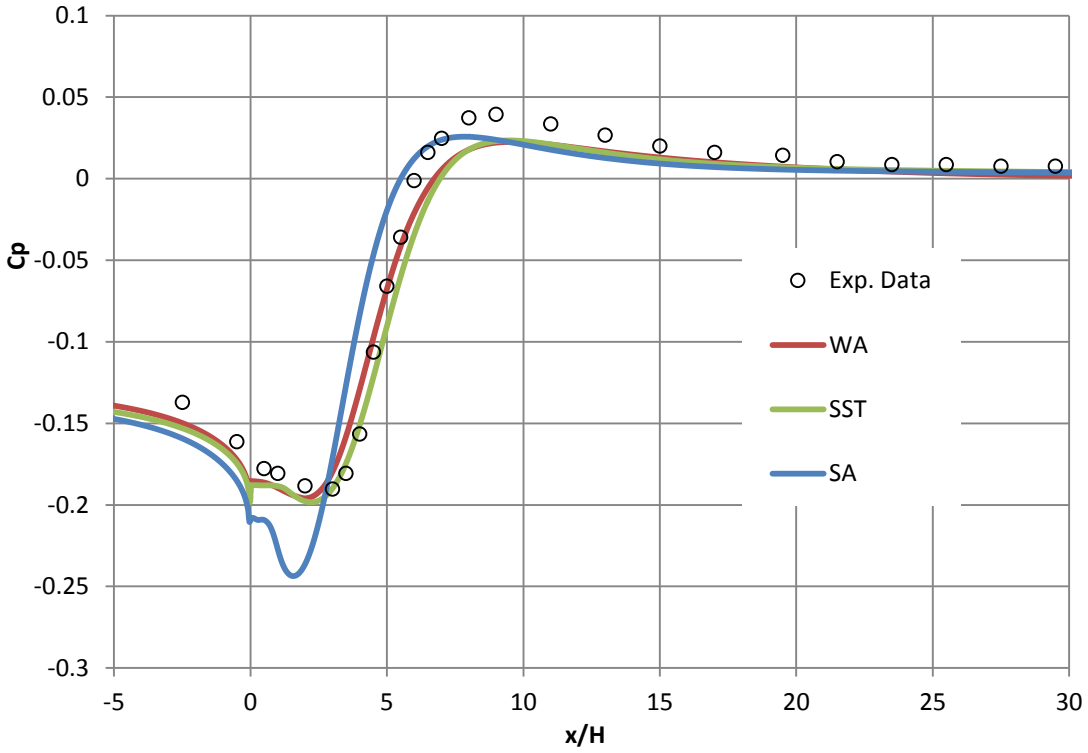


Figure 5.18: Pressure coefficient comparison for the 2D backward facing step.

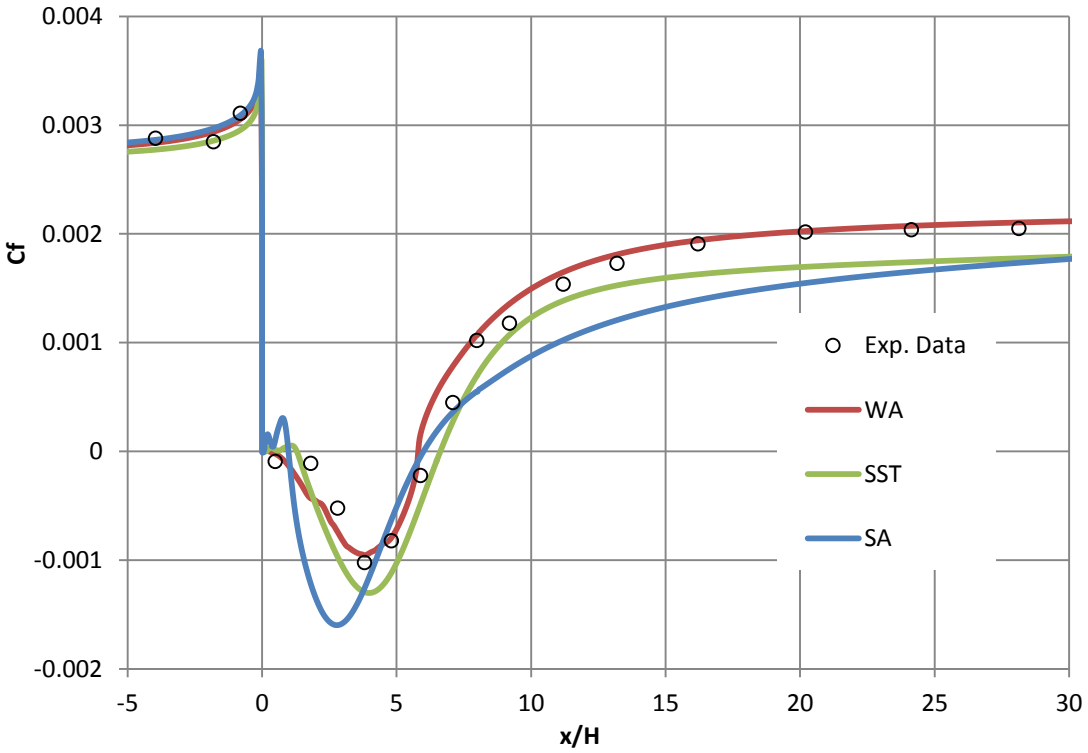


Figure 5.19: Skin friction coefficient comparison for the 2D backward facing step.

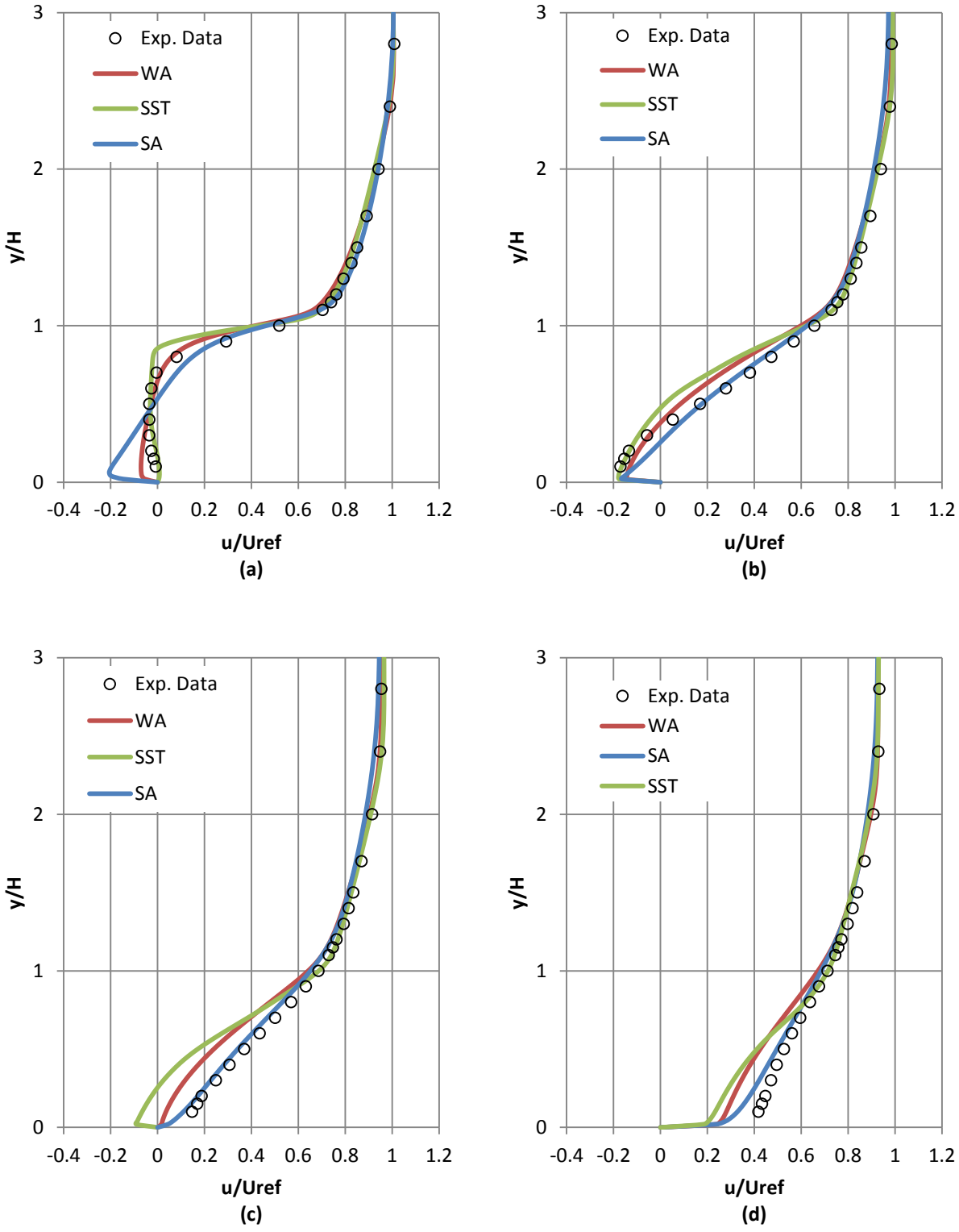


Figure 5.20: Velocity profile comparisons at x/H = (a)1, (b)4, (c)6, and (d)10.

5.2.9 2D Asymmetric Diffuser

In this section results for the NPARC 2D Asymmetric Diffuser are presented. The diffuser has an opening angle of $\alpha = 10°$ and an expansion ratio of *4.7* corresponding to the experiment conducted by Buice [41]. The inlet flow is a fully developed channel flow with Reynolds number Re_H = *20,000* based on the centerline velocity and channel height. An adverse pressure gradient causes the flow to separate. The computational grid was created using ANSYS ICEM and was uniformly refined until grid independence was achieved. The final grid contained ~175,000 cells and a $y^+ <$ *1.3* along the bottom wall. The quantities of interest to compare are: the surface pressure coefficient C_p and skin friction coefficient C_f along the bottom and top walls, the normalized u-velocity at *x/H = -3, 3, 6, 14, 17, 20, 24, 27, 30, 34, 40 47, 53, 60,* and *67*.

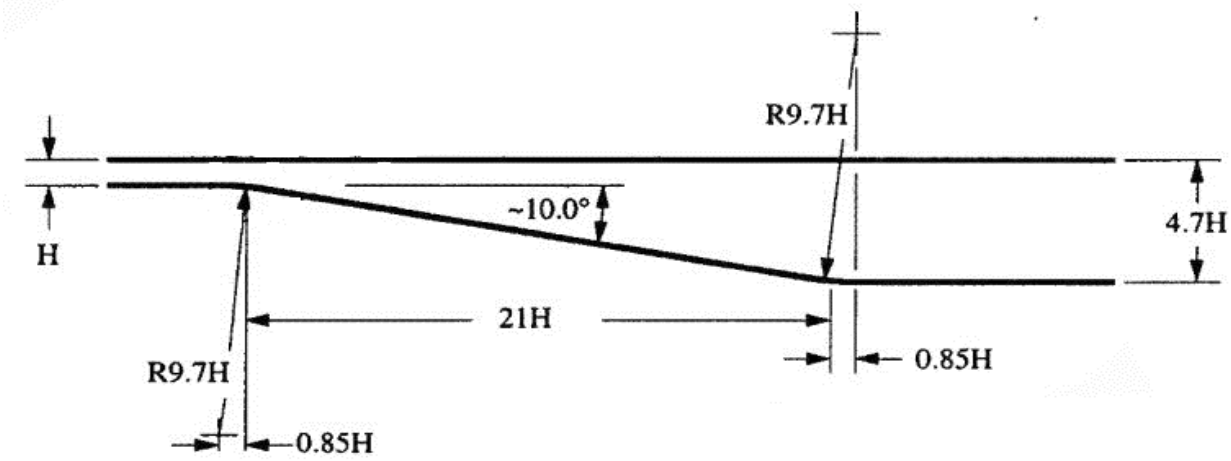


Figure 5.21: 2D asymmetric diffuser geometry.

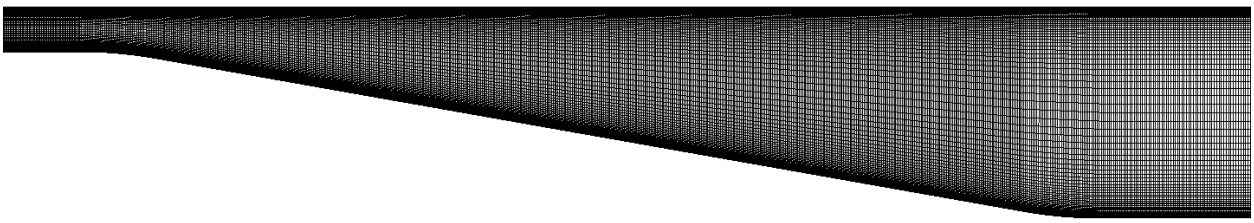


Figure 5.22: 2D asymmetric diffuser mesh.

The skin friction coefficient along the bottom wall is shown in Figure 5.23. The separation point in the experiment is *x/H = 6.6* and the reattachment point is at *x/H = 27.5*. Of the three models, the WA model most accurately predicts the separation location with a value of *x/H = 5.9*. The SA and SST *k-ω* models predict the separation location much too early at *x/H = 4.5* and *2.3* respectively. The WA model under predicts the reattachment point at x/H=24.9 while the SA and SST *k-ω* have an over prediction at *x/H ~ 30.9*. Figure 5.24 shows the skin friction coefficient along the top wall of the diffuser. The WA and SST *k-ω* most accurately predict the skin friction distribution. The SA model significantly under-predicts the skin friction coefficient in the region above the separation bubble. The pressure coefficient along the bottom and top wall are shown in Figure 5.25 and Figure 5.26 respectively. For both walls, the SST *k-ω* model is closest to the experimental values followed by the WA model and lastly by the SA model. All three models over-predict the pressure coefficient.

A comparison of the velocity profiles at several streamwise locations is shown in Figure 5.27 and Figure 5.28. All three models enter the diffuser with nearly the same developed channel flow evident at *x/H = -3*. As the flow enters the expansion, the SA model is first to slow while the WA and SST models maintain general agreement with the experimental data at *x/H = 3* and *6*. As the separation region begins to grow, the WA model most accurately matches the experimental data as seen at *x/H = 14* and *17*. The WA model reattaches sooner than the experiment and begins recovery at *x/H ~ 24*. The WA and SA model recover at about the same rate, while the SST *k-ω* model continues to overpredict the velocity in the upper region of the diffuser.

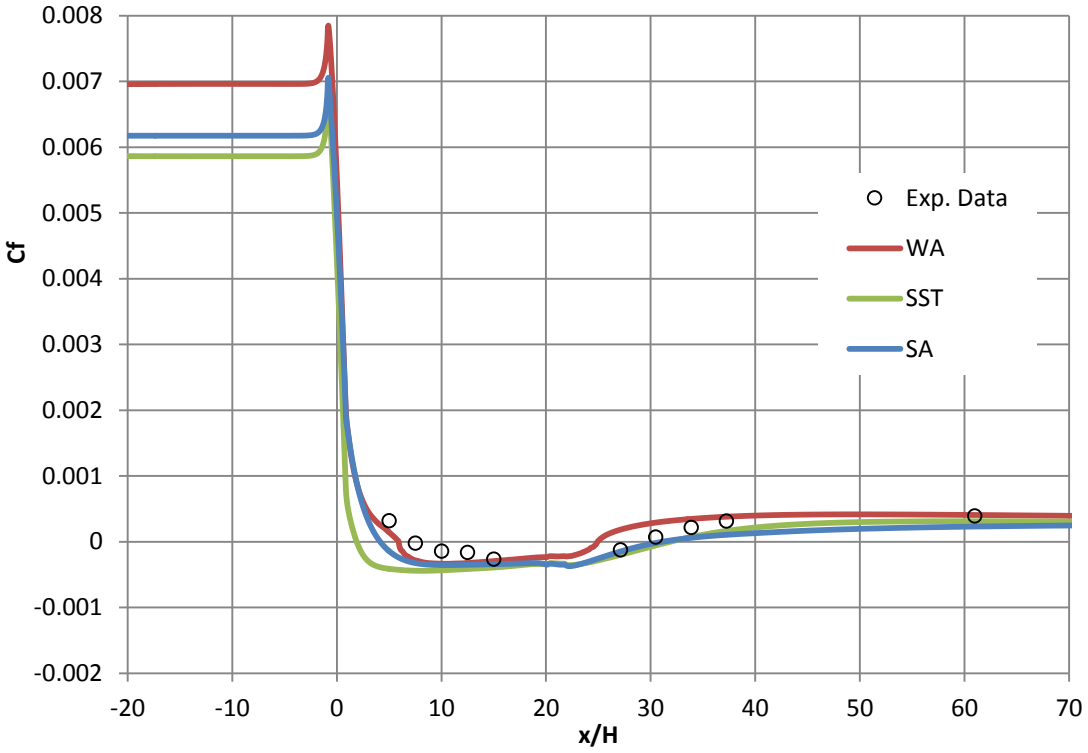


Figure 5.23: Comparisons of skin friction coefficient along the bottom diffuser wall.

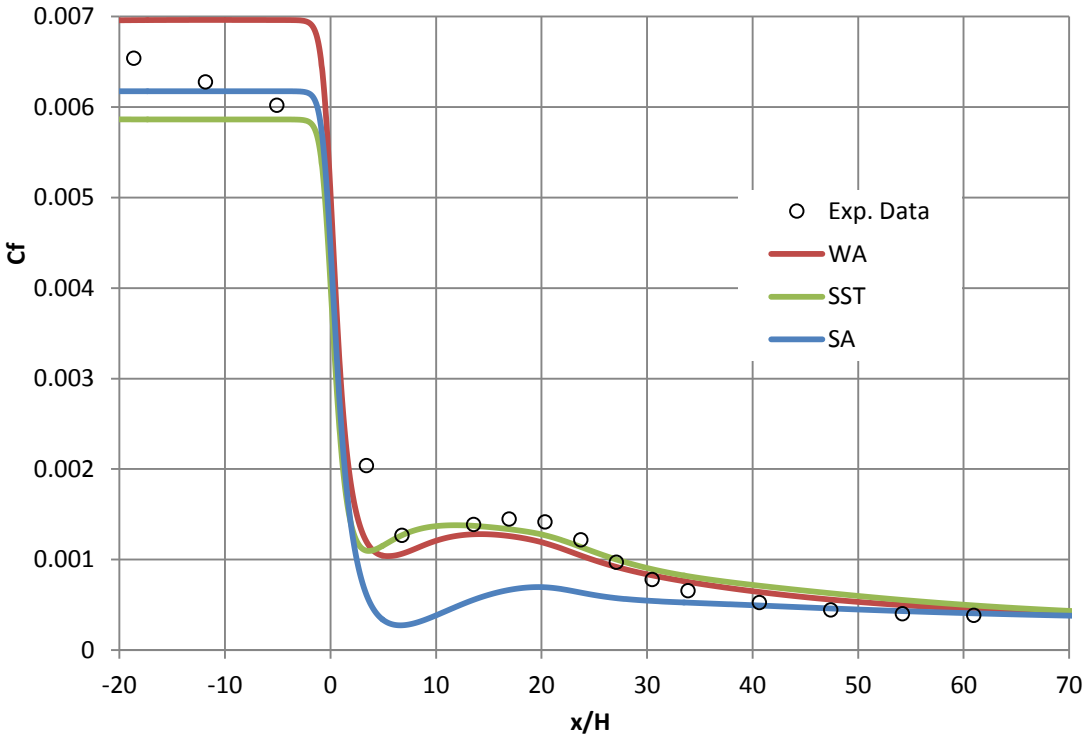


Figure 5.24: Comparisons of skin friction coefficient along the top diffuser wall.

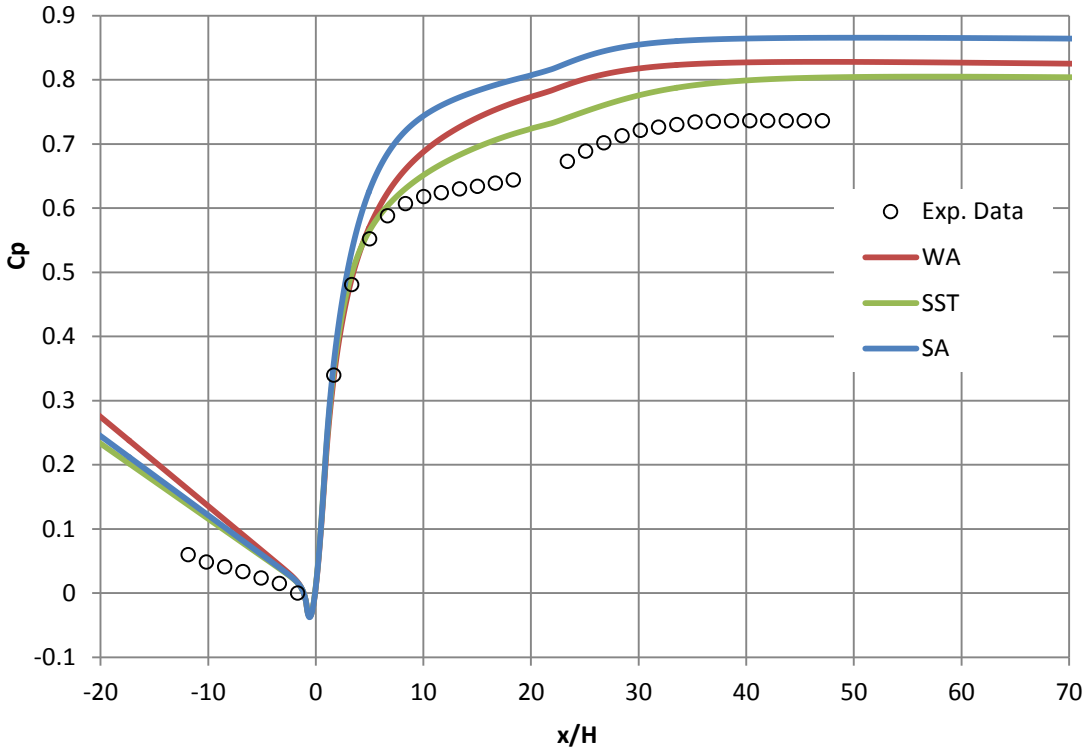


Figure 5.25: Comparison of the pressure coefficient along the bottom diffuser wall.

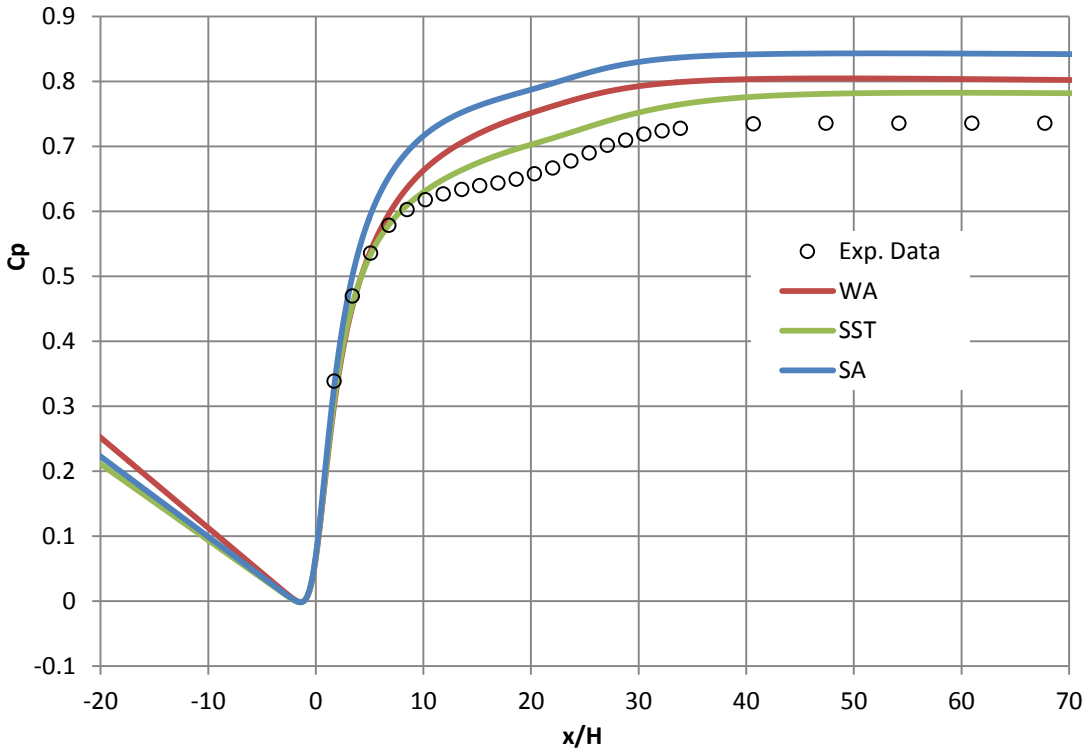


Figure 5.26: Comparison of the pressure coefficient along the top diffuser wall.

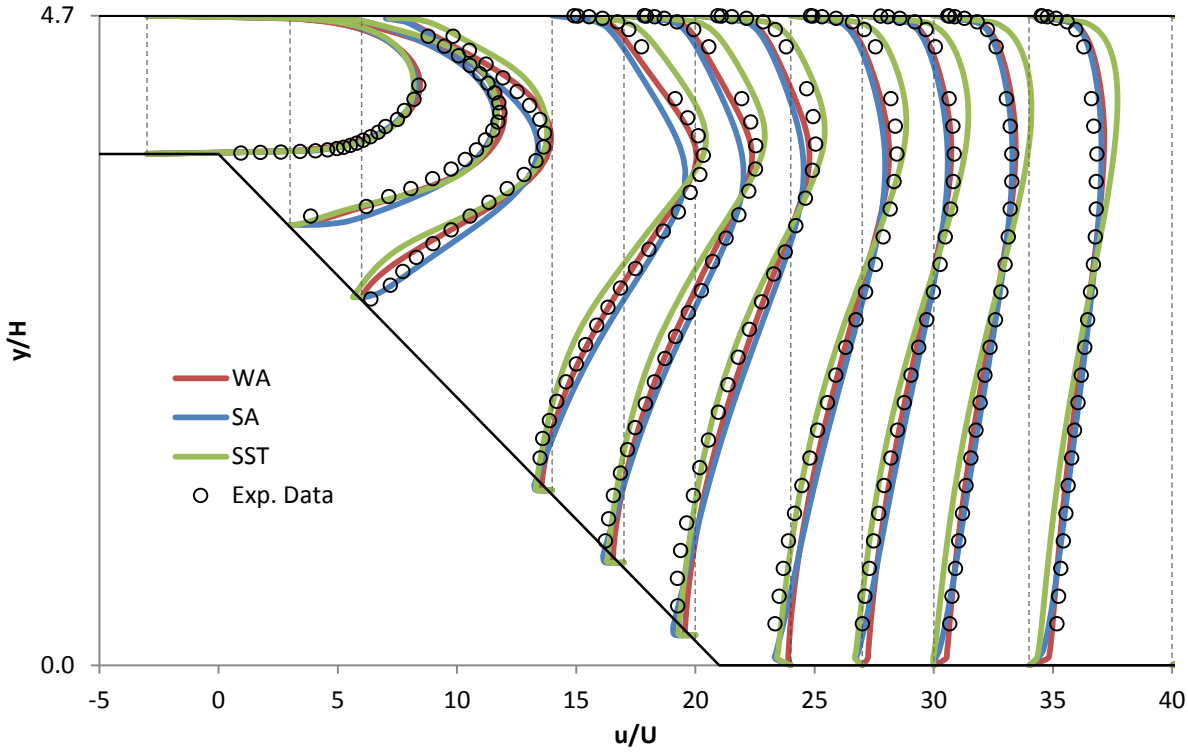


Figure 5.27: Velocity profile comparisons at stations x/H = -3 through 35.

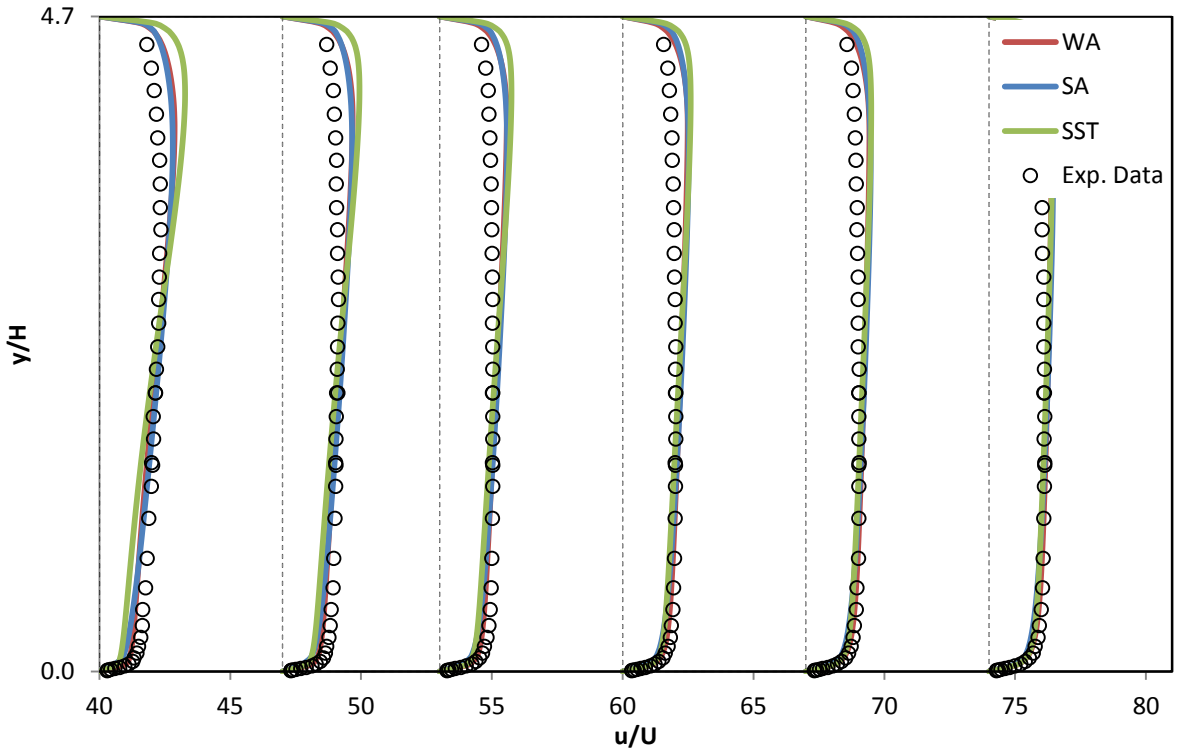


Figure 5.28: Velocity profile comparisons at stations x/H = 40 through 74.

5.2.10 2D NACA4412 Airfoil

In this section results for the TMR NACA4412 Airfoil with Trailing Edge Separation are presented. Results are compared with data obtained from the experiment of Coles and Wadcock [42,43]. At Mach number $M = 0.09$ and angle of attack $\alpha = 13.9°$, a trailing edge separation occurred on the upper surface of the airfoil. The Reynolds number based on chord length is $Re_C = 1.52x10^6$. The definition of the airfoil is slightly altered to achieve a sharp trailing edge with chord length $c = 1$. A family of computational grids is provided by the TMR website. Results presented below were obtained with the second-finest grid. The computational grid is shown in Figure 5.29. The quantities of interest to compare are: the pressure coefficient C_p, and normalized *u*-velocity *at* $x/c = 0.6753, 0.7308, 0.8418$, and 0.9528. It is important to note that the experimental velocities were non-dimensionalized by the velocity at a location about *1* chord below and behind the airfoil and not by the usual freestream value. The surface pressure coefficient from experiment was not corrected and therefore should only be viewed in a qualitative sense.

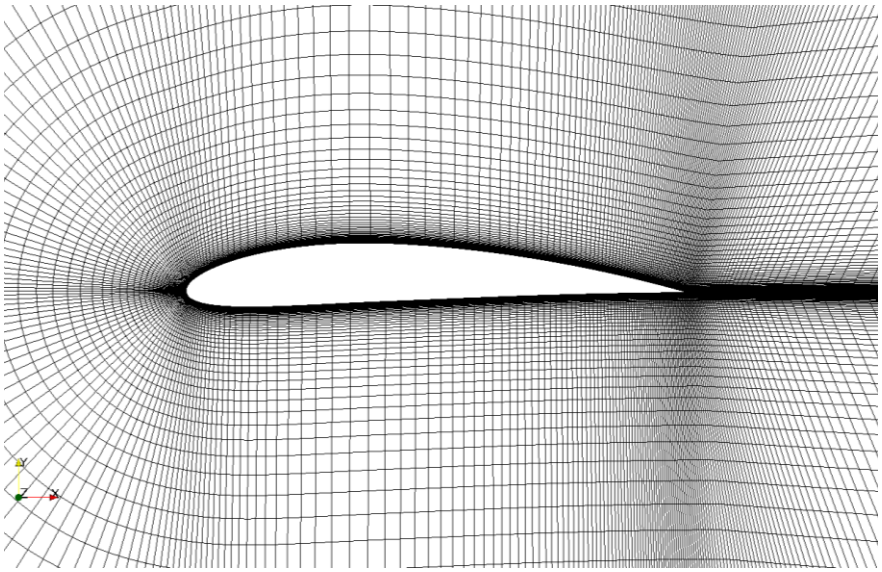


Figure 5.29: 2D NACA 4412 airfoil mesh.

A comparison of the surface pressure coefficients is shown in Figure 5.30 with a detailed view of the trailing edge separation region shown in Figure 5.31. The predictions of the WA and SST

models are nearly indistinguishable. While all three models are in general agreement with the experimental data over most the airfoil, none of them can accurately predict the pressure in the separation region.

The comparison of the velocity profiles through the separation region is a better indicator of the relative models performance. It can be seen in Figure 5.32(a)-(d) that leading up to the separation region, the SST model is in excellent agreement with the experimental data followed closely by the WA model. Both the WA and SST model correctly account for the non-equilibrium effects of the large adverse pressure gradient. As the flow separates the SST model most accurately predicts the bubble height followed closely by the WA model as evident in the upper regions of Figure 5.32(d)-(f). The SA model's poor velocity prediction early on along the airfoil only becomes worse moving downstream.

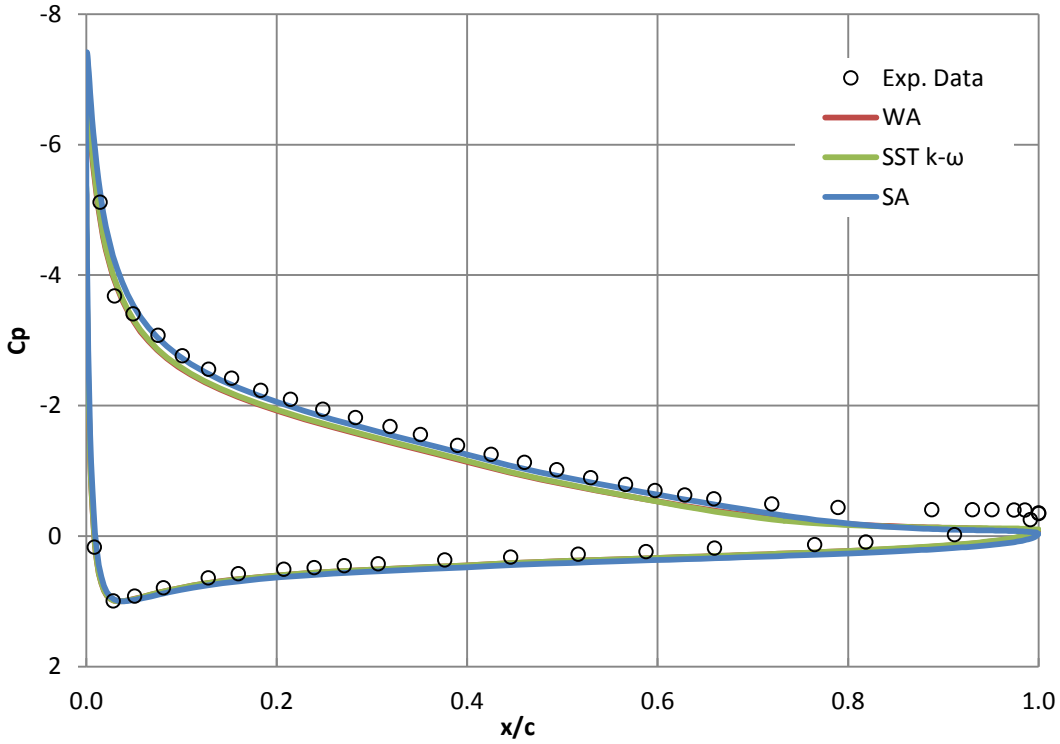


Figure 5.30: Comparison of the pressure coefficient distribution over the NACA4412 airfoil.

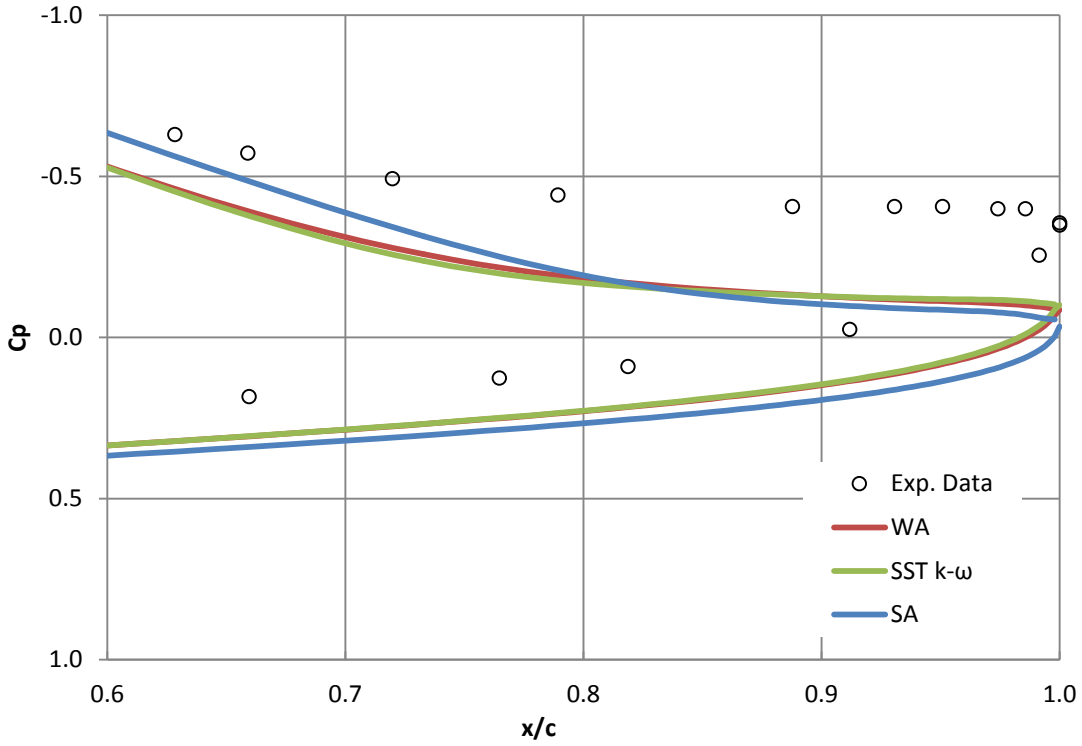


Figure 5.31: Pressure coefficient comparison in the trailing edge region of the NACA4412 airfoil.

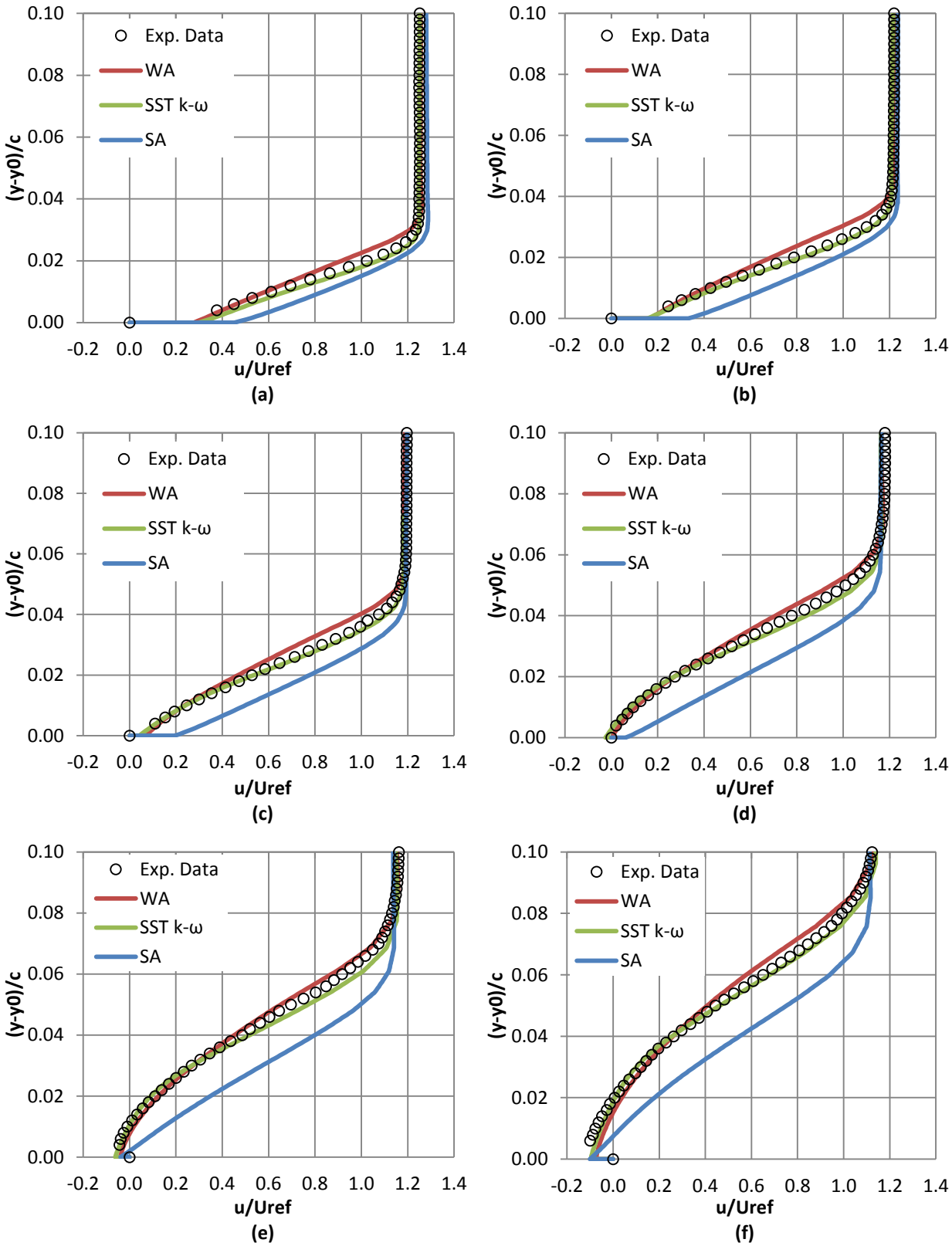


Figure 5.32: Normalized velocity profile comparisons at x/c = (a)0.6753, (b)0.7308, (c)0.7863, (d)0.8418, (e)0.8973, and (f)0.9528 for the NACA4412 airfoil.

5.2.11 2D Wall-Mounted Hump

In this section results for the TMR 2D NASA Wall-Mounted Hump Separated Flow are presented. The focus of this case is to assess the ability of turbulence models to predict separation, reattachment, and boundary layer recovery. The geometry designed by Seifert and Pack [44] is run under the test conditions given in Greenblatt et al. [45]. The Reynolds number based on the hump length is $Re_C = 936{,}000$ and the Mach number is $M = 0.1$. A family of computational grids is provided by the TMR website. Results presented below were obtained with the second-finest grid. The flow control plenum present in the experiment is removed from the grid. The upper wall is contoured to approximately account for side-plate blockage present in the original geometry. The computational grid is shown in Figure 5.33. The inlet boundary condition is prescribed to match the experimental values at $x/c = -2.14$ as shown in Figure 5.34. The quantities of interest to compare are: the flow separation point, the flow reattachment point, the pressure coefficient C_p and skin friction coefficient C_f along the hump and lower wall, and normalized u-velocity profiles at $x/c = -2.14, 1.0, 1.1, 1.2$, and 1.3.

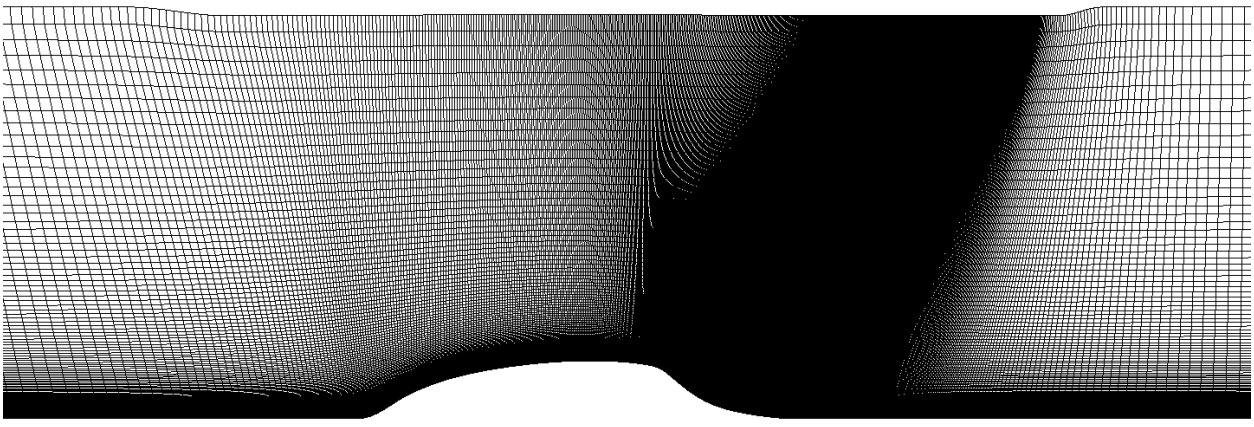


Figure 5.33: 2D wall mounted hump mesh.

The skin friction coefficient along the hump wall is shown in Figure 5.35. All three models show separation at $x/c = 0.665$ which is in good agreement with the experimental value. The

reattachment point in the experiment was found to be in the range of *x/c = 1.07* to *1.13*. All three models predict a reattachment point in the range of *x/c = 1.26-1.29*, greatly over predicting the size of the separation region. Results for the mean pressure coefficient along the hump wall are presented in Figure 5.36. Leading up to the hump and over the top of the hump, all three models are in agreement with the experimental data. The models predict higher pressure levels in the separation region than measured experimentally and the stretching of the separation region is also evident.

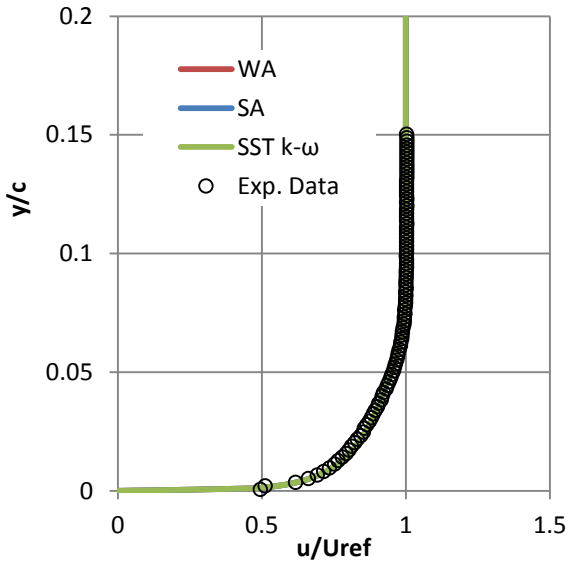


Figure 5.34: Inlet velocity profile for the NASA 2D hump case.

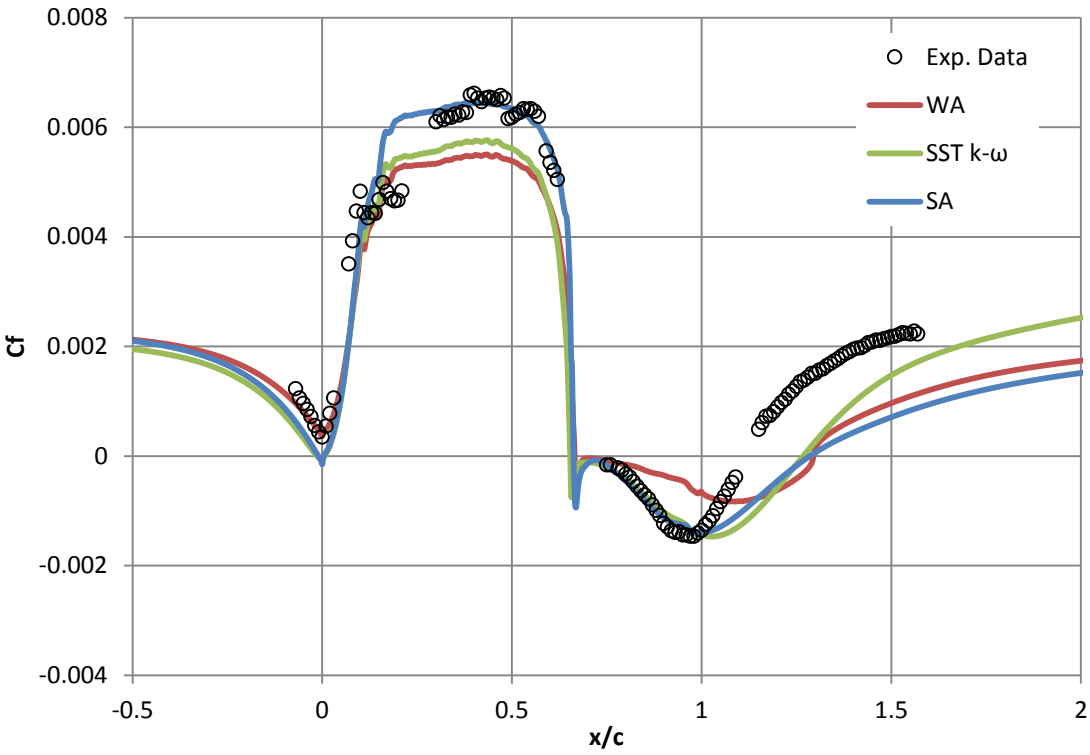


Figure 5.35: Skin friction coefficient comparison along the NASA 2D hump.

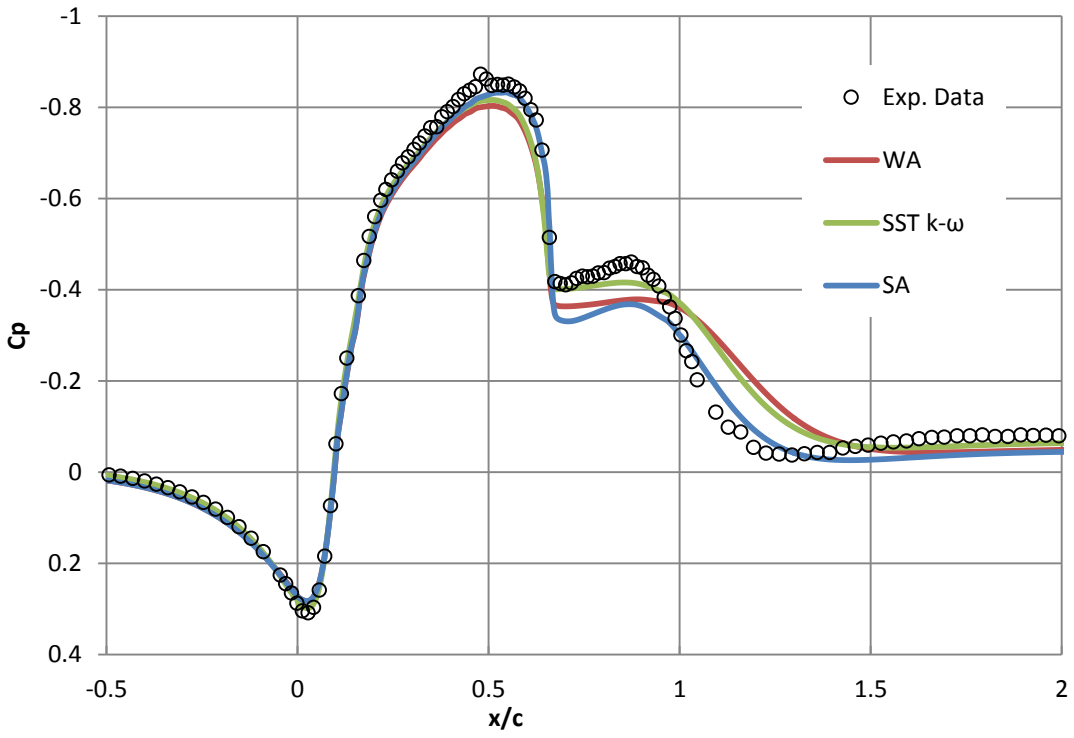


Figure 5.36: Pressure coefficient comparison along the NASA 2D hump.

Velocity profiles at four stations through the separation and recovery region are shown in Figure 5.37(a)-(d). The behavior of the WA model is very similar to that of the SST model. The large error in the reattachment point prediction is also evident in the velocity profile comparisons. All the models fail to accurately predict the velocity profiles.

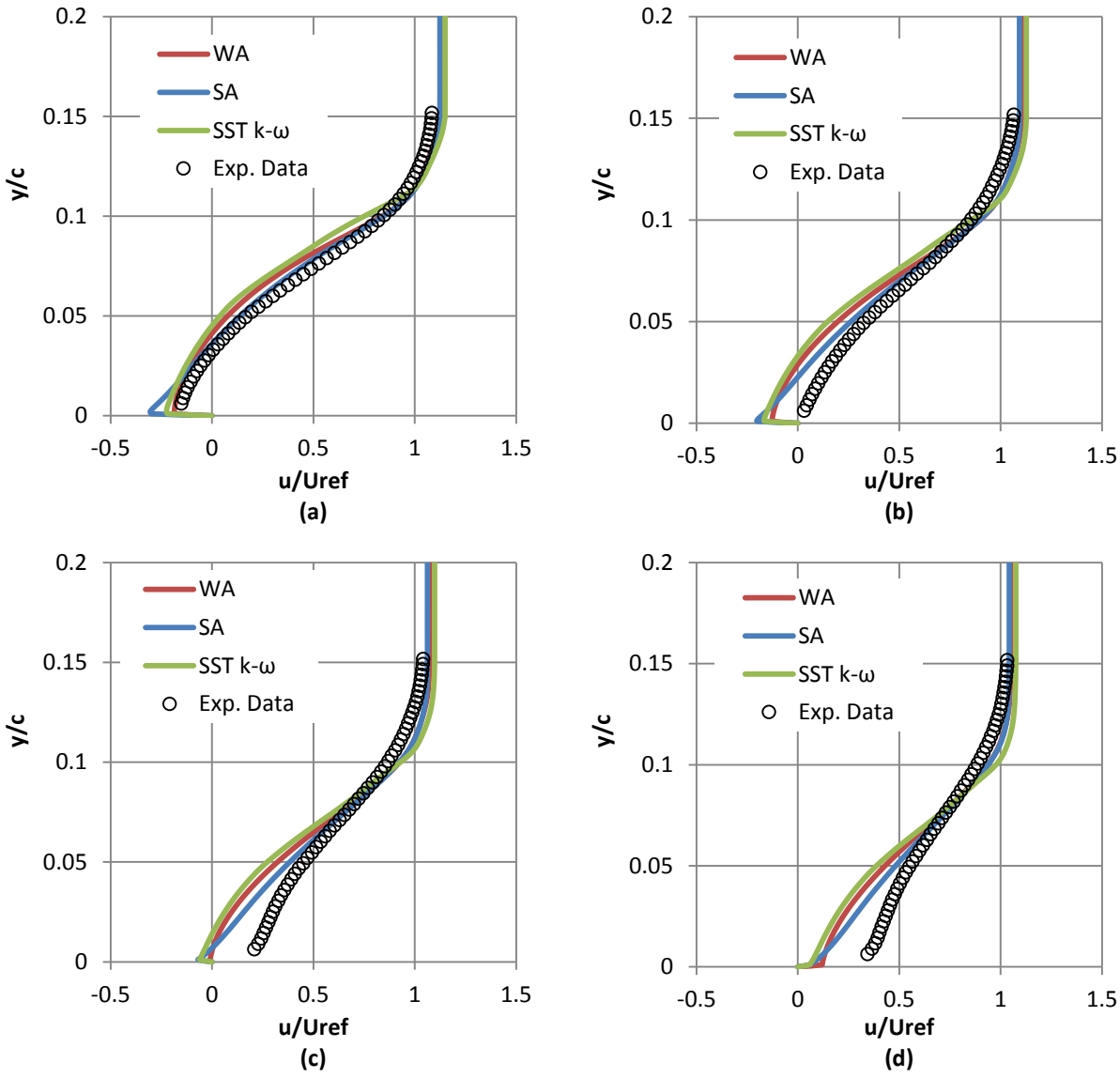


Figure 5.37: Normalized velocity profile comparisons at x/c=(a)1.0, (b)1.1, (c)1.2, and (d)1.3.

5.2.12 Axisymmetric Separated Boundary Layer

In this section results for the TMR Axisymmetric Separated Boundary Layer are presented. The geometry and flow conditions for this case are taken from the experiment by Driver [47]. In the

experiment an adverse pressure gradient was imposed so that the boundary layer separated and reattached. A streamline shape far from the boundary layer was provided in the experiment and was used as an inviscid upper boundary condition for the computation. The geometry and boundary conditions are shown in Figure 5.38. The reference Mach number is $M = 0.08812$ and Reynolds number $Re = 2x10^6$. A family of computational grids is provided by the TMR website. Results presented below were achieved with the second-finest grid. The quantities of interest to compare are: the separation and reattachment points, the surface skin friction coefficient C_f, the surface pressure coefficient C_p, and normalized u-velocity profiles at $x=-0.3302$, 0.0508, 0.1524, and 0.3048.

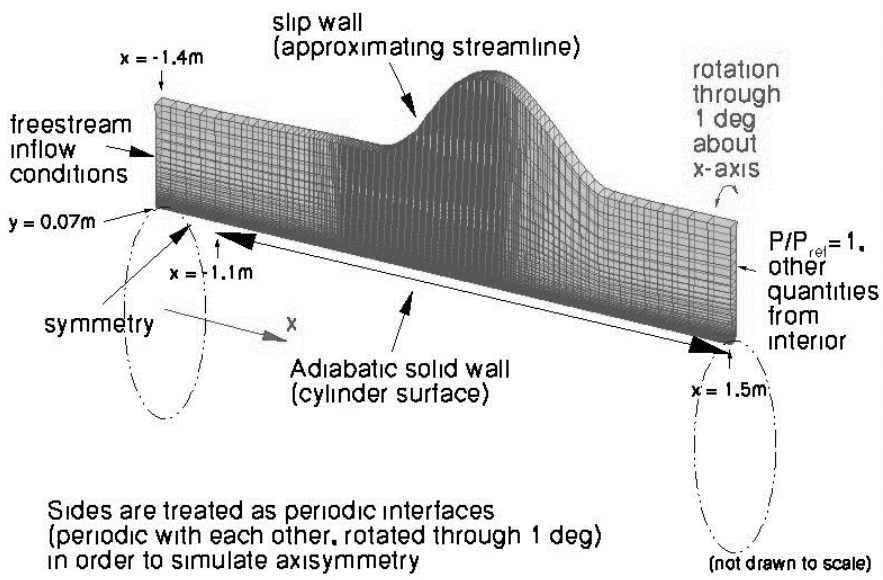


Figure 5.38: Axisymmetric separated boundary layer geometry and boundary conditions [5].

The mean surface friction coefficient over the cylinder wall is shown in Figure 5.40. The three models predict nearly the same separation point. The WA model vastly under predicts the separation bubble size. The SST and SA models over predict the bubble length. A comparison of the pressure coefficients is shown in Figure 5.39. The WA and SST models are in excellent agreement with the experimental data.

All models approach the separation with nearly the same developed boundary layer as shown in Figure 5.41(a). Continuing into the separation region the WA and SST models have very similar predicted profiles, except very near the wall where the SST model correctly predicts a small separation region.

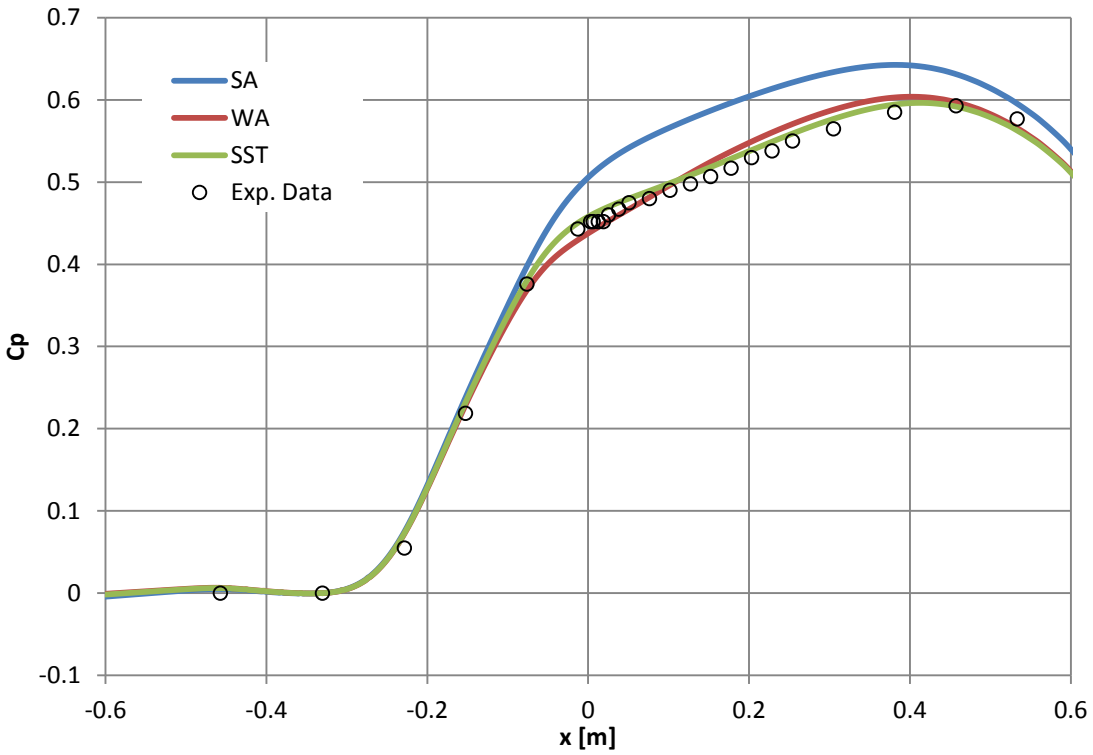


Figure 5.39: Comparison of the pressure coefficient for the axisymmetric separated boundary layer flow.

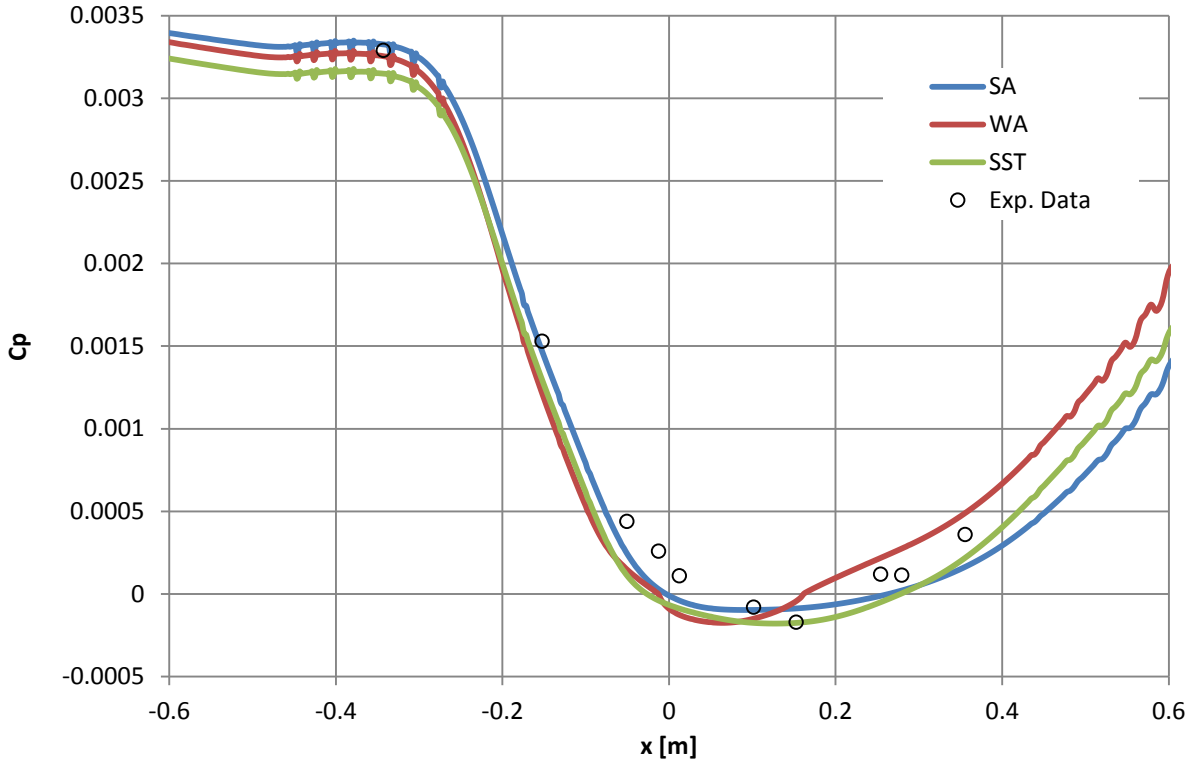


Figure 5.40: Comparison of the skin friction coefficient for the axisymmetric separated boundary layer flow.

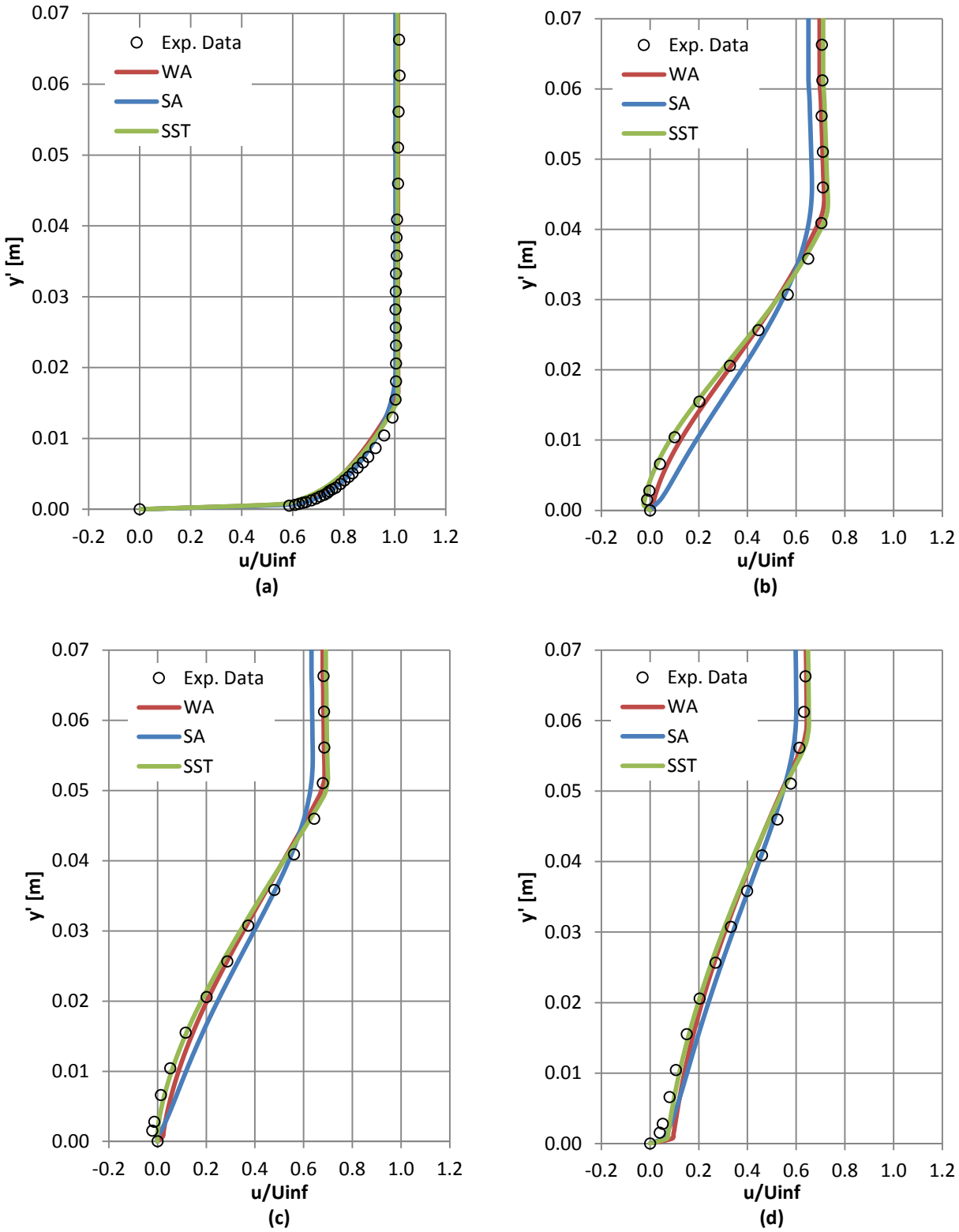


Figure 5.41: Comparison of the normalized velocity profiles for the axisymmetric separated boundary layer at x=(a)-0.3302, (b)0.0508, (c)0.1524, and (d)0.3048 meters.

5.3 Transonic Flows

5.3.1 2D RAE2822Airfoil

In this section results for the NPARC 2D Transonic RAE2822 Airfoil are presented. Results are compared with data obtained from the experiment of Cook et al. [50] At Mach number $M = 0.725$ and angle of attack $\alpha = 2.79°$ a shockwave develops over the suction side of the airfoil. The Reynolds number based on chord length is $Re_C = 6.2x10^6$. A grid was created for this case with ANSYS ICEM and uniformly refined until grid independence was achieved. The final mesh had *165,000* cells and a maximum $y^+ < 0.7$ across the airfoil.

The comparison of the pressure coefficients is given in Figure 5.42. As can be seen, all the models show very good agreement with the experimental data over the majority of the airfoil surface. The only discernable difference among the models is the prediction of the shock location. The SA model exhibits a shift in downstream of the experimental data. The SST and WA models predict nearly identical shock positions slightly upstream of the experimental position.

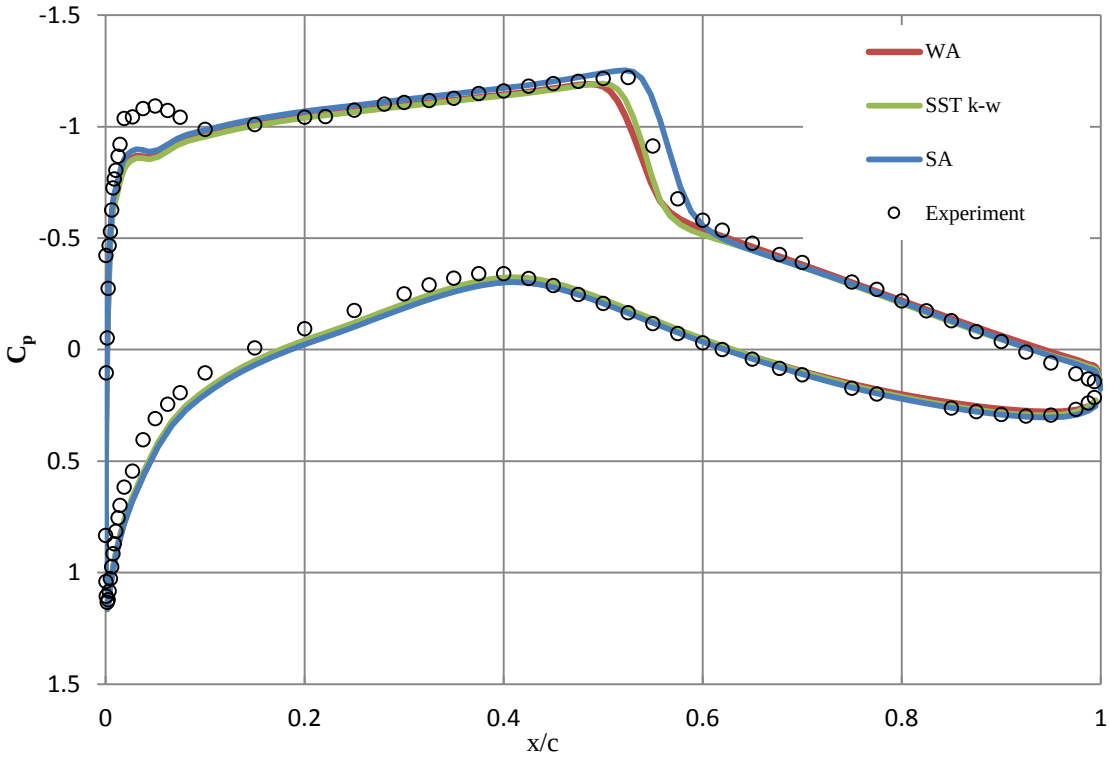


Figure 5.42: Comparison of the surface pressure coefficient for the RAE2822 airfoil.

5.3.2 Axisymmetric Bump

In this section results for the TMR Axisymmetric Transonic Bump are presented. The computations of this case correspond to the experiment of Bachalo and Johnson [51]. In the experiment a shockwave developed near the trailing edge of a bump protruding from a cylinder. The shock and adverse pressure gradient caused flow separation with eventual reattachment. The reference Mach number is $M = 0.875$ and Reynolds number is $Re = 2.763x10^6$. The geometry is shown in Figure 5.43. A family of computational grids is provided by the TMR website. Results presented below were achieved with the second-finest grid. The quantities of interest to compare are: the separation and reattachment points, the surface skin friction coefficient C_f, the surface pressure coefficient C_p, and normalized u-velocity profiles at $x/c = -0.250, 0.688, 0.813, 0.938, 1.125$, and 1.25.

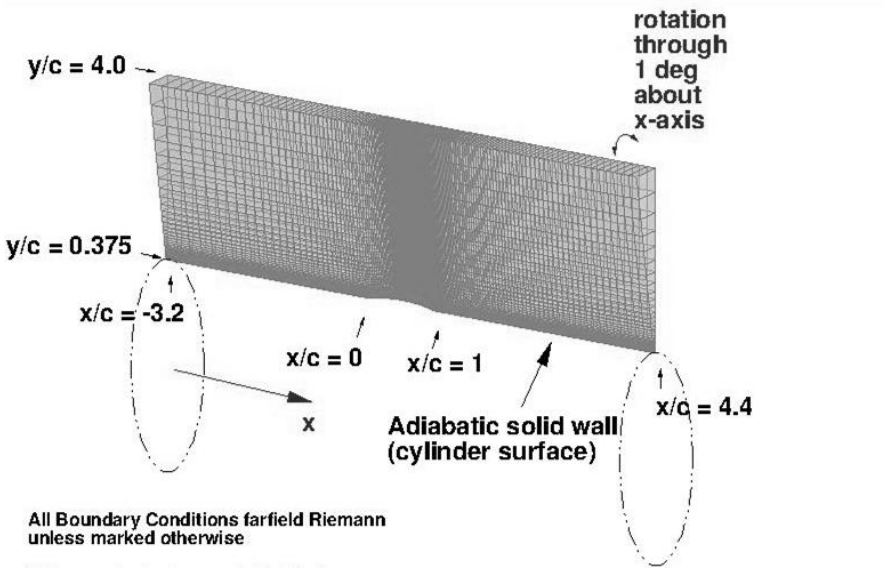


Figure 5.43: Computational domain of the axisymmetric transonic bump [5].

Figure 5.44 shows the comparison of the pressure coefficients along the surface of the bump. The SA model predicts a delay of the shock position and overpredicts the pressure after the shock and into the separation region. The SST k-ω model most accurately predicts the shock location followed closely by the WA model. The predicted pressure coefficients of the SST k-ω and WA models are comparable and are in good agreement with the experimental data after the shock and throughout the separation region. Oil-flow visualizations have determined that the separation and reattachment occur at approximately x/c = 0.7 and x/c = 1.1 respectively. The predicted separation and reattachment points determined from the skin friction coefficient are shown in Table 5.2. It can be seen that the SA model most accurately predicts the separation point while the WA model most accurately predicts the reattachment point. The shock position and separation bubble size have a strong coupling. It is noteworthy that even though the WA model predicts an earlier shock than SST k-ω and SA models, it has the latest separation. This is in closer agreement with the behavior observed in the experiment where the separation occurred slightly downstream of the shock.

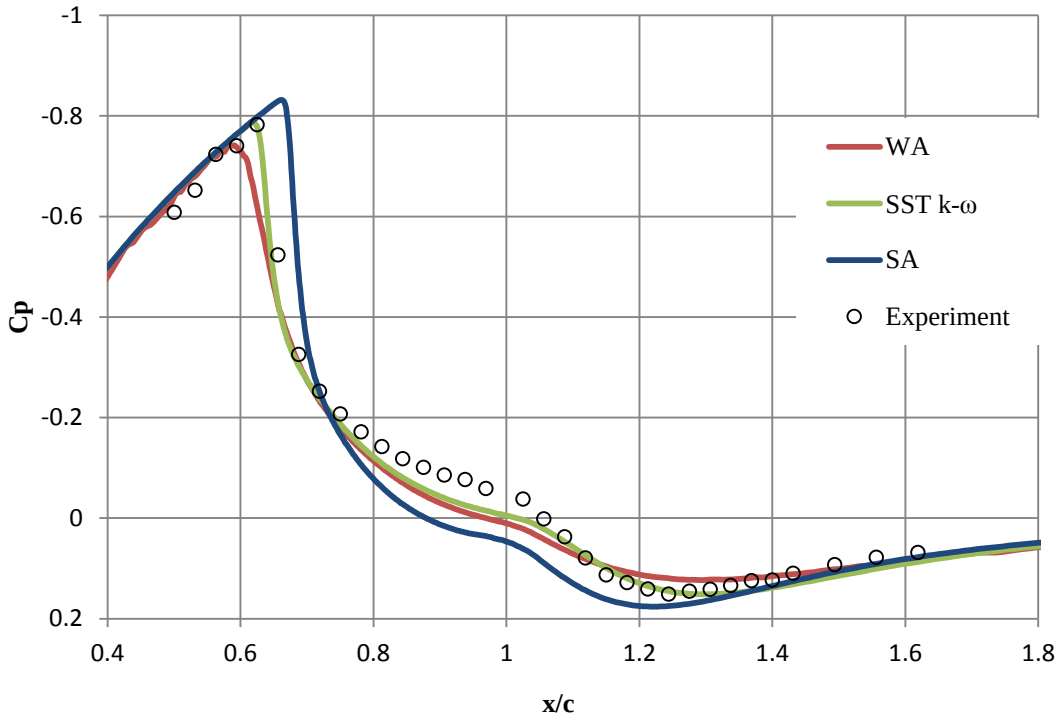


Figure 5.44: Comparison of the surface pressure coefficients for the axisymmetric transonic bump.

Table 5.2: Comparison of the flow separation and reattachment points for the axisymmetric transonic bump.

	Experiment	WA	% Error	SST *k-ω*	% Error	SA	% Error
Separation	0.7	0.75	7.14	0.65	-7.14	0.69	-1.43
Reattachment	1.1	1.109	-0.82	1.16	5.45	1.16	5.45

Comparisons of the velocity profiles at specific measured positions are shown in Figure 5.45. The wall normal coordinate y'/c is defined such that $y'/c = 0$ on the geometry surface. The first velocity measurements were taken just before the flow reached the bump at $x/c = -0.25$. It can be seen from Figure 5.45(a) that all three models develop a boundary layer indistinguishable from each other and give results in excellent agreement with the experimental data. Figure 5.45(b) shows the velocity profiles over the bump. It can be seen that this location intersects the predicted shockwaves from the WA and SST *k-ω* models. Figure 5.45(c-e) compare the velocity profiles through the separation region and just after reattachment. Very close to the wall, the WA model

most accurately predicts the velocity profile. This is most evident in Figure 5.45(e), where the WA model is the only model that predicts the reattachment of the flow. After $y'/c \sim 0.15$ the WA model begins to underpredict the velocity and the SST k-ω and SA models are in better agreement with the experimental data. Downstream of the reattachment point and into the recovery region, the SA model is in best agreement with the experimental data.

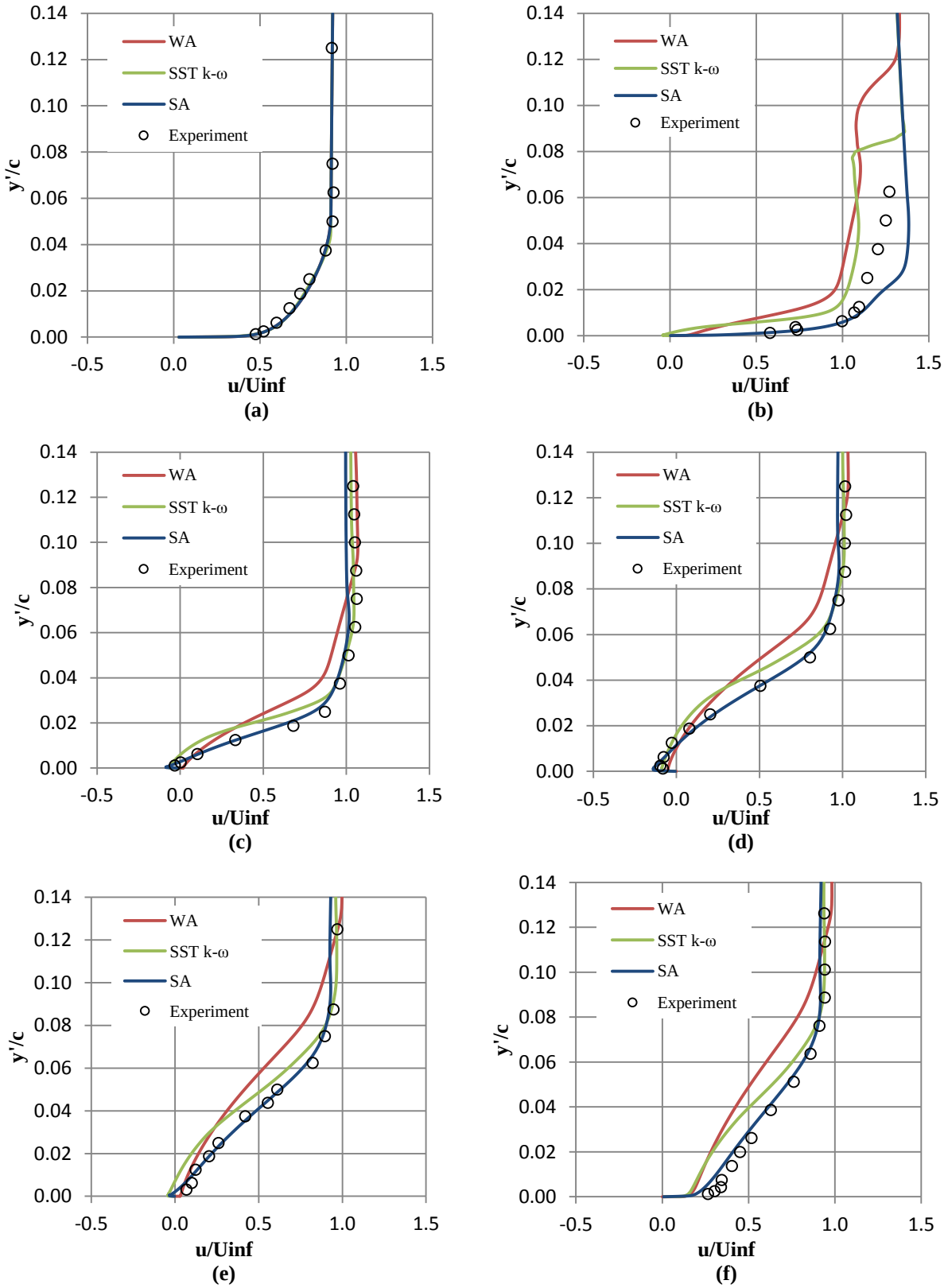


Figure 5.45: Comparison of the mean velocity profiles at x/c =(a)-0.25, (b)0.688, (c)0.813, (d)0.938, (e)1.125 and (f)1.25.

5.4 Supersonic Flows

5.4.1 2D Flat Plate

In this section results for the TMR 2D Zero Pressure Gradient High Mach Number Flat Plate are presented. High speed flow over a flat plate at four flow conditions was calculated. The Reynolds number based on unit length was $Re_L=15x10^6$. The four flow conditions are:

- $M_{inf}=2.0, T_w/T_{inf}=1.712$
- $M_{inf}=5.0, T_w/T_{inf}=1.090$
- $M_{inf}=5.0, T_w/T_{inf}=2.725$
- $M_{inf}=5.0, T_w/T_{inf}=5.450$

A family of computational grids is provided by the TMR website. Results presented below were achieved with the finest grid. The computational grid is shown in Figure 5.46.

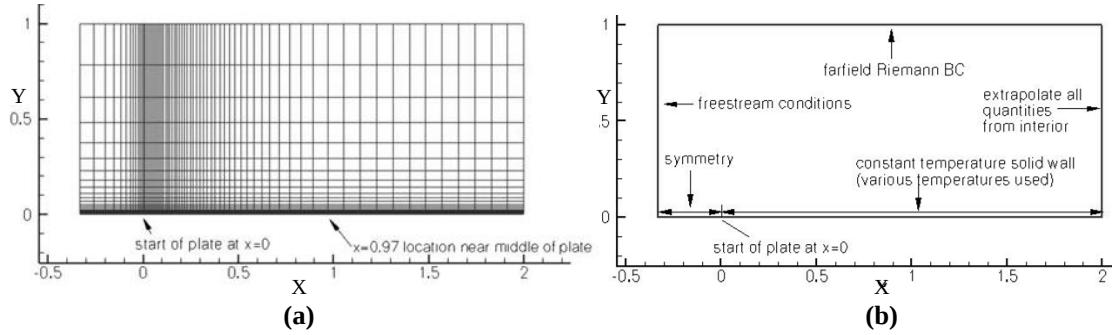


Figure 5.46: Sonic flat plate (a) grid and (b) boundary conditions [5].

The quantities of interest to compare are: the wall skin friction coefficient C_f, and the nondimensional velocity at $Re_\theta = 10{,}000$. Results are compared to empirical incompressible correlations by use of the van Driest transformation [52] of the Karman-Schoenherr [53] relation. It should be noted that the correlations are imperfect.

Skin friction results for the flat plate simulations for the four flow conditions are shown in Figure 5.47 and Figure 5.48. For all the cases the three models are in general agreement with the theoretical value. The largest spread in predictions is seen in the $M_{inf}=5.0$, $T_w/T_{inf}=1.090$ case shown in Figure 5.48. For this case, the SA model notably predicts a much lower skin friction coefficient. The results of the WA model are comparable to that of the SST k-ω model.

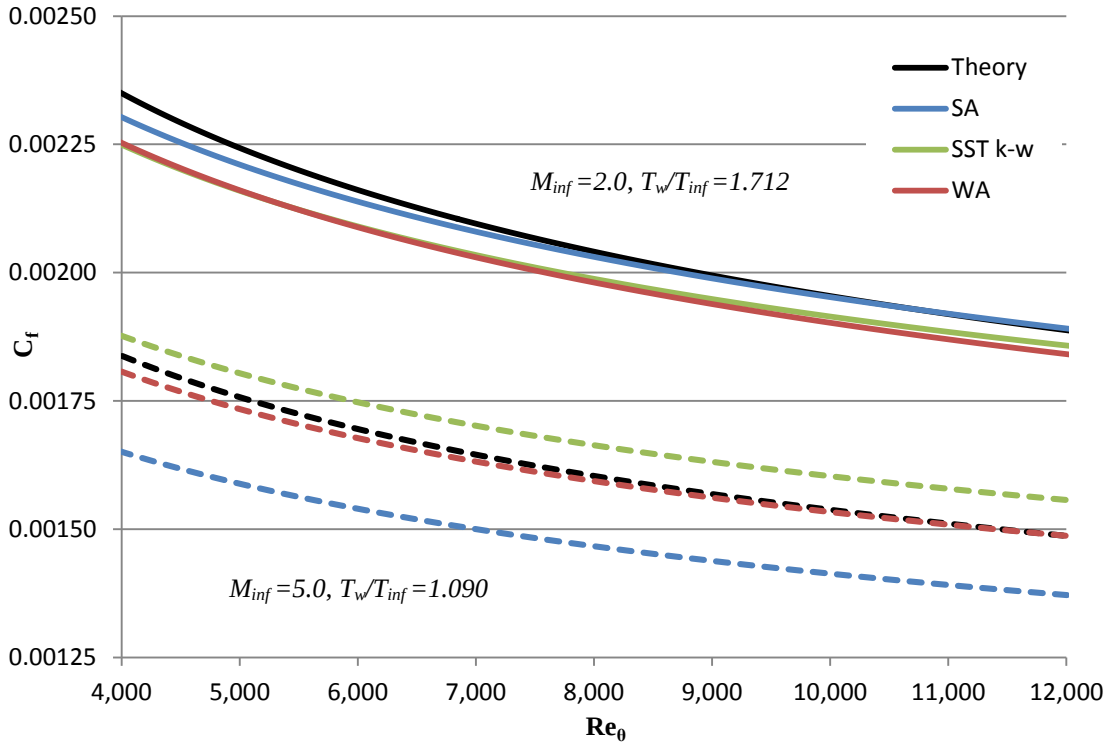


Figure 5.47: Skin friction coefficient comparisons for the $M_{inf}=2.0$, $T_w/T_{inf}=1.712$ and $M_{inf}=5.0$, $T_w/T_{inf}=1.090$ case.

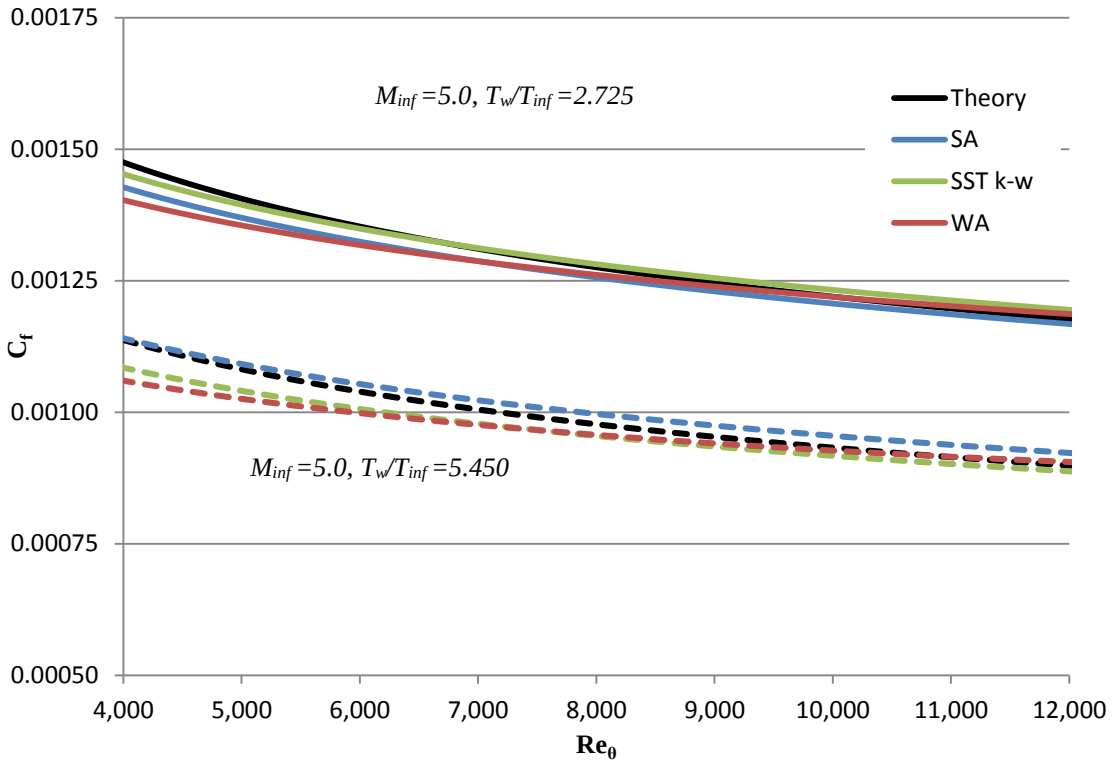


Figure 5.48: Skin friction coefficient comparisons for the M_{inf} =2.0, T_w/T_{inf} =2.725 and M_{inf} =5.0, T_w/T_{inf} =5.450 case.

5.4.2 2D Slot Nozzle Ejector

In this section results for the NPARC 2D Slot Nozzle Ejector are presented. The nozzle configuration evaluated experimentally by Gilbert and Hill [54] is considered. The geometry is shown in Figure 5.49. High speed flow ejected from the nozzle entrains the ambient air into the mixing chamber as a means to create additional thrust. A grid was created for this case with ANSYS ICEM and uniformly refined until grid independence was achieved. The final mesh had *205,000* cells and a maximum y^+=*1.5* along the mixing section wall. The quantities of interest to compare are: the surface static pressure along the mixing section wall, and velocity profiles at *x = 3", 7"* and *10.5"* from the nozzle outlet.

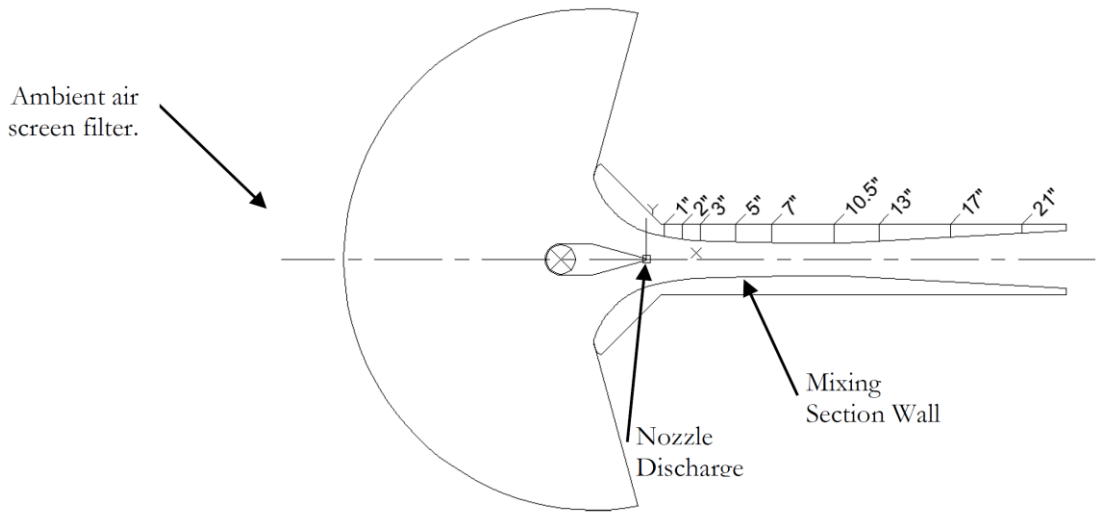


Figure 5.49: Experimental apparatus cross-section and measurement locations.

Computational results for the mixing section wall static pressure are compared to the experimental data in Figure 5.50. It can be seen that the WA and SST *k-ω* models are in best agreement with the experimental data. The SA model has the correct trend but greatly underpredicts the pressure magnitude.

Figure 5.51 compares the computed and measured velocity profiles at three 3", 7", and 10.5" locations downstream from the nozzle outlet. It can be seen from Figure 5.51(a) that at location 3" away from the nozzle, all the models overpredict the velocity of the jet core. The SST *k-ω* and WA models are in agreement with the experimental data away from the jet. The prediction of the centerline velocity of the SST *k-ω* and WA models approaches the experimental values further from the nozzle discharge as can be seen from Figure 5.51(b,c). The SA model continues to greatly overpredict the velocity even at location 10.5" away from the nozzle discharge.

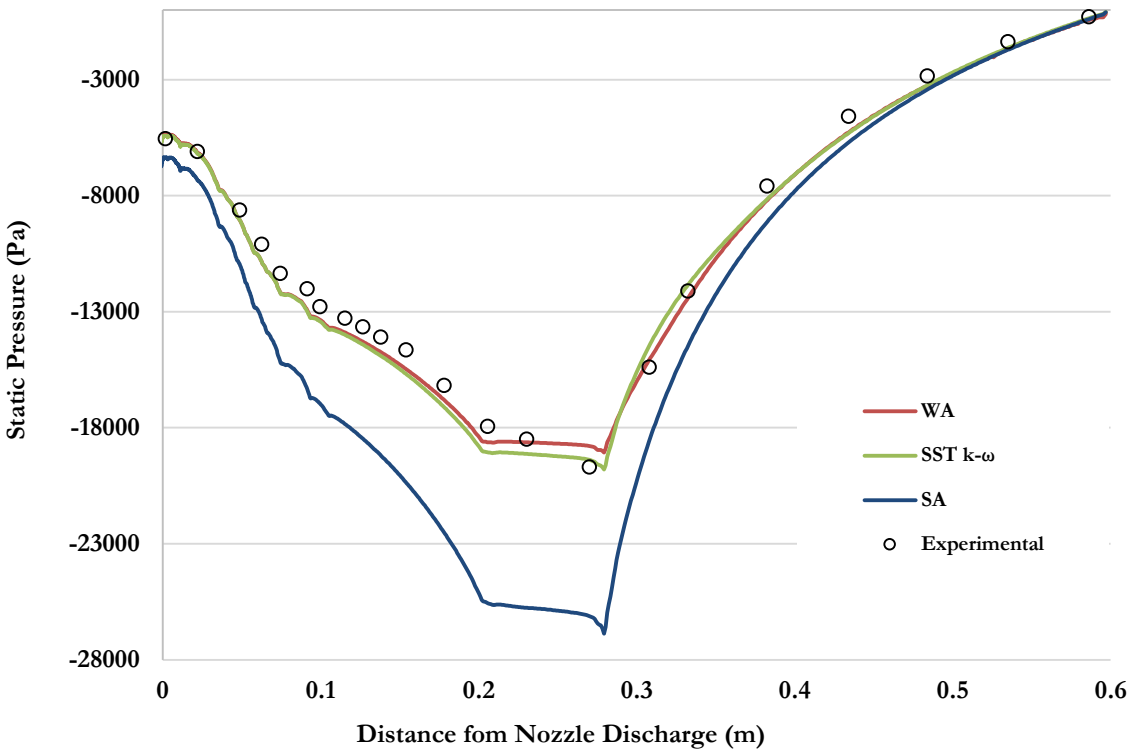


Figure 5.50: Comparison of the mixing section wall pressure distribution.

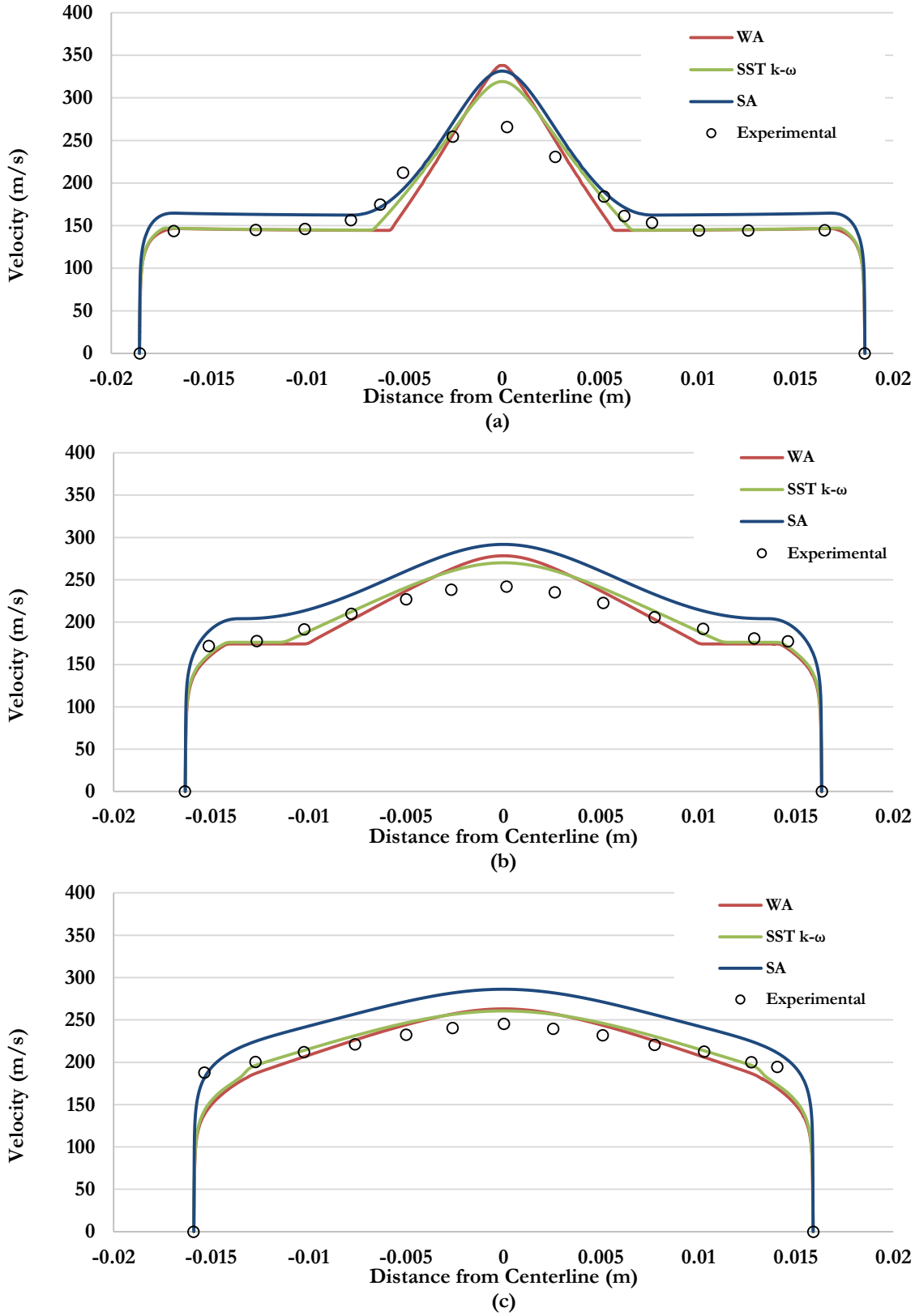


Figure 5.51: Comparison of the velocity profiles at locations (a)3", (b)7", and (c)10.5" downstream of the slot nozzle ejector.

5.4.3 Axisymmetric Shock Wave/Boundary Layer Interaction (SWBLI)

In this section results for the TMR Axisymmetric Shock Wave Boundary Layer Interaction are presented. Results are compared with data obtained from the experiment of Kussoy and Horstman [55]. High speed flow over an ogive cylinder reaches a flare designed to produce an oblique shock and shock wave boundary layer interaction. The experiment with a flare of 20° was considered in this case. The geometry is shown in Figure 5.52. This case has a freestream Mach number of $M = 7.11$ and Reynolds number $Re = 57{,}000$. The wall had constant temperature of $T_{wall} = 311K$. A family of computational grids is provided by the TMR website. Results presented below were obtained with the second-finest grid. The quantities of interest to compare are the nondimensional surface pressure and heat transfer.

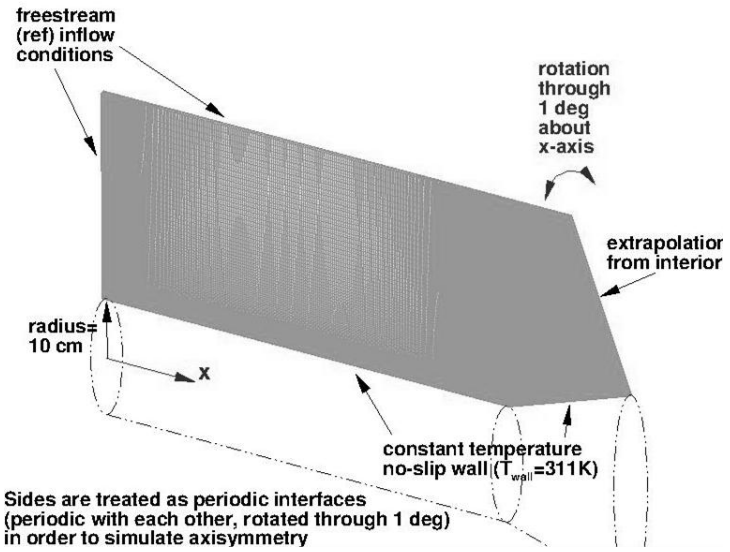


Figure 5.52: Computational domain of the axisymmetric SWBLI [5].

Figure 5.53 and Figure 5.54 compare the predicted pressure and heat transfer profiles along the surface with the experimental data. The pressure along the wall is comparable for all three models as seen in Figure 5.53. The SST k-ω model predicts a noticeable separation region at the beginning of the flare which is not seen in the experiment. All models slightly over predict the pressure in

the region near the shock. Further downstream the models underpredict the pressure. It can be seen in Figure 5.54 that the SA model is in best agreement with the experimental heat transfer. The WA predicts a sharp increase in the heat transfer at the beginning of the flare, similar to the SA model but more extreme. The SST *k-ω* and WA model slightly overpredict the heat transfer immediately following the shock. All three models return to the same heat transfer level after the flow downstream of the shock.

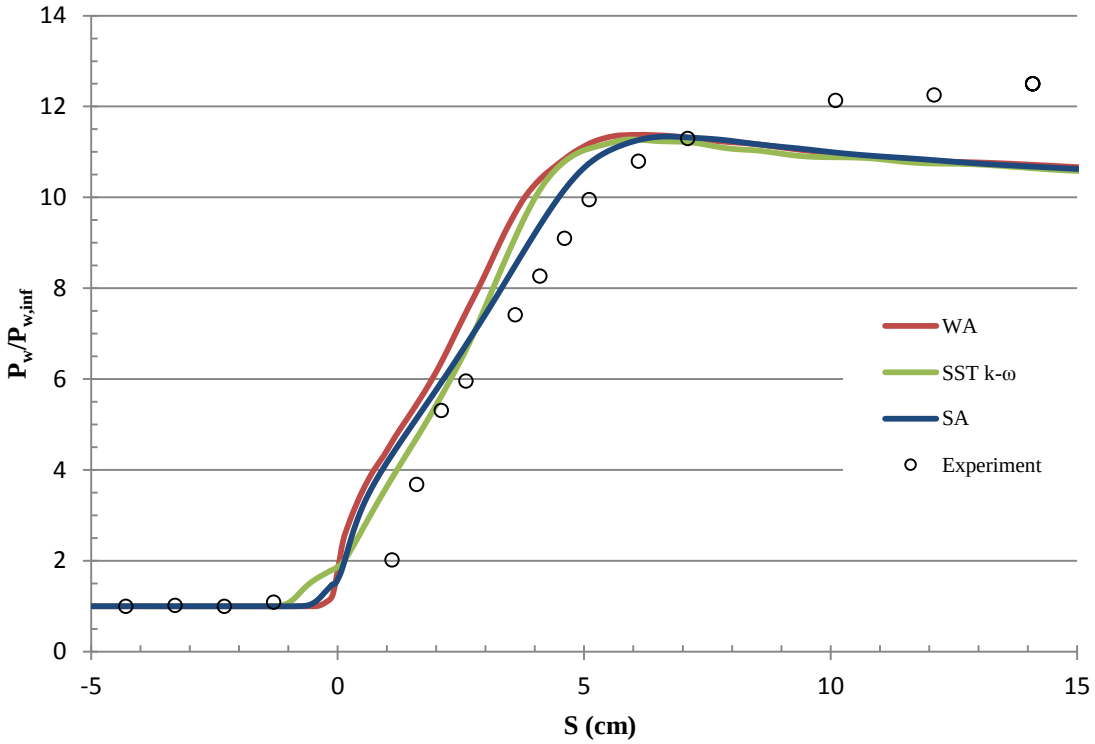


Figure 5.53: Comparison of the nondimensional wall pressure for the axisymmetric SWBLI

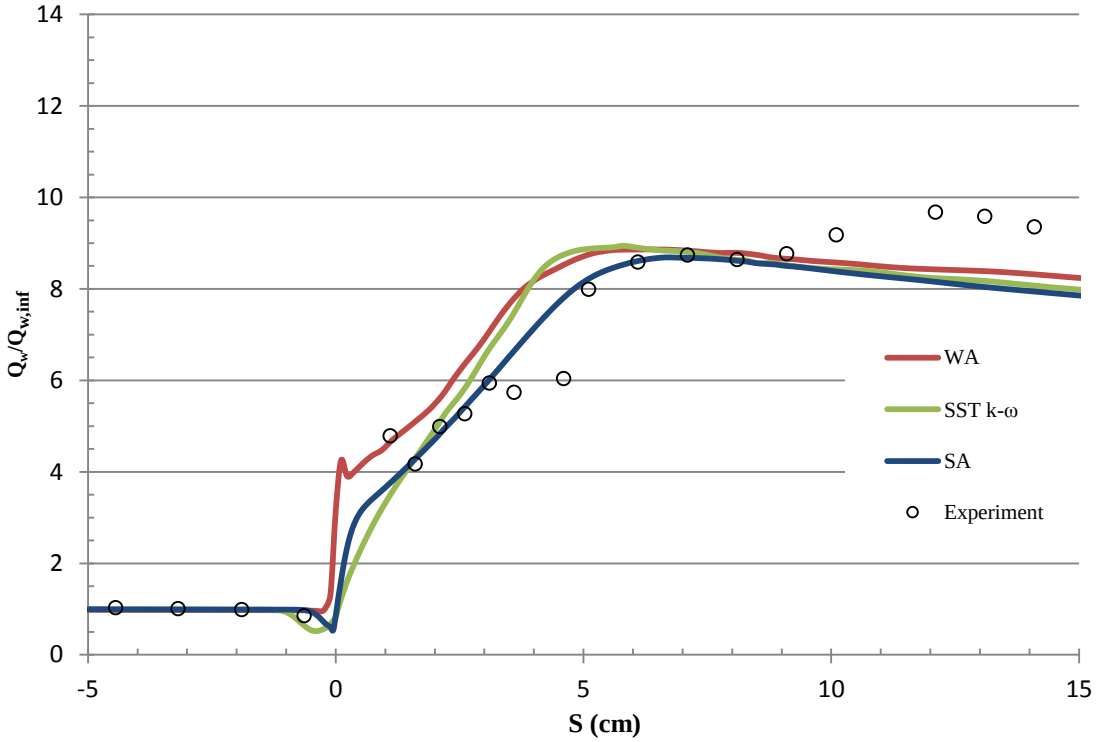


Figure 5.54: Comparison of the nondimensional wall heat flux for the axisymmetric SWBLI.

Chapter 6 Rotation/Curvature and Surface Roughness Corrections

6.1 Rotation and Curvature Corrections

6.1.1 Introduction

One of the major deficiencies of linear eddy-viscosity turbulence models is their inability to capture the effects of streamline curvature and system rotation. Since these effects are significant in many engineering flows, modification of existing turbulence models to account for these effects is of practical interest. The Spalart-Shur correction [56] has been shown to improve the predictive capability of the SA and SST k-ω turbulence models when applied to flows where streamline curvature and system rotation are present [57,58]. This motivates the application of the Spalart-Shur correction to the WA model.

In the following section, the WA turbulence model is modified to include the Spalart-Shur correction. The new model is then applied to compute the 2D flow in a curved channel. Along with WA-RC results, results from the original WA, the SA, SA-RC, SST k-ω, and SST-RC models are also obtained. The results from all the models are compared with experimental data. The results show that the new model more accurately predicts skin-friction coefficient and surface pressure coefficient than the base model.

6.1.2 The Spalart-Shur Correction

To account for rotation and curvature effects, the empirical function shown in Eq. 39 is used. This function multiplies the production term in the SA eddy viscosity transport equation and similarly multiplies the production terms of both k and ω equations in the SST k-ω model. For the WA model, implementation of the Spalart-Shur correction is also a straightforward modification to the

transport equation shown in Eq. 28. The source term ($C_1 RS$) is simply multiplied by the rotation function shown in Eq. 39.

$$f_{r1}(r^*, \tilde{r}) = (1 + c_{r1}) \left[\frac{2r^*}{1 + r^*}\right] [1 - c_{r3} \tan^{-1}(c_{r2}\tilde{r})] - c_{r1} \tag{39}$$

where r^* and $\tilde{r}$ are nondimensional quantities given by:

$$r^* = \frac{S}{\omega}, \qquad \tilde{r} = 2\omega_{ij} S_{jk} (\frac{\mathrm{D}S_{ij}}{\mathrm{D}t}) / D^4 \tag{40}$$

$$S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \qquad \omega_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}) \tag{41}$$

$$D^2 = \omega^2{}_{ij} + S^2{}_{ij} \tag{42}$$

The derivative in Eq. 40, $\mathrm{D}S_{ij}/\mathrm{D}t$, are the components of the Lagrangian derivative of the strain tensor. For finite volume method codes, it may be more convenient to represent the derivative using an Eulerian formulation. This procedure is outlined in Ref. 58 and was used in the present OpenFOAM implementation. The three model constants $c_{r1} = 1.0$, $c_{r2} = 8.0$, and $c_{r3} = 1.0$ were calibrated for the WA model.

6.1.1 2D Convex Curvature Boundary Layer

In this section results for the TMR 2D Convex Curvature Boundary Layer are presented. The focus of this case is to assess the capability of turbulence models to capture the effects of curvature on the boundary layer formation. This flow has been computed at the conditions corresponding to the experiment of Smits et al. [59]. The experiment measured flow through a constant area square duct of height *0.127* m with a rapid *30°* bend. The aspect ratio of the experimental duct was 6:1, but the case was modeled as 2D in the present computation. The duct was essentially reduced to a channel flow with Mach number *M = 0.093* and Reynolds number *Re = 2.1x10^6*. A family of computational grids is provided by the TMR website. Results presented below were obtained with

the second-finest grid. The computational grid is shown in Figure 6.1. The experiment showed the presence of Görtler vortices which cannot be predicted in the 2D simulation. For this reason, comparison of the CFD and experimental results should be considered ambiguous but a comparison among the models can still be made. The quantities of interest to compare are the pressure coefficient C_p on the convex wall and the skin friction coefficient C_f along the convex wall.

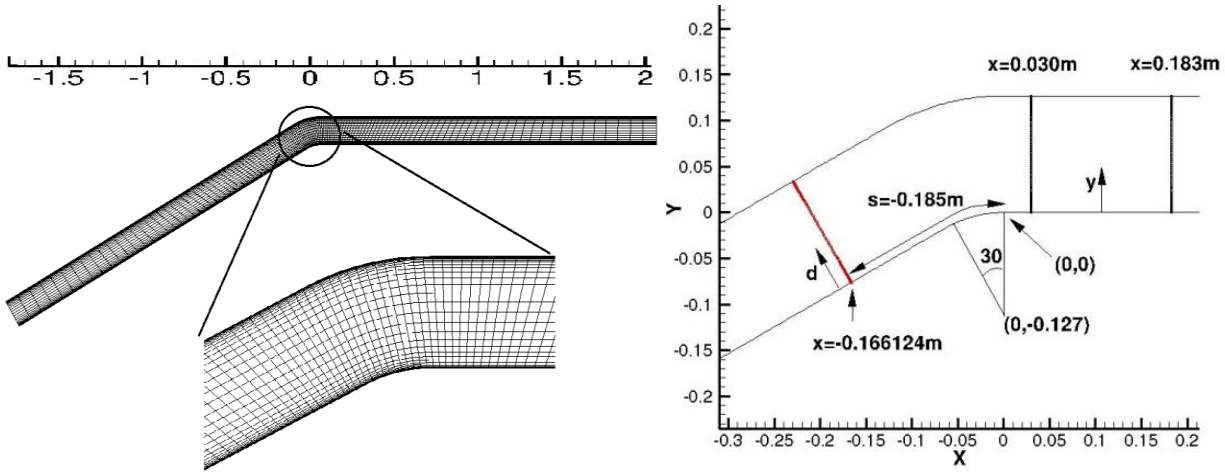


Figure 6.1: Computational grid and coordinate system of the convex curvature boundary layer[5].

Figure 6.2 shows the calculated and experimental values of the pressure coefficient along the convex wall. It can be clearly seen that there is no discernible difference between the turbulence models and every model is in excellent agreement with the experimental data. The addition of the Spalart-Shur correction has no effect on this quantity. Comparison of the calculated and experimental skin friction coefficients are shown in Figure 6.3. The positive effect of Spalart-Shur correction is clearly demonstrated. The results of the SA-RC and SST-RC are in excellent agreement with FUN3D results from the NASA TMR [5], verifying their implementation in OpenFOAM. An improvement in the accuracy of each base model is accomplished with the addition of the RC correction.

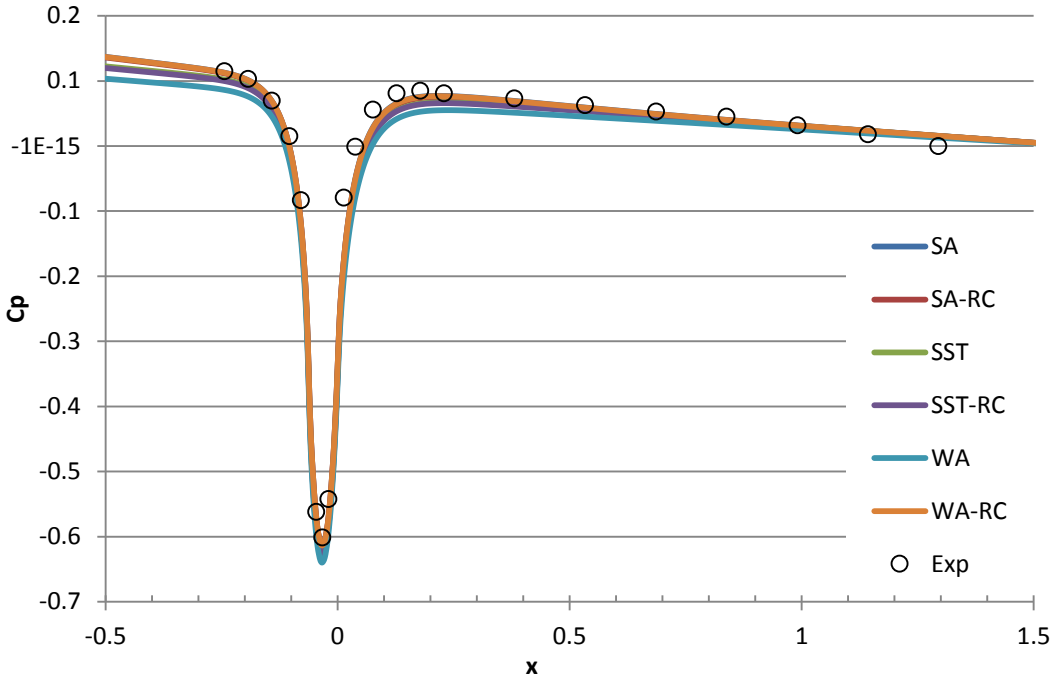


Figure 6.2: Comparison of the convex wall pressure coefficient.

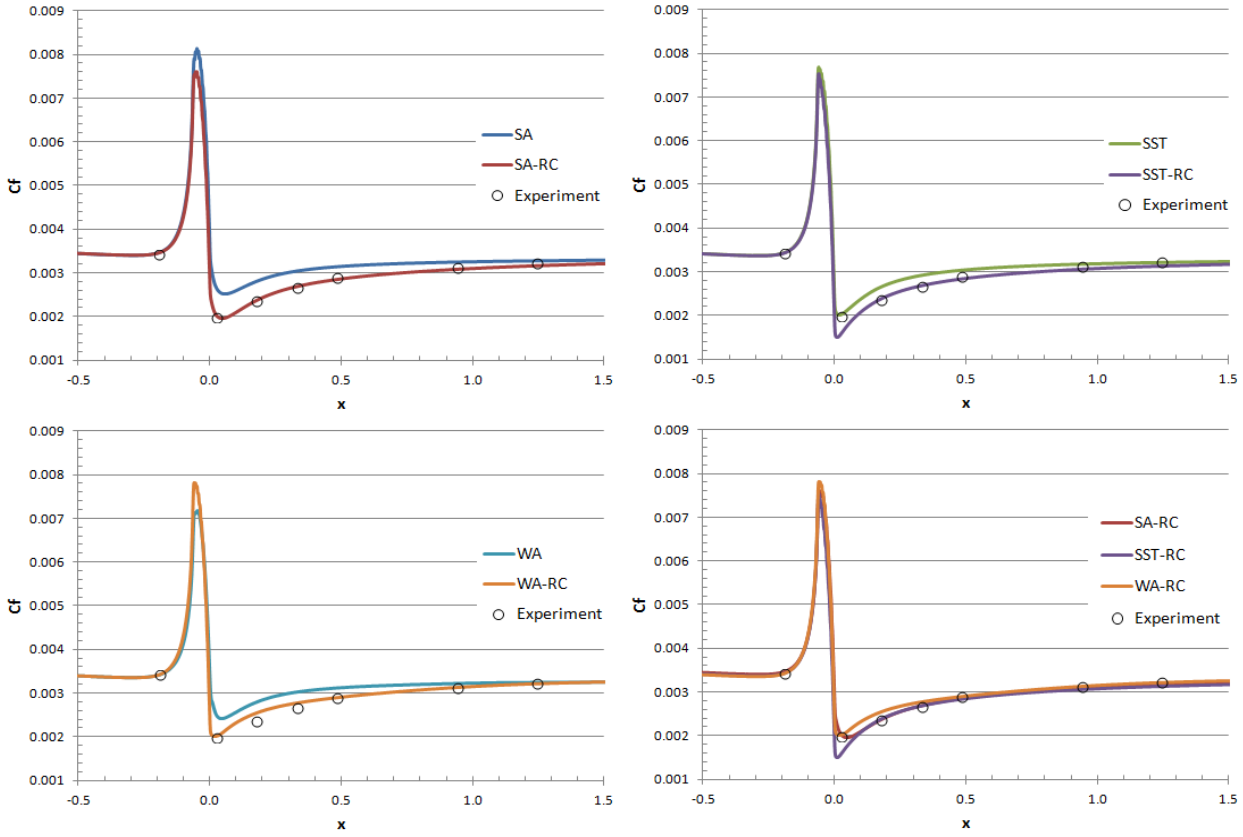


Figure 6.3: Comparison of the convex wall skin friction coefficient.

6.2 Surface Roughness

6.2.1 Introduction

Many aerodynamic and heat transfer applications require a model that can accurately simulate flow over smooth and rough surfaces. Surface roughness is generally caused by manufacturing or by some environmental degradation of the surface material, such as erosion. Even though surface conditions play a major role in boundary-layer characteristics, most turbulence models require an additional modification in order to account for it. Various surface-roughness modelling techniques have been suggested. For example, a wall-function approach offers a simple and easy to implement modification, but has not found broad adoption because wall-functions are reliable only for simple, attached flows. More common and robust methods of surface-roughness modelling modify the boundary conditions of the turbulence variable as a function of the surface roughness.

The most common method used to model surface roughness is the equivalent sand grain approach based on the work of Nikuradse [60]. In the equivalent sand grain approach an equivalent sand grain height, k_s, is assigned to the surface. This height is determined based on empirical correlations and the geometry of the surface. The need for additional geometric parameters in order to define k_s is a problem when dealing with complex geometries. Also, although the inclusion of k_s improves the accuracy of the temperature distribution, the correlation between k_s and the heat transfer has no physical basis. Despite these drawbacks it is still very widely used.

In the next section a surface roughness modification is presented for the WA turbulence model. Surface roughness modifications based on the equivalent sand grain approach exist for both the SA [61] and SST k-ω turbulence models [62]. The three modified models with roughness corrections, designated WA-rough, SST k-ω-rough, and SA-rough, were implemented into OpenFOAM. The models were then used to simulate the flow over a rough flat plate and rough

S809 airfoil and were compared with experimental and empirical data. The results show that the WA-rough model accurately accounts for surface roughness characteristics.

6.2.2 WA Model Roughness Correction

The equivalent sand grain approach represents the physical roughness with an idealized roughness, based on the work of Nikuradse [60] along with empirical correlations. The roughness effect is captured by increasing the eddy viscosity in the near wall region as a function of the equivalent sand grain height. Nikuradse showed that for fully rough surfaces, a shift occurred in the boundary layer velocity profiles. The velocity profile in the log-layer obeys:

$$u^+ = \frac{1}{\kappa} ln \frac{y}{k_s} + 8.5$$

The development of the WA-rough model follows the procedure of the SA-rough model. The wall distance, *d*, is modified to match the expected shift in the log-layer. The value of *d*, present in the blending function is replaced with d_{new} so that the switch between destruction terms still occurs near the top of the log-layer.

$$d_{new} = d + 0.03k_s$$

The viscous damping must also be modified to give realistic viscous sublayer and buffer layer profiles. This was accomplished by replacing Eq. (8) with the following:

$$f_\mu = \frac{\chi^3}{\chi^3 + C_w^3}, \qquad \chi = \frac{R}{\nu} + C_{r1} \frac{k_s}{d}$$

where $C_{r1}=0.5$ and the value of C_w becomes $C_w=13.0$.

A non-zero value of *R* is used at the wall to capture the increase in the eddy viscosity. Therefore the wall boundary condition, $R_{wall}=0$, becomes:

$$\frac{\partial R}{\partial n} = \frac{R}{d_{new}}$$

where *n* is the wall normal vector.

The boundary condition modification of *R* does not give a large enough increase in the eddy viscosity near the wall at higher roughness values. To increase the eddy viscosity further, the coefficient of *k-ω* destruction term is modified. The destruction coefficient $C_{2k\omega}$ in Eq. 28 is replaced by Eq. 43 with $C_{r2}=0.006$.

$$f_r = C_{2k\omega}\left(\frac{1}{1+\frac{C_{r2}k_s}{d_{new}}}\right) \tag{43}$$

6.2.3 2D Rough flat plate

Boundary layers over uniformly roughened subsonic zero pressure gradient flat plates were computed for varying surface roughness. A uniform inlet velocity of $U_{ref} = 66.3$ *m/s* was used with fully turbulent inlet conditions. The height of the first near-wall grid node was at $y^+ < 1.0$ across the entire plate. Mills and Hang [63] have deduced the following semi-empirical formula for the skin friction coefficient on a sand-roughened flat plate:

$$c_f = \left(3.476 + 0.707 ln\frac{x}{k_s}\right)^{-2.46} \tag{44}$$

Figure 6.4-6.7 show the computed skin friction distributions of the WA-rough, SA-rough, and SST *k-ω*-rough models for four surface roughness values. It can be seen that the WA-rough and SST *k-ω*-rough models are in very close agreement with Eq. (16) for low roughness cases. As the surface roughness increases, the WA-rough and SST *k-ω*-rough models begin to under predict the skin friction coefficient near the leading edge of the plate. The SA model significantly underpredicts the skin friction coefficient for the entire roughness regime.

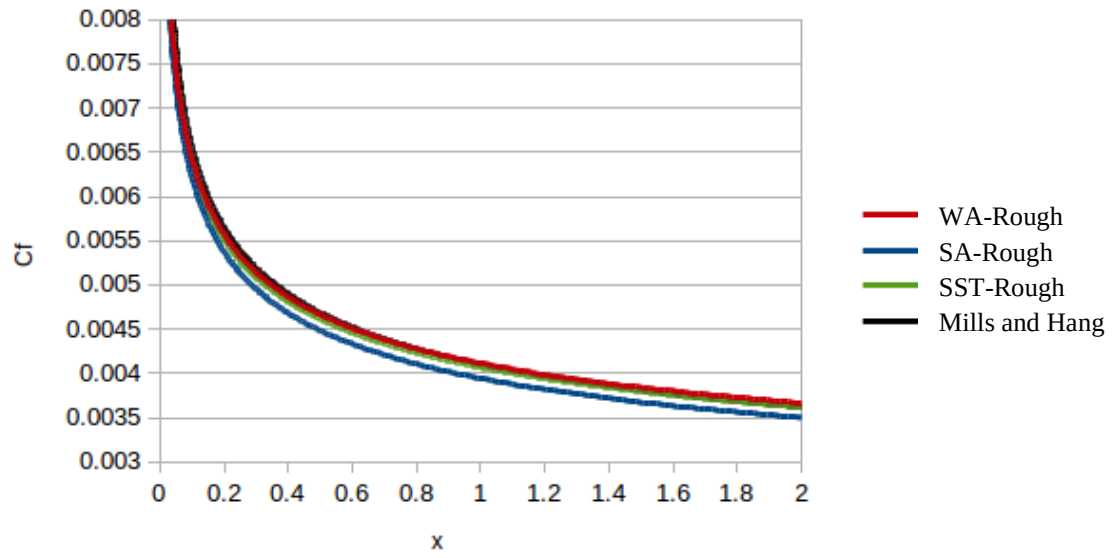


Figure 6.4: Comparison of skin friction coefficients for a roughness of k_s=0.00025.

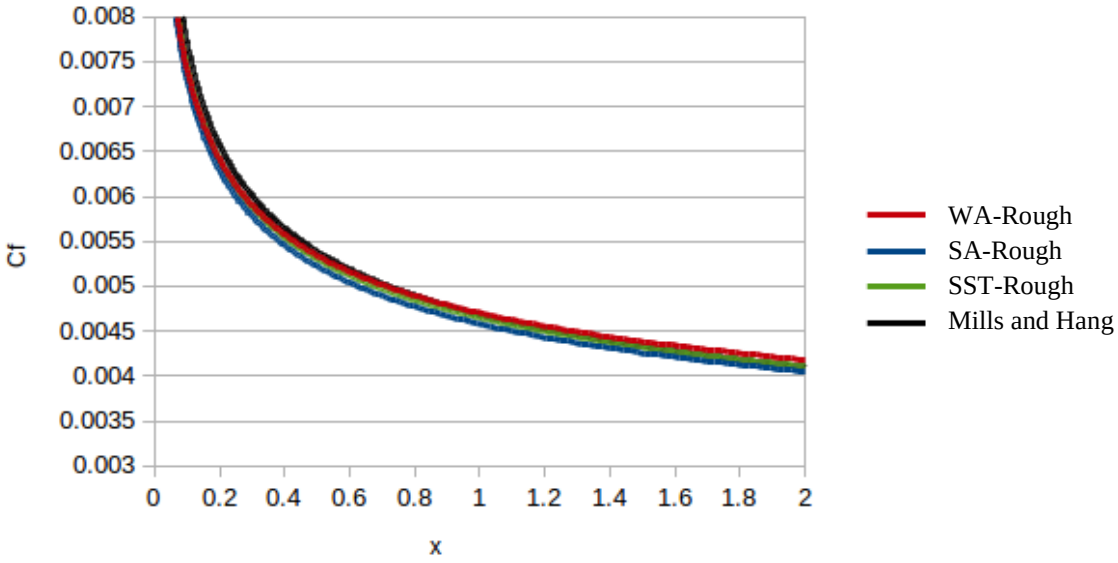


Figure 6.5 Comparison of skin friction coefficients for a roughness of k_s=0.0005.

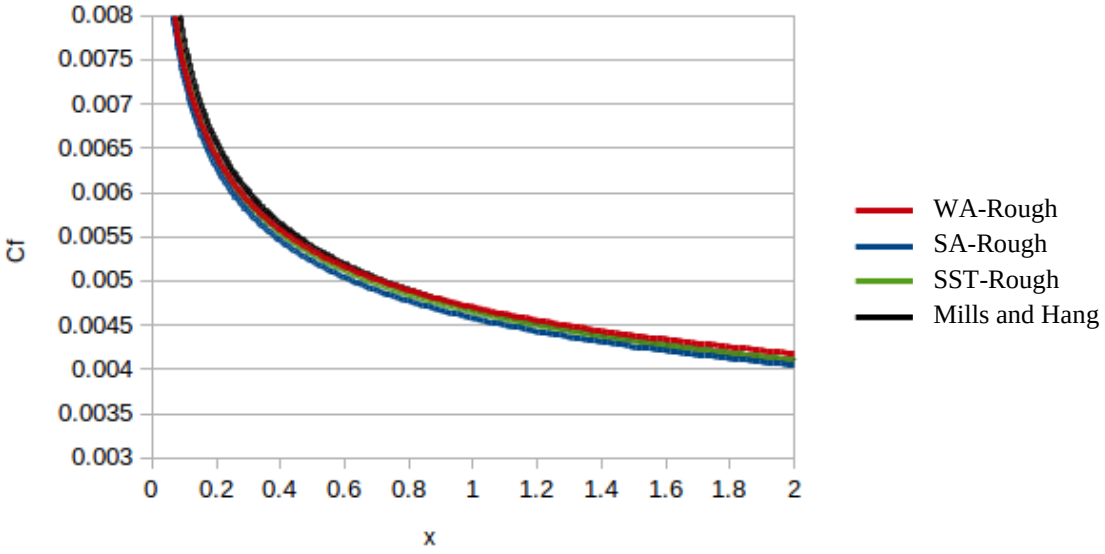


Figure 6.6: Comparison of skin friction coefficients for a roughness of k_s=0.0010.

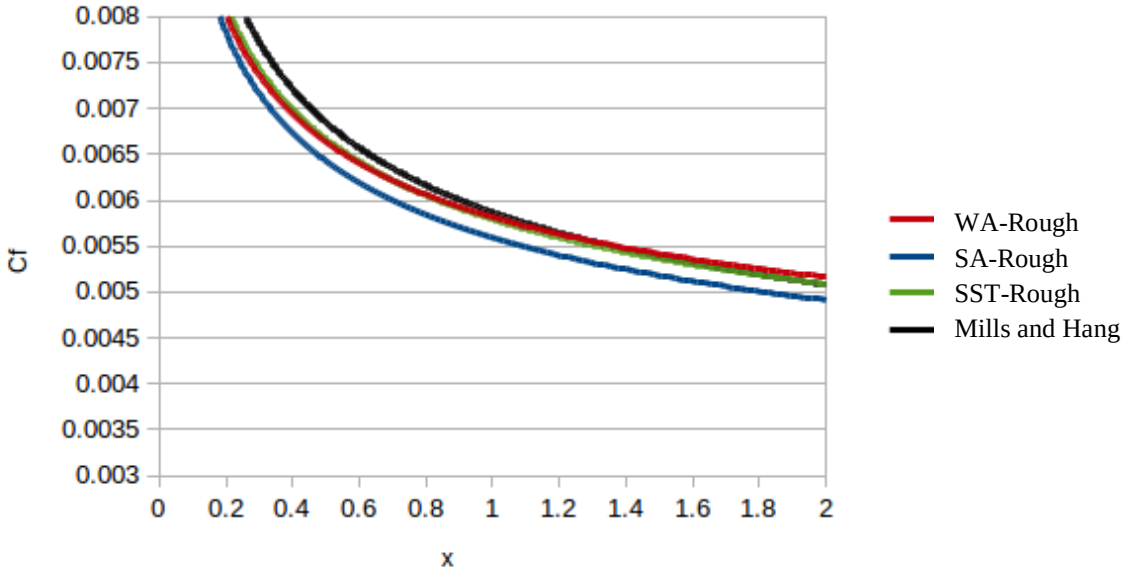


Figure 6.7: Comparison of skin friction coefficients for a roughness of k_s=0.0015.

6.2.4 2D Rough S809 Airfoil

The next case examined is flow over a S809 airfoil. The S809 airfoil is commonly used in horizontal-axis wind turbines. A leading edge grit roughness was applied to simulate surface irregularities that occur on wind turbine blades. These irregularities are caused by the accumulation of insect debris, ice, and erosion. The effect of surface roughness on these airfoils is of importance because the surface conditions directly affect the efficiency of the of the wind turbines. Accurate simulation of this roughness effect will directly aid in the design of more efficient wind turbines.

The simulation conditions correspond to the experiments conducted by Somers at the National Renewable Energy Laboratory [64] and by Ruess Ramsay et. al [65]. Although experiments were carried out for 2D airfoil and 3D blade, only the two-dimensional case is considered in the present effort. A chord Reynolds number of 1 million is considered in the present study since it was used in both the experiments. The experimental steady state results from Refs. 64 and 65 showed a baseline maximum lift coefficient of 1.03 at an angle of attack of 15°. The application of leading edge roughness reduced the maximum lift coefficient by 15.5%.

Figure 6.8 and Figure 6.9 show the lift coefficient as a function of angle attack. In Figure 6.8, the simulation results of the WA, SST k-ω and SA models are compared to the experimental data. It can be seen that the turbulence models agree with the experimental data for small angles of attack but greatly overpredict the lift coefficient for angles of attack larger than 8°. Figure 6.9 shows the results of the SST k-ω model and a laminar-turbulent transition model, called the SST k-ω Re_θ-γ model [66]. It can be seen that the use of a transition model significantly improves the quality of the solution for large angles of attack. The prediction of the laminar separation bubble present near the leading edge of the airfoil is extremely important. The SST k-ω Re_θ-γ model, however, does not account for surface roughness and cannot be applied to the rough airfoil case. This motivates

the development of laminar-turbulent transition models sensitized to roughness effects, which has recently been investigated by Hou [67].

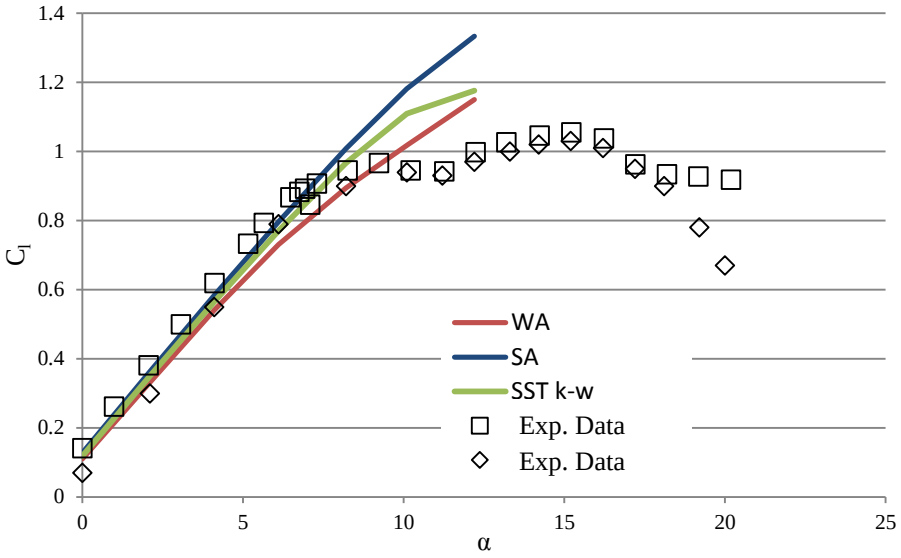


Figure 6.8: Comparison of lift coefficients for smooth S809 airfoil.

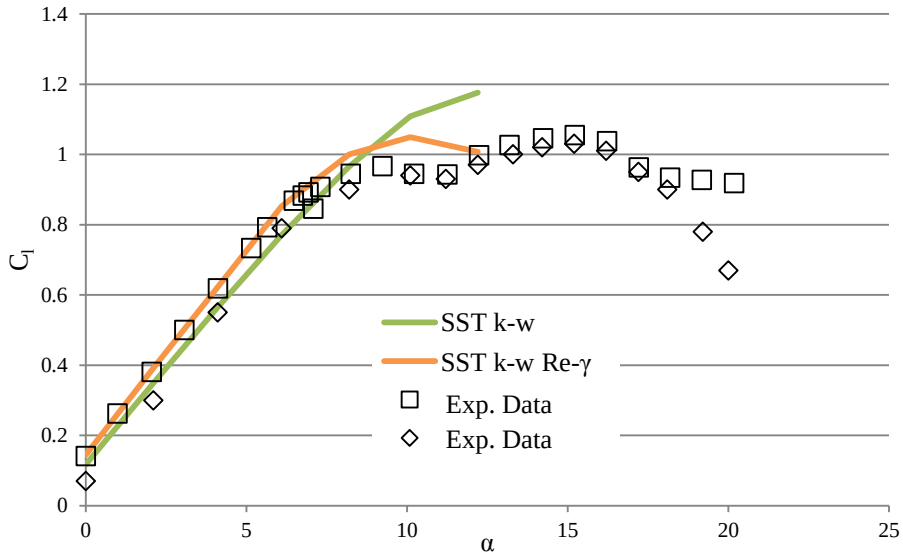


Figure 6.9: Transitional model effect on the lift coefficients for smooth S809 airfoil.

For the experimental roughness case of Ref. 65, a standard pattern was developed using a molded insect pattern taken from a wind turbine. Based on average particle size from the field specimen, a standard #40 lapidary grit was used for the roughness elements, giving $k_s/c = 0.0019$. Figure 6.10 shows the predicted lift coefficients as a function of angle of attack for the WA-rough, SST *k-ω*-

rough, and SA-rough models. It can be seen that all three models correctly predict a decrease in lift coefficient compared to the smooth airfoil case. SA-rough and WA-rough most closely match the experimental lift coefficients. SA-rough slightly overpredicts the lift coefficients while WA-rough slightly underpredicts the lift coefficient. SST *k*-ω-rough significantly underpredicts the lift coefficient for angles of attack larger than 2°.

A surface roughness correction was successfully applied to the WA turbulence model. The new model is then compared with the SA-rough and SST *k*-ω-rough models. The three models were implemented in OpenFOAM and verified for rough flat plate flows. An improvement in the accuracy of each base model is accomplished with the addition of the surface roughness correction.

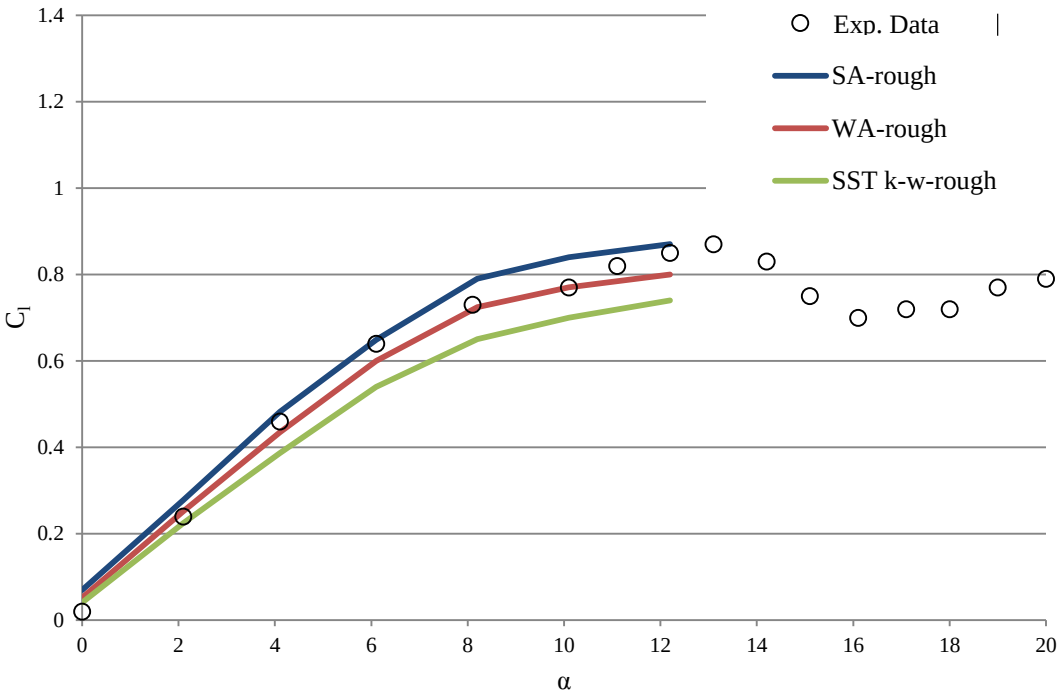


Figure 6.10: Comparison of lift coefficient for rough S809 airfoil.

Chapter 7: Summary and Future Work

7.1 Summary

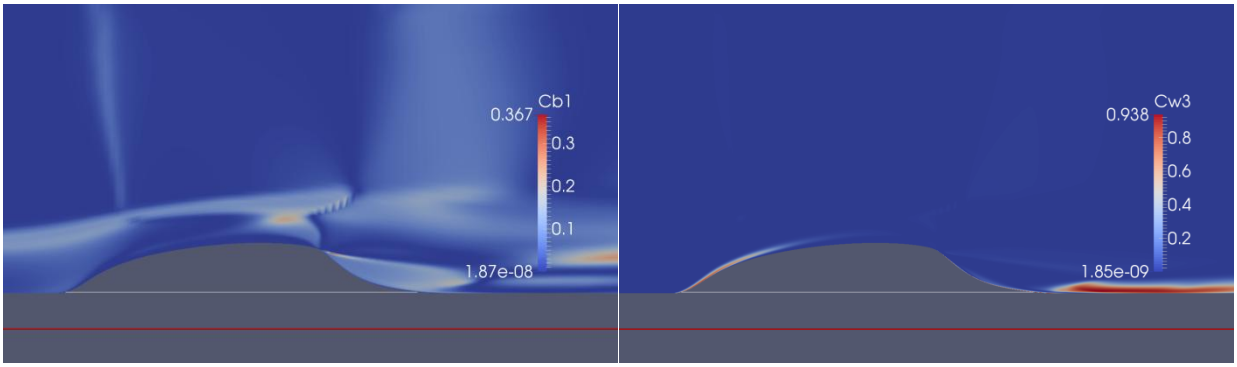
The new one-equation Wray-Agarwal model was successfully developed and applied to simulate a large number of benchmark flows. Cases included freeshear flows, wall bounded flows, flows with separation, subsonic, transonic, and supersonic flows, and SWBLIs. Additionally a rotation/curvature correction and surface roughness correction were applied to the WA model to extend its applicability to these flows. In every case considered the WA model did as well as or better than the SA model. Two flow types with the most noticeable benefit from the use of the WA model were the adverse pressure gradient flows of the NACA4412 airfoil and asymmetric diffuser and the jet flow of the 2D slot nozzle ejector. The SA model was shown to be incapable of accurately predicting the separation due to the adverse pressure gradients. Also the SA model could not predict the spreading and entrainment of the 2D slot nozzle ejector. This led to an under prediction of the pressure along the mixing section wall. Adverse pressure gradient and jet flows are typically simulated using the two-equation models. The behavior of the WA model is very similar to the SST k-ω model for these types of flows. Based on the results shown in this dissertation, the WA model can provide the industry and CFD researchers an accurate one-equation alternative for the computation of a large class of turbulent flows.

An additional accomplishment of this research has been the implementation and verification of the SA, SA-RC, SA-Rough, SST k-ω, SST-RC, and SST-Rough turbulence models in OpenFOAM. The implementation of these industry standard models into an open-source package will allow CFD users and researchers to investigate a wide range of engineering flows without the purchase of proprietary codes or the use of user restricted government codes.

7.2 Future Work: Turbulence Model Closure Coefficients Sensitivity Analysis

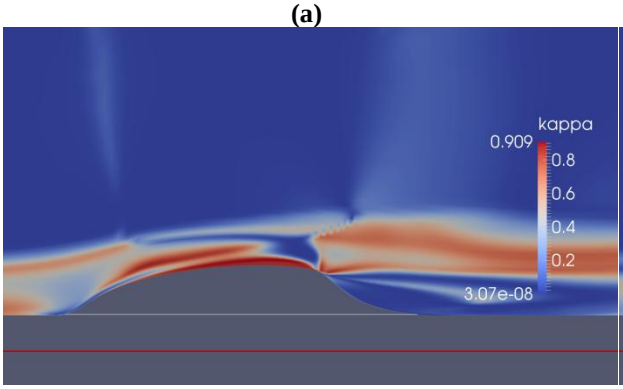
Interest in uncertainty quantification (UQ) in CFD has grown in recent years. UQ has been successfully applied to design, optimization, and modeling problems, and is becoming a standard tool for verification and validation of numerical solutions. The development of non-intrusive UQ methods has reduced the computational expense of UQ and has allowed uncertainty propagation through complex models without alteration of the underlying model.

In this section the sensitivities of the closure coefficients of the SA model are presented. The subsonic flow over the NASA wall-mounted hump is considered. A non-intrusive Kriging model [68] is used to propagate the uncertainty of the closure coefficients. DAKOTA [69] is used to calculate the Sobol indices which quantify the sensitivity of some quantity of interest to each turbulence model coefficient. Previous studies have calculated the sobol indices along the bottom wall of the hump [70]. The preliminary results given below extend the analysis to the entire flow field. The benefit of this analysis is that the model coefficients can be linked to flow features instead of a single local quantity.

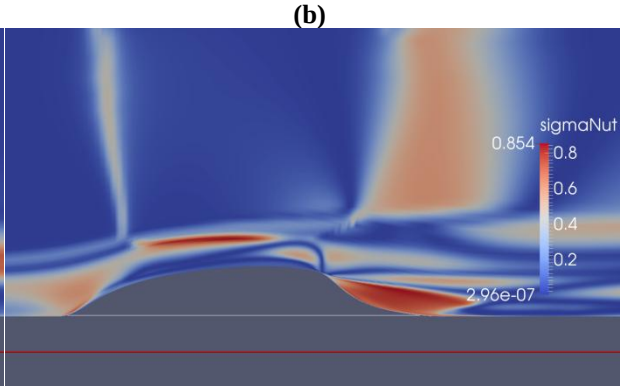


(a)

(b)



(c)



(d)

Figure 7.1: SA Closure Coefficient Sobol Indices.

The results of this sensitivity analysis are shown in Figure 7.1. The quantity of interest investigated here is the Reynolds stress. Results from Section 5.2.9 showed that the SA model overpredicts the reattachment length. This is due to the under prediction of the Reynolds stress. The results of the sensitivity analysis demonstrate that modification of diffusion constant, σ_v, and the production constant, C_{b1}, will have the largest impact on the Reynolds stress in the separation region. Also von Karman's constant, κ, has a large influence on the boundary layer over the hump but not in the separation region. The model constant C_{w3} is very large only in the recovery region. Identifying the coefficients that affect these flow features will aid modelers in improving the accuracy of turbulence model predictions for these flows.

References

1. Spalart, P. R., "Strategies for Turbulence Modeling and Simualations," Int. J. Heat Fluid Flow, Vol. 21, 2000, pp. 252-263.

2. Spalart, P. R. and Allmaras, S. R, "A One Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 1992-0439, 1992.

3. Menter, F. R., "Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications," AIAA J., Vol. 32, No. 8, August 1994, pp. 1598-1605.

4. Wilcox, D. C., "Formulation of the *k-ω* Turbulence Model Revisited," AIAA Journal, Vol. 46, No. 11, 2008, pp. 2823-2838

5. NASA Langley Turbulence Modeling Resource website, http://turbmodels.larc.nasa.gov. [retrieved November 2016]

6. Eca, L., Hoekstra, M., Hay, A., and Pelletier, D., "A Manufactured Solution for a Two-Dimensional Steady Wall Bounded Incompressible Turbulent Flow," International Journal of Computational Fluid Dynamics, Vol. 21, 2007, pp. 175-188.

7. Menter, F. R., Kuntz, M., and Langtry, R., "Ten Years of Industrial Experience with the SST Turbulence Model," Turbulence, Heat and Mass Transfer 4, ed: K. Hanjalic, Y. Nagano, and M. Tummers, Begell House, Inc., 2003, pp. 625 - 632.

8. CFL3D Version 6 website, http://cfl3d.larc.nasa.gov. [retrieved November 2016]

9. FUN3D website, http://fun3d.larc.nasa.gov. [retrieved November 2016]

10. The Spalart-Allmaras Turbulence Model, NASA Langley Turbulence Modeling Resource website, http://turbmodels.larc.nasa.gov/spalart. [retrieved November 2016]

11. Rumsey, C. L., "Apparent Transition Behavior of Widely-Used Turbulence Models," International Journal of Heat and Fluid Flow, Vol. 28, 2007, pp. 1460-1471.

12. Baldwin, B. S., and Barth, T. J., "A One-Equation Turbulence Model for High Reynolds Number Wall Bounded Flows," NASA TM-102847, 1990.

13. Menter, F. R., "Eddy Viscosity Transport Equations and Their Relation to the k-ε Model," Journal of Fluids Engineering, Vol. 119, 1997, pp. 876-884.

14. Goldberg, U., "Hypersonic Flow Heat Transfer Prediction Using Single Equation Turbulence Models," Journal of Heat Transfer, Vol. 123, 2001, pp. 65-69.

15. Nagano, Y., Pei, C. Q., and Hattori, H., "A New Low-Reynolds-Number One-Equation Model of Turbulence," Flow, Turbulence and Combustion, Vol. 63, 2000, pp. 135-151.

16. Fares, E., and Schröder, W., "A General One-Equation Turbulence Model for Free Shear and Wall-bounded Flows," Flow, Turbulence and Combustion, Vol. 73, 2005, pp. 187-215.

17. Elkhoury, M., "Modified Menter Model in Comparison with Recently Developed Single-Equation Turbulenc Closures," AIAA Journal, Vol. 49, 2011, pp.1399-1408.

18. Elkhoury, M., "Assessment and Modification of One-Equation Models of Turbulence for Wall Bounded Flows," Journal of Fluids Engineering, Vol 129, 2007, pp. 921-928.

19. Rahman, M., Siikonen, T., and Agarwal, R. K., "Improved Low-Reynolds-Number One-Equation Turbulence Model," AIAA Journal, Vol. 49, 2011, pp.735-747.

20. Rahman, M., Agarwal, R. K., and Siikonen, T., "A Modified One-Equation Turbulence Model Based on Turbulent Kinetic Energy Equation, 54th AIAA Aerospace Sciences Meeting, AIAA 2016-1598.

21. Rahman, M., Agarwal, R. K., and Siikonen, T., "One-Equation Turbulence Model Based on ε-Equation," 46th AIAA Fluid Dynamics Conference, AIAA 2016-3643.

22. Bradshaw, P., Ferriss, D. H., and Atwell, N. P., "Calculation of Boundary Layer Development Using the Turbulent Energy Equation," Journal of Fluid Mechanics, Vol. 28, 1967, pp. 593-616.

23. Townsend, A. A., "Equilibrium Layers and Wall Turbulence," Journal of Fluid Mechanics, Vol. 11, 1962.

24. Wilcox, D. C., *Turbulence Modeling for CFD*, 3rd ed., DCW Industries, Inc., La Canada, CA, 2006.

25. Bardina, J. E., Huang, P. G., and Coakley, T. J., "Turbulence Modeling Validation, Testing, and Development," NASA TM-110446, 1997.

26. Fage, A., and Falkner, V. M., "Note on Experiments on the Temperature and Velocity in the Wake of a Heated Cylindrical Obstacle," Proc. R. Soc., London, Vol. A135, 1932, pp. 702-705.

27. Bradbury, L. J. S., "The Structure of a Self-Preserving Turbulent Plane Jet," Journal of Fluid Mechanics, Vol. 23, 1965, pp. 31-64.

28. Wygnanski, I., and Fiedler, H. E., "Some Measurements in the Self-Preserving Jet," Journal of Fluid Mechanics, Vol. 38, 1969, pp. 577-612.

29. Witze, P. O., and Dwyer, H. A., "The Turbulent Radial Jet," Journal of Fluid Mechanics, Vol. 75, 1967, pp. 401-417.

30. Mellor, G. L., and Herring, H. J., "Two Methods of Calculating Turbulent Boundary Layer Behavior Based on Numerical Solution of the Equation of Motion," Computation of Turbulent Boundary Layers-1968 AFOSR-IFP-Stanford Conference Proceedings.

31. Menter, F. R., "Influence of Free-stream Values on k-ω Turbulence Model Predictions," AIAA Journal, Vol. 30, No. 6, 1992.

32. NPARC Alliance National Program for Applications-Oriented Research in CFD website, https://www.grc.nasa.gov/WWW/wind/ [retrieved November 2016]

33. Delville, J., Bellin, S., Garem, J. H., and Bonnet J. P., “Analysis of Structures in a Turbulent, Plane Mixing Layer by Use of Pseudo Flow Visualization Method Based on Hot-Wire Anemometry,” in: Advances in Turbulence 2, eds: H.-H. Fernholz and H. E. Fiedler, Proceedings of the Second European Turbulence Conference, Berlin, Aug 30-Sept 2, 1988, Springer Verlag, Berlin, 1989, pp. 251-256.

34. Nakayama, A., "Characteristics of the Flow around Conventional and Supercritical Airfoils," J. Fluid Mech. (1985), Vol. 160, pp. 155-179.

35. Weighardt K. and Tillman, W., “On the Turbulent Friction Layer for Rising Pressure,” NACA TM-1314, 1951.

36. Coles D., “The Law of the Wake in the Turbulent Boundary Layer.” Journal of Fluid Mechanics, Vol. 1, 1956, pp. 191-226.

37. Lee, M. and Moser, R. D., “Direct numerical simulation of turbulent channel flow up to $Re_\tau = 5200$,” Journal of Fluid Mechanics, vol. 774, 2015, pp. 395-415.

38. Ladson, C. L., "Effects of Independent Variation of Mach and Reynolds Numbers on the Low-Speed Aerodynamic Characteristics of the NACA 0012 Airfoil Section," NASA TM 4074, October 1988

39. Gregory, N. and O'Reilly, C. L., "Low-Speed Aerodynamic Characteristics of NACA 0012 Aerofoil Sections, including the Effects of Upper-Surface Roughness Simulation Hoar Frost," NASA R&M 3726, Jan 1970.

40. Driver, D. M. and Seegmiller, H. L., "Features of Reattaching Turbulent Shear Layer in Divergent Channel Flow," AIAA Journal, Vol. 23, No. 2, Feb 1985, pp. 163-171.

41. Bruice, C. and Eaton, J. K., “Experimental Investigation of Flow through an Asymmetric Plane Diffuser.” Department of Mech. Engineering, Thermoscience Div., Rept. TSD-107, Stanford University, California, CA, 1997.

42. Coles, D. and Wadcock, A. J., "Flying-Hot-Wire Study of Flow Past an NACA 4412 Airfoil at Maximum Lift," AIAA Journal, Vol. 17, No. 4, April 1979, pp. 321-329.

43. Wadcock, A. J., "Structure of the Turbulent Separated Flow around a Stalled Airfoil," NASA-CR-152263, February 1979.

44. Seifert, A. and Pack, L. G., "Active Flow Separation Control on Wall-Mounted Hump at High Reynolds Numbers," AIAA Journal, Vol. 40, No. 7, July 2002.

45. Greenblatt, D., Paschal, K. B., Yao, C.-S., Harris, J., Schaeffler, N. W., and Washburn, A. E., "A Separation Control CFD Validation Test Case, Part 1: Baseline and Steady Suction," AIAA Journal, Vol. 44, No. 12, 2006, pp. 2820-2830.

46. Rumsey, C. L., Gatski, T. B., Sellers III, W. L., Vasta, V. N., and Viken, S. A., “Summary of the 2004 Computational Fluid Dynamics Validation Workshop on Synthetic Jets,” AIAA Journal, Vol. 44, No. 2, 2006, pp. 194-207.

47. Driver, D. M., "Reynolds Shear Stress Measurements in a Separated Boundary Layer Flow," AIAA Paper 91-1787, from the AIAA 22nd Fluid Dynamics, Plasma Dynamics, and Lasers Conference, June 1991, Honolulu, HI.

48. Wellborn, S.R., Reichert, B.A, and Okiishi, T.H., "An Experimental Investigation of the Flow in a Diffusing S-Duct," AIAA Paper 92-3622, 28th Joint Propulsion Conference and Exhibit, Nashville, TN, 6-8 July 1992.

49. Wellborn, S.R., Okiishi, T.H., and Reichert, B.A, "A Study of Compressible Flow through a Diffusing S-Duct," NASA Technical Memorandum 106411, December 1993.

50. Cook, P.H., McDonald, M.A., and Firmin, M.C.P., "Aerofoil RAE 2822 - Pressure Distributions, and Boundary Layer and Wake Measurements," Experimental Data Base for Computer Program Assessment, AGARD Report AR 138, 1979.

51. Bachalo, W. D., and Johnson, D. A., "An Investigation of Transonic Turbulent Boundary Layer Separation Generated on an Axisymmetric Flow Model," AIAA Paper 79-1479.

52. White, F. M., "Viscous Fluid Flow," McGraw-Hill, 1974, New York, pp. 632.

53. Schoenherr, K. E., Trans SNAME. 40:279-313, 1932.

54. Gilbert, G. B., and Hill, P. G., "Analysis and Testing of Two-dimensional Slot Nozzle Ejectors with Variable Area Mixing Sections," NASA CR-2251, 1973.

55. Kussoy, M. I. and Horstman, C. C., "Documentation of Two- and Three-Dimensional Hypersonic Shock Wave Boundary Layer Interaction Flows," NASA TM 101075, January 1989.

56. Spalart, P. R. and Shur, M., "On the Sensitization of Turbulence Models to Rotation and Curvature," Aerospace Science and Technology, 1997, pp. 297-302.

57. Shur, M. L., Strelets, M. K., Travin, A. K., Spalart, P. R., "Turbulence Modeling in Rotating and Curved Channels: Assessing the Spalart-Shur Correction," AIAA Journal Vol. 38, No. 5, 2000, pp. 784-792.

58. Smirnov, P. E., Menter, F. R., "Sensitization of the SST Turbulence Model to Rotation and Curvature by Applying the Spalart-Shur Correction Term," ASME Journal of Turbomachinery, Vol. 131, October 2009, 041010.

59. Smits, A. J., Young, S. T. B., and Bradshaw, P. "The Effect of Short Regions of High Surface Curvature on Turbulent Boundary Layers," Journal of Fluid Mechanics, Vol. 94, Part 2, 1979, pp. 209-242.

60. Nikuradse, J., "Law of Flow in Rough Pipes," Technical Report 1292, VDI-Forchungsheft 361, Series B, Vol. 4, 1933; NACA TM-1292, 1950.

61. Aupoix, B., and Spalart, P. R., "Extensions of the Spalart-Allmaras Turbulence Model to Account for Wall Roughness," International Journal of Heat and Fluid Flow, Vol. 24, 2003, pp. 454-462.

62. Hellsten, A., and Laine, S., "Extnesion of the k-ω-SST Turbulence Model for Flows Over Rough Surfaces," 22nd Atmospheric Flight Mechanics Conference, 1997.

63. Mills, A., and Hank, X., “On the Skin Friction Coefficient for a Fully Rough Flat Plate,” Journal of Fluids Engineering, September 1983, pp. 364-365.

64. Somers, D. M., “Design and Experimental Results for the S809 Airfoil.” NREL/SR-440-6918, January 1997.

65. Reuss Ramsay, R., Hoffmann, M. J., and Gregorek, G. M., “Effects of Grit Roughness and Pitch Oscillations on the S809 Airfoil.” NREL/TP-442-7817, December 1999.

66. Langtry, R. B., and Menter, F. R., “Transition Modeling for General CFD Applications in Aeronautics,” 43red AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, January 2005.

67. Hou, Y., Wray, T. J., and Agarwal, R. K., “Application of the SST k-ω Transition Model to Flow Past Smooth and Rough Airfoils,” AIAA SciTech, 2017

68. Shimoyama, K., Kawai, S., and Alonso, J. J., “Dynamic adaptive sampling based on Kriging surrogate models for efficient uncertainty quantification,” in 15th AIAA Non-Deterministic Approaches Conference, AIAA 2013–1470

69. Adams, B.M., Bauman, L.E., Bohnhoff, W.J., Dalbey, K.R., Ebeida, M.S., Eddy, J.P., Eldred, M.S., Hough, P.D., Hu, K.T., Jakeman, J.D., Stephens, J.A., Swiler, L.P., Vigil, D.M., and Wildey, T.M., "Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User’s Manual," Sandia Technical Report SAND2014-4633, July 2014. Updated November 2015 (Version 6.3).

70. Schaefer, J., West, T., Hosder, S., Rumsey, C., Carlson, J.-R., and Kleb, W., “Uncertainty Quantification of Turbulence Model Closure Coefficients for Transonic Wall-Bounded Flows,” 22nd AIAA Computational Fluid Dynamics Conference, AIAA Paper 2015-2461, 2015.

Appendix A

WrayAgarwal.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#ifndef WrayAgarwal_H
#define WrayAgarwal_H

#include "RASModel.H"
#include "wallDist.H"

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

namespace Foam
{
namespace incompressible
{
namespace RASModels
{

/*---------------------------------------------------------------------------*\
                      Class WrayAgarwal Declaration
\*---------------------------------------------------------------------------*/

class WrayAgarwal
:
    public RASModel
{

protected:

    // Protected data
```

```
        // Model coefficients

            dimensionedScalar kappa_;
            dimensionedScalar Cmu_;
            dimensionedScalar alpha1_;
            dimensionedScalar alpha2_;
            dimensionedScalar Aplus_;
            dimensionedScalar C1ke_;
                    dimensionedScalar C1kw_;
            dimensionedScalar sigmake_;
            dimensionedScalar sigmakw_;
                    dimensionedScalar C2ke_;
            dimensionedScalar C2kw_;

        // Fields

            volScalarField Rnu_;
            volScalarField nut_;
                    volScalarField Switch_;

            wallDist d_;

    // Protected Member Functions

        tmp<volScalarField> chi() const;

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

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

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

        tmp<volScalarField> blend
        (
            const volScalarField& F1,
            const dimensionedScalar& psi1,
            const dimensionedScalar& psi2
        ) const;

        tmp<volScalarField> sigma(const volScalarField& F1) const
        {
            return blend(F1, sigmakw_, sigmake_);
        }

        tmp<volScalarField> C1(const volScalarField& F1) const
        {
            return blend(F1, C1kw_, C1ke_);
        }

public:

    //- Runtime type information
```

```
TypeName("WrayAgarwal");

// Constructors

    //- Construct from components
    WrayAgarwal
    (
        const volVectorField& U,
        const surfaceScalarField& phi,
        transportModel& transport,
        const word& turbulenceModelName = turbulenceModel::typeName,
        const word& modelName = typeName
    );


//- Destructor
virtual ~WrayAgarwal()
{}


// Member Functions

    //- Return the turbulence viscosity
    virtual tmp<volScalarField> nut() const
    {
        return nut_;
    }

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

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

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

    //- Return the Reynolds stress tensor
    virtual tmp<volSymmTensorField> R() const;

    //- Return the effective stress tensor including the laminar stress
    virtual tmp<volSymmTensorField> devReff() const;

    //- Return the source term for the momentum equation
    virtual tmp<fvVectorMatrix> divDevReff(volVectorField& U) const;

    //- Return the source term for the momentum equation
    virtual tmp<fvVectorMatrix> divDevRhoReff
    (
        const volScalarField& rho,
        volVectorField& U
    ) const;

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

    //- Read RASProperties dictionary
```

```
        virtual bool read();
};


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

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

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

#endif

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

WrayAgarwal.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "WrayAgarwal.H"
#include "addToRunTimeSelectionTable.H"
#include "wallFvPatch.H"

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

namespace Foam
{
namespace incompressible
{
namespace RASModels
{

// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

defineTypeNameAndDebug(WrayAgarwal, 0);
addToRunTimeSelectionTable(RASModel, WrayAgarwal, dictionary);

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

tmp<volScalarField> WrayAgarwal::chi() const
{
    return Rnu_/nu();
}

tmp<volScalarField> WrayAgarwal::fv1(const volScalarField& chi) const
{
    const volScalarField chi3(pow3(chi));
    return chi3/(chi3 + pow3(Aplus_));
}
```

```
tmp<volScalarField> WrayAgarwal::blend
(
       const volScalarField& F1,
       const dimensionedScalar& psi1,
       const dimensionedScalar& psi2
) const
{
       return F1*(psi1 - psi2) + psi2;
}

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

WrayAgarwal::WrayAgarwal
(
    const volVectorField& U,
    const surfaceScalarField& phi,
    transportModel& transport,
    const word& turbulenceModelName,
    const word& modelName
)
:
    RASModel(modelName, U, phi, transport, turbulenceModelName),

    kappa_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "kappa",
            coeffDict_,
            0.41
        )
    ),

    Cmu_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "Cmu",
            coeffDict_,
            0.09
        )
    ),
    alpha1_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "alpha1",
            coeffDict_,
            0.03
        )
    ),
    alpha2_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "alpha2",
```

```
        coeffDict_,
        200.0
    )
),

Aplus_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Aplus",
        coeffDict_,
        8.54
    )
),

C1ke_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "C1ke",
        coeffDict_,
                0.16
    )
),

C1kw_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "C1kw",
        coeffDict_,
        0.0833
    )
),

sigmake_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "sigmake",
        coeffDict_,
        1.0
    )
),

sigmakw_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "sigmakw",
        coeffDict_,
        0.72
    )
),

C2ke_(C1ke_/sqr(kappa_)+sigmake_),
```

```
    C2kw_(C1kw_/sqr(kappa_)+sigmakw_),

    Rnu_
    (
        IOobject
        (
            "Rnu",
            runTime_.timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_
    ),

    nut_
    (
        IOobject
        (
            "nut",
            runTime_.timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_
    ),

    Switch_
    (
        IOobject
        (
            "Switch",
            runTime_.timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("0", dimensionSet(0, 0, 0, 0, 0), 0)
    ),

    d_(mesh_)
{
       Info<< "C2ke: " << C2ke_.value() <<endl;
       Info<< "C2kw: " << C2kw_.value() <<endl;
    printCoeffs();
}


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

tmp<volScalarField> WrayAgarwal::DRnuEff(volScalarField Switch) const
{
    return tmp<volScalarField>
    (
        new volScalarField("DRnuEff", Rnu_*sigma(Switch) + nu())
```

```
    );
}


tmp<volScalarField> WrayAgarwal::k() const
{
    return tmp<volScalarField>
    (
        new volScalarField
        (
            IOobject
            (
                "k",
                runTime_.timeName(),
                mesh_
            ),
            mesh_,
            dimensionedScalar("0", dimensionSet(0, 2, -2, 0, 0), 0)
        )
    );
}


tmp<volScalarField> WrayAgarwal::epsilon() const
{
    return tmp<volScalarField>
    (
        new volScalarField
        (
            IOobject
            (
                "epsilon",
                runTime_.timeName(),
                mesh_
            ),
            mesh_,
            dimensionedScalar("0", dimensionSet(0, 2, -3, 0, 0), 0)
        )
    );
}


tmp<volSymmTensorField> WrayAgarwal::R() const
{
    return tmp<volSymmTensorField>
    (
        new volSymmTensorField
        (
            IOobject
            (
                "R",
                runTime_.timeName(),
                mesh_,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            ((2.0/3.0)*I)*k() - nut()*twoSymm(fvc::grad(U_))
```

```
        )
    );
}


tmp<volSymmTensorField> WrayAgarwal::devReff() const
{
    return tmp<volSymmTensorField>
    (
        new volSymmTensorField
        (
            IOobject
            (
                "devRhoReff",
                runTime_.timeName(),
                mesh_,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
           -nuEff()*dev(twoSymm(fvc::grad(U_)))
        )
    );
}


tmp<fvVectorMatrix> WrayAgarwal::divDevReff(volVectorField& U) const
{
    const volScalarField nuEff_(nuEff());

    return
    (
      - fvm::laplacian(nuEff_, U)
      - fvc::div(nuEff_*dev(T(fvc::grad(U))))
    );
}


tmp<fvVectorMatrix> WrayAgarwal::divDevRhoReff
(
    const volScalarField& rho,
    volVectorField& U
) const
{
    volScalarField muEff("muEff", rho*nuEff());

    return
    (
      - fvm::laplacian(muEff, U)
      - fvc::div(muEff*dev(T(fvc::grad(U))))
    );
}


bool WrayAgarwal::read()
{
    if (RASModel::read())
    {
```

```
            alpha1_.readIfPresent(coeffDict());
            alpha2_.readIfPresent(coeffDict());
            C1kw_.readIfPresent(coeffDict());
            C1ke_.readIfPresent(coeffDict());
            sigmakw_.readIfPresent(coeffDict());
            sigmake_.readIfPresent(coeffDict());
            C2ke_ = (C1ke_/sqr(kappa_)+sigmake_);
            C2kw_ = (C1kw_/sqr(kappa_)+sigmakw_);
            Aplus_.readIfPresent(coeffDict());
            kappa_.readIfPresent(coeffDict());

        return true;
    }
    else
    {
        return false;
    }
}


void WrayAgarwal::correct()
{
    RASModel::correct();

    if (!turbulence_)
    {
        // Re-calculate viscosity
        nut_ = Rnu_;
        nut_.correctBoundaryConditions();

        return;
    }

    if (mesh_.changing())
    {
        d_.correct();
    }


       volScalarField S2(2.0*magSqr(symm(fvc::grad(U_))));
       volScalarField S = sqrt(S2);
       bound(S, dimensionedScalar("0", S.dimensions(), SMALL));
       bound(S2, dimensionedScalar("0", S2.dimensions(), SMALL));

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


       volScalarField Ebb = max(magSqr(fvc::grad(Rnu_)),dimensionedScalar("EbbMin",
dimensionSet(0, 2, -2, 0, 0), SMALL));
       volScalarField Eke = sqr(Rnu_)*magSqr(fvc::grad(S))/S2;
       volScalarField boundEke = 7.0*Ebb*tanh(Eke/(7.0*Ebb));
       volScalarField Ekw = Rnu_/S*(fvc::grad(Rnu_) & fvc::grad(S));
       volScalarField boundEkw = min(Rnu_/S*(fvc::grad(Rnu_) &
fvc::grad(S)),dimensionedScalar("EbbMin", dimensionSet(0, 2, -2, 0, 0), -1*SMALL)) ;

       Switch_ = tanh(pow(1.66*(nu()+Rnu_)/(sqr(kappa_)*S)*1.0/sqr(d_),4.0));
```

```
            const fvPatchList& patches = mesh_.boundary();
            forAll(patches, patchi)
            {
                const fvPatch& curPatch = patches[patchi];

                if (isType<wallFvPatch>(curPatch))
                {
                          Switch_.boundaryField()[patchi] == 1.0;
                   }
            }

       tmp<fvScalarMatrix> RnuEqn
    (
        fvm::ddt(Rnu_)
      + fvm::div(phi_, Rnu_)
      - fvm::laplacian(DRnuEff(Switch_), Rnu_)
     ==
        C1(Switch_)*Rnu_*S
         + C2kw_*Switch_*boundEkw
         - (1.0-Switch_)*(C2ke_)*boundEke
    );

    RnuEqn().relax();
    solve(RnuEqn);
    bound(Rnu_, dimensionedScalar("0", Rnu_.dimensions(), 0.0));
    Rnu_.correctBoundaryConditions();

    // Re-calculate viscosity
    nut_.internalField() = fv1*Rnu_.internalField();
    nut_.correctBoundaryConditions();

}


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

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

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

SpalartAllmarasNoft2.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::incompressible::RASModels::SpalartAllmaras

Group
    grpIcoRASTurbulence

SourceFiles
    SpalartAllmarasNoft2.C

\*---------------------------------------------------------------------------*/

#ifndef SpalartAllmarasNoft2_H
#define SpalartAllmarasNoft2_H

#include "RASModel.H"
#include "wallDist.H"

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

namespace Foam
{
namespace incompressible
{
namespace RASModels
{

/*---------------------------------------------------------------------------*\
                      Class SpalartAllmarasNoft2 Declaration
\*---------------------------------------------------------------------------*/

class SpalartAllmarasNoft2
:
```

```
    public RASModel
{

protected:

    // Protected data

        // Model coefficients

            dimensionedScalar sigmaNut_;
            dimensionedScalar kappa_;

            dimensionedScalar Cb1_;
            dimensionedScalar Cb2_;
            dimensionedScalar Cw1_;
            dimensionedScalar Cw2_;
            dimensionedScalar Cw3_;
            dimensionedScalar Cv1_;

        // Fields

            volScalarField nuTilda_;
            volScalarField nut_;

            wallDist d_;

    // Protected Member Functions

        tmp<volScalarField> chi() const;

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

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

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

public:

    //- Runtime type information
    TypeName("SpalartAllmarasNoft2");


    // Constructors

        //- Construct from components
        SpalartAllmarasNoft2
        (
            const volVectorField& U,
            const surfaceScalarField& phi,
            transportModel& transport,
            const word& turbulenceModelName = turbulenceModel::typeName,
            const word& modelName = typeName
        );
```

```
    //- Destructor
    virtual ~SpalartAllmarasNoft2()
    {}

    // Member Functions

        //- Return the turbulence viscosity
        virtual tmp<volScalarField> nut() const
        {
            return nut_;
        }

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

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

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

        //- Return the Reynolds stress tensor
        virtual tmp<volSymmTensorField> R() const;

        //- Return the effective stress tensor including the laminar stress
        virtual tmp<volSymmTensorField> devReff() const;

        //- Return the source term for the momentum equation
        virtual tmp<fvVectorMatrix> divDevReff(volVectorField& U) const;

        //- Return the source term for the momentum equation
        virtual tmp<fvVectorMatrix> divDevRhoReff
        (
            const volScalarField& rho,
            volVectorField& U
        ) const;

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

        //- Read RASProperties dictionary
        virtual bool read();
};

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

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

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

#endif

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

SpalartAllmarasNoft2.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "SpalartAllmarasNoft2.H"
#include "addToRunTimeSelectionTable.H"

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

namespace Foam
{
namespace incompressible
{
namespace RASModels
{

// * * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //

defineTypeNameAndDebug(SpalartAllmarasNoft2, 0);
addToRunTimeSelectionTable(RASModel, SpalartAllmarasNoft2, dictionary);

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

tmp<volScalarField> SpalartAllmarasNoft2::chi() const
{
    return nuTilda_/nu();
}


tmp<volScalarField> SpalartAllmarasNoft2::fv1(const volScalarField& chi) const
{
    const volScalarField chi3(pow3(chi));
    return chi3/(chi3 + pow3(Cv1_));
```

```
}


tmp<volScalarField> SpalartAllmarasNoft2::fv2
(
    const volScalarField& chi,
    const volScalarField& fv1
) const
{

    return 1.0 - chi/(1.0 + chi*fv1);

}


tmp<volScalarField> SpalartAllmarasNoft2::fw(const volScalarField& Stilda) const
{
    volScalarField r
    (
        min
        (
            nuTilda_
           /(
               max
               (
                   Stilda,
                   dimensionedScalar("SMALL", Stilda.dimensions(), SMALL)
               )
              *sqr(kappa_*d_)
            ),
            scalar(10.0)
        )
    );
    r.boundaryField() == 0.0;

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

    return g*pow((1.0 + pow6(Cw3_))/(pow6(g) + pow6(Cw3_)), 1.0/6.0);
}


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

SpalartAllmarasNoft2::SpalartAllmarasNoft2
(
    const volVectorField& U,
    const surfaceScalarField& phi,
    transportModel& transport,
    const word& turbulenceModelName,
    const word& modelName
)
:
    RASModel(modelName, U, phi, transport, turbulenceModelName),

    sigmaNut_
    (
        dimensioned<scalar>::lookupOrAddToDict
```

```
    (
        "sigmaNut",
        coeffDict_,
        0.66666
    )
),
kappa_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "kappa",
        coeffDict_,
        0.41
    )
),

Cb1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cb1",
        coeffDict_,
        0.1355
    )
),
Cb2_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cb2",
        coeffDict_,
        0.622
    )
),
Cw1_(Cb1_/sqr(kappa_) + (1.0 + Cb2_)/sigmaNut_),
Cw2_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cw2",
        coeffDict_,
        0.3
    )
),
Cw3_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cw3",
        coeffDict_,
        2.0
    )
),
Cv1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
```

```
            "Cv1",
            coeffDict_,
            7.1
        )
    ),

    nuTilda_
    (
        IOobject
        (
            "nuTilda",
            runTime_.timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_
    ),

    nut_
    (
        IOobject
        (
            "nut",
            runTime_.timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_
    ),

    d_(mesh_)
{
    printCoeffs();
}


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

tmp<volScalarField> SpalartAllmarasNoft2::DnuTildaEff() const
{
    return tmp<volScalarField>
    (
        new volScalarField("DnuTildaEff", (nuTilda_ + nu())/sigmaNut_)
    );
}


tmp<volScalarField> SpalartAllmarasNoft2::k() const
{
    WarningIn("tmp<volScalarField> SpalartAllmarasNoft2::k() const")
        << "Turbulence kinetic energy not defined for Spalart-Allmaras model. "
        << "Returning zero field" << endl;

    return tmp<volScalarField>
    (
```

```
        new volScalarField
        (
            IOobject
            (
                "k",
                runTime_.timeName(),
                mesh_
            ),
            mesh_,
            dimensionedScalar("0", dimensionSet(0, 2, -2, 0, 0), 0)
        )
    );
}


tmp<volScalarField> SpalartAllmarasNoft2::epsilon() const
{
    WarningIn("tmp<volScalarField> SpalartAllmarasNoft2::epsilon() const")
        << "Turbulence kinetic energy dissipation rate not defined for "
        << "Spalart-Allmaras model. Returning zero field"
        << endl;

    return tmp<volScalarField>
    (
        new volScalarField
        (
            IOobject
            (
                "epsilon",
                runTime_.timeName(),
                mesh_
            ),
            mesh_,
            dimensionedScalar("0", dimensionSet(0, 2, -3, 0, 0), 0)
        )
    );
}


tmp<volSymmTensorField> SpalartAllmarasNoft2::R() const
{
    return tmp<volSymmTensorField>
    (
        new volSymmTensorField
        (
            IOobject
            (
                "R",
                runTime_.timeName(),
                mesh_,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            ((2.0/3.0)*I)*k() - nut()*twoSymm(fvc::grad(U_))
        )
    );
}
```

```
tmp<volSymmTensorField> SpalartAllmarasNoft2::devReff() const
{
    return tmp<volSymmTensorField>
    (
        new volSymmTensorField
        (
            IOobject
            (
                "devRhoReff",
                runTime_.timeName(),
                mesh_,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
           -nuEff()*dev(twoSymm(fvc::grad(U_)))
        )
    );
}


tmp<fvVectorMatrix> SpalartAllmarasNoft2::divDevReff(volVectorField& U) const
{
    const volScalarField nuEff_(nuEff());

    return
    (
      - fvm::laplacian(nuEff_, U)
      - fvc::div(nuEff_*dev(T(fvc::grad(U))))
    );
}


tmp<fvVectorMatrix> SpalartAllmarasNoft2::divDevRhoReff
(
    const volScalarField& rho,
    volVectorField& U
) const
{
    volScalarField muEff("muEff", rho*nuEff());

    return
    (
      - fvm::laplacian(muEff, U)
      - fvc::div(muEff*dev(T(fvc::grad(U))))
    );
}


bool SpalartAllmarasNoft2::read()
{
    if (RASModel::read())
    {
        sigmaNut_.readIfPresent(coeffDict());
        kappa_.readIfPresent(coeffDict());
```

```
        Cb1_.readIfPresent(coeffDict());
        Cb2_.readIfPresent(coeffDict());
        Cw1_ = Cb1_/sqr(kappa_) + (1.0 + Cb2_)/sigmaNut_;
        Cw2_.readIfPresent(coeffDict());
        Cw3_.readIfPresent(coeffDict());
        Cv1_.readIfPresent(coeffDict());

        return true;
    }
    else
    {
        return false;
    }
}


void SpalartAllmarasNoft2::correct()
{
    RASModel::correct();

    if (!turbulence_)
    {
        // Re-calculate viscosity
        nut_ = nuTilda_*fv1(this->chi());
        nut_.correctBoundaryConditions();

        return;
    }

    if (mesh_.changing())
    {
        d_.correct();
    }

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

    const volScalarField Stilda
    (
        sqrt(2.0)*mag(skew(fvc::grad(U_)))
      + fv2(chi, fv1)*nuTilda_/sqr(kappa_*d_)
    );

    tmp<fvScalarMatrix> nuTildaEqn
    (
        fvm::ddt(nuTilda_)
      + fvm::div(phi_, nuTilda_)
      - fvm::laplacian(DnuTildaEff(), nuTilda_)
      - Cb2_/sigmaNut_*magSqr(fvc::grad(nuTilda_))
     ==
        Cb1_*Stilda*nuTilda_
      - fvm::Sp(Cw1_*fw(Stilda)*nuTilda_/sqr(d_), nuTilda_)
    );

    nuTildaEqn().relax();
    solve(nuTildaEqn);
    bound(nuTilda_, dimensionedScalar("0", nuTilda_.dimensions(), 0.0));
```

```
    nuTilda_.correctBoundaryConditions();

    // Re-calculate viscosity
    nut_.internalField() = fv1*nuTilda_.internalField();
    nut_.correctBoundaryConditions();
}


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

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

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

kOmegaSST2003.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2012 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::incompressible::RASModels::kOmegaSST

SourceFiles
    kOmegaSST2003.C

\*---------------------------------------------------------------------------*/

#ifndef kOmegaSST2003_H
#define kOmegaSST2003_H

#include "RASModel.H"
#include "wallDist.H"

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

namespace Foam
{
namespace incompressible
{
namespace RASModels
{

/*---------------------------------------------------------------------------*\
                           Class kOmegaSST2003 Declaration
\*---------------------------------------------------------------------------*/

class kOmegaSST2003
:
    public RASModel
{
```

```
protected:

    // Protected data

        // Model coefficients

            dimensionedScalar kappa_;
            dimensionedScalar sigmak1_;
            dimensionedScalar sigmak2_;
            dimensionedScalar sigmaOmega1_;
            dimensionedScalar sigmaOmega2_;
            dimensionedScalar Prt_;
            dimensionedScalar beta1_;
            dimensionedScalar beta2_;
            dimensionedScalar betaStar_;
            dimensionedScalar gamma1_;
            dimensionedScalar gamma2_;
            dimensionedScalar a1_;

        //- Wall distance
        //  Note: different to wall distance in parent RASModel
              wallDist y_;

        // Fields

            volScalarField k_;
            volScalarField omega_;
            volScalarField nut_;
                    volScalarField F1_;
                    volScalarField F2_;


    // Protected Member Functions

        tmp<volScalarField> F1(const volScalarField& CDkOmega) const;
        tmp<volScalarField> F2() const;

        tmp<volScalarField> blend
        (
            const volScalarField& F1,
            const dimensionedScalar& psi1,
            const dimensionedScalar& psi2
        ) const
        {
            return F1*(psi1 - psi2) + psi2;
        }

        tmp<volScalarField> sigmak(const volScalarField& F1) const
        {
            return blend(F1, sigmak1_, sigmak2_);
        }

        tmp<volScalarField> sigmaOmega(const volScalarField& F1) const
        {
            return blend(F1, sigmaOmega1_, sigmaOmega2_);
        }
```

```
        tmp<volScalarField> beta(const volScalarField& F1) const
        {
            return blend(F1, beta1_, beta2_);
        }

        tmp<volScalarField> gamma(const volScalarField& F1) const
        {
            return blend(F1, gamma1_, gamma2_);
        }


public:

    //- Runtime type information
    TypeName("kOmegaSST2003");


    // Constructors

        //- Construct from components
        kOmegaSST2003
        (
            const volVectorField& U,
            const surfaceScalarField& phi,
            transportModel& transport,
            const word& turbulenceModelName = turbulenceModel::typeName,
            const word& modelName = typeName
        );


    //- Destructor
    virtual ~kOmegaSST2003()
    {}


    // Member Functions

        //- Return the turbulence viscosity
        virtual tmp<volScalarField> nut() const
        {
            return nut_;
        }

        //- Return the effective diffusivity for k
        tmp<volScalarField> DkEff(const volScalarField& F1) const
        {
            return tmp<volScalarField>
            (
                new volScalarField("DkEff", sigmak(F1)*nut_ + nu())
            );
        }

        //- Return the effective diffusivity for omega
        tmp<volScalarField> DomegaEff(const volScalarField& F1) const
        {
            return tmp<volScalarField>
            (
```

```
            new volScalarField("DomegaEff", sigmaOmega(F1)*nut_ + nu())
        );
    }

    //- Return the turbulence kinetic energy
    virtual tmp<volScalarField> k() const
    {
        return k_;
    }

    //- Return the turbulence specific dissipation rate
    virtual tmp<volScalarField> omega() const
    {
        return omega_;
    }

    //- Return the turbulence kinetic energy dissipation rate
    virtual tmp<volScalarField> epsilon() const
    {
        return tmp<volScalarField>
        (
            new volScalarField
            (
                IOobject
                (
                    "epsilon",
                    mesh_.time().timeName(),
                    mesh_
                ),
                betaStar_*k_*omega_,
                omega_.boundaryField().types()
            )
        );
    }

    //- Return the Reynolds stress tensor
    virtual tmp<volSymmTensorField> R() const;

    //- Return the effective stress tensor including the laminar stress
    virtual tmp<volSymmTensorField> devReff() const;

    //- Return the source term for the momentum equation
    virtual tmp<fvVectorMatrix> divDevReff(volVectorField& U) const;

    //- Return the source term for the momentum equation
    virtual tmp<fvVectorMatrix> divDevRhoReff
    (
        const volScalarField& rho,
        volVectorField& U
    ) const;

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

    //- Read RASProperties dictionary
    virtual bool read();
};
```

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

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

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

#endif

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

kOmegaSST2003.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "kOmegaSST2003.H"
#include "addToRunTimeSelectionTable.H"
#include "wallFvPatch.H"

#include "backwardsCompatibilityWallFunctions.H"

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

namespace Foam
{
namespace incompressible
{
namespace RASModels
{

// * * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //

defineTypeNameAndDebug(kOmegaSST2003, 0);
addToRunTimeSelectionTable(RASModel, kOmegaSST2003, dictionary);

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

tmp<volScalarField> kOmegaSST2003::F1(const volScalarField& CDkOmega) const
{
    tmp<volScalarField> arg1 = min
    (
        max
        (
            (scalar(1)/betaStar_)*sqrt(k_)/(omega_*y_),
            scalar(500)*nu()/(sqr(y_)*omega_)
```

```
        ),
        (4.0*sigmaOmega2_)*k_/(CDkOmega*sqr(y_))
    );

    return tanh(pow4(arg1));
}

tmp<volScalarField> kOmegaSST2003::F2() const
{
    tmp<volScalarField> arg2 = max
    (
        (scalar(2)/betaStar_)*sqrt(k_)/(omega_*y_),
        scalar(500)*nu()/(sqr(y_)*omega_)
    );

    return tanh(sqr(arg2));
}


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

kOmegaSST2003::kOmegaSST2003
(
    const volVectorField& U,
    const surfaceScalarField& phi,
    transportModel& transport,
    const word& turbulenceModelName,
    const word& modelName
)
:
    RASModel(modelName, U, phi, transport, turbulenceModelName),

    kappa_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "kappa",
            coeffDict_,
            0.41
        )
    ),
    sigmak1_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "sigmak1",
            coeffDict_,
            0.85
        )
    ),
    sigmak2_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "sigmak2",
            coeffDict_,
            1.0
```

```
    )
),
sigmaOmega1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "sigmaOmega1",
        coeffDict_,
        0.5
    )
),
sigmaOmega2_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "sigmaOmega2",
        coeffDict_,
        0.856
    )
),
Prt_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Prt",
        coeffDict_,
        0.9//1.0
    )
),
beta1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "beta1",
        coeffDict_,
        0.075
    )
),
beta2_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "beta2",
        coeffDict_,
        0.0828
    )
),
betaStar_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "betaStar",
        coeffDict_,
        0.09
    )
),
gamma1_
```

```
(
    //dimensioned<scalar>::lookupOrAddToDict
    //(
    //    "gamma1",
    //    coeffDict_,
        beta1_/betaStar_-sigmaOmega1_*kappa_*kappa_/sqrt(betaStar_)//0.55556
    //)
),
gamma2_
(
    //dimensioned<scalar>::lookupOrAddToDict
    //(
    //    "gamma2",
    //    coeffDict_,
        beta2_/betaStar_-sigmaOmega2_*kappa_*kappa_/sqrt(betaStar_)//0.44
    //)
),
a1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "a1",
        coeffDict_,
        0.31
    )
),

y_(mesh_),

k_
(
    IOobject
    (
        "k",
        runTime_.timeName(),
        mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_
),
omega_
(
    IOobject
    (
        "omega",
        runTime_.timeName(),
        mesh_,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh_
),

nut_
(
    IOobject
```

```
        (
            "nut",
            runTime_.timeName(),
            mesh_,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_
    ),

    F1_
    (
        IOobject
        (
            "F1",
            runTime_.timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("1.0", dimensionSet(0, 0, 0, 0, 0), 1.0)
    ),
    F2_
    (
        IOobject
        (
            "F2",
            runTime_.timeName(),
            mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        mesh_,
        dimensionedScalar("1.0", dimensionSet(0, 0, 0, 0, 0), 1.0)
    )

{
    bound(k_, kMin_);
    bound(omega_, omegaMin_);

    nut_ =
    (
        a1_*k_
      / max
        (
            a1_*omega_,
            F2()*sqrt(2.0)*mag(skew(fvc::grad(U_)))
        )
    );
    nut_.correctBoundaryConditions();

    printCoeffs();

       const fvPatchList& patches = mesh_.boundary();
    forAll(patches, patchi)
    {
```

```
        const fvPatch& curPatch = patches[patchi];

        if (isType<wallFvPatch>(curPatch))
        {
                    if
(!isType<zeroGradientFvPatchScalarField>(omega_.boundaryField()[patchi]))
                    {
                    FatalErrorIn("wall-function evaluation")
              << omega_.boundaryField()[patchi].type()
              << " is the wrong omega patchField type for wall-functions on patch "
              << curPatch.name() << nl
              << "    should be zeroGradient"
              << exit(FatalError);
                    }
              }
       }
}


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

tmp<volSymmTensorField> kOmegaSST2003::R() const
{
    return tmp<volSymmTensorField>
    (
        new volSymmTensorField
        (
            IOobject
            (
                "R",
                runTime_.timeName(),
                mesh_,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            ((2.0/3.0)*I)*k_ - nut_*twoSymm(fvc::grad(U_)),
            k_.boundaryField().types()
        )
    );
}


tmp<volSymmTensorField> kOmegaSST2003::devReff() const
{
    return tmp<volSymmTensorField>
    (
        new volSymmTensorField
        (
            IOobject
            (
                "devRhoReff",
                runTime_.timeName(),
                mesh_,
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
           -nuEff()*dev(twoSymm(fvc::grad(U_)))
```

```
        )
    );
}


tmp<fvVectorMatrix> kOmegaSST2003::divDevReff(volVectorField& U) const
{
    return
    (
      - fvm::laplacian(nuEff(), U)
      - fvc::div(nuEff()*dev(T(fvc::grad(U))))
    );
}


tmp<fvVectorMatrix> kOmegaSST2003::divDevRhoReff
(
    const volScalarField& rho,
    volVectorField& U
) const
{
    volScalarField muEff("muEff", rho*nuEff());

    return
    (
      - fvm::laplacian(muEff, U)
      - fvc::div(muEff*dev(T(fvc::grad(U))))
    );
}


bool kOmegaSST2003::read()
{
    if (RASModel::read())
    {
        kappa_.readIfPresent(coeffDict());
        sigmak1_.readIfPresent(coeffDict());
        sigmak2_.readIfPresent(coeffDict());
        sigmaOmega1_.readIfPresent(coeffDict());
        sigmaOmega2_.readIfPresent(coeffDict());
        Prt_.readIfPresent(coeffDict());
        beta1_.readIfPresent(coeffDict());
        beta2_.readIfPresent(coeffDict());
        betaStar_.readIfPresent(coeffDict());
       // gamma1_.readIfPresent(coeffDict());
       // gamma2_.readIfPresent(coeffDict());
        a1_.readIfPresent(coeffDict());

        return true;
    }
    else
    {
        return false;
    }
}
```

```
void kOmegaSST2003::correct()
{
    RASModel::correct();

    if (!turbulence_)
    {
        return;
    }

    if (mesh_.changing())
    {
        y_.correct();
        bound(k_, kMin_);
        bound(omega_, omegaMin_);
    }

    const volScalarField S2(2.0*magSqr(symm(fvc::grad(U_))));
    const volScalarField W2(2.0*magSqr(skew(fvc::grad(U_))));
    volScalarField G(GName(), nut_*S2);

       #include "calcWallOmega.H"
    // Update omega and G at the wall
    //omega_.boundaryField().updateCoeffs();

    const volScalarField CDkOmega
    (
        max((2.0*sigmaOmega2_)*(fvc::grad(k_) &
fvc::grad(omega_))/omega_,dimensionedScalar("1.0e-10", dimless/sqr(dimTime), 1.0e-20))
    );

    const volScalarField F1(this->F1(CDkOmega));
       F1_ = F1;

    // Turbulent frequency equation
    tmp<fvScalarMatrix> omegaEqn
    (
        fvm::ddt(omega_)
      + fvm::div(phi_, omega_)
      + fvm::SuSp(-fvc::div(phi_), omega_)
      - fvm::laplacian(DomegaEff(F1), omega_)
     ==
        gamma(F1)*min(G, 10.0*betaStar_*k_*omega_)/nut_
      - fvm::Sp(beta(F1)*omega_, omega_)
      - fvm::SuSp
        (
            (F1 - scalar(1))*CDkOmega/omega_,
            omega_
        )
    );

    omegaEqn().relax();

       #include "setWallOmega.H"
    //omegaEqn().boundaryManipulate(omega_.boundaryField());

    solve(omegaEqn);
    bound(omega_, omegaMin_);
```

```
    // Turbulent kinetic energy equation
    tmp<fvScalarMatrix> kEqn
    (
        fvm::ddt(k_)
      + fvm::div(phi_, k_)
      + fvm::SuSp(-fvc::div(phi_), k_)
      - fvm::laplacian(DkEff(F1), k_)
     ==
        min(G, 10.0*betaStar_*k_*omega_)
      - fvm::Sp(betaStar_*omega_, k_)
    );

    kEqn().relax();
    solve(kEqn);
    bound(k_, kMin_);

    // Re-calculate viscosity
    nut_ = a1_*k_/max(a1_*omega_, F2()*sqrt(S2));
    //nut_.correctBoundaryConditions();
}


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

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

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

calcWallOmega.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

\*---------------------------------------------------------------------------*/

{
    labelList cellBoundaryFaceCount(omega_.size(), 0);

    scalar Cmu25 = pow(betaStar_.value(), 0.25);

    const fvPatchList& patches = mesh_.boundary();

    //- Initialise the near-wall omega and G fields to zero
    forAll(patches, patchi)
    {
        const fvPatch& curPatch = patches[patchi];

        if (isType<wallFvPatch>(curPatch))
        {
            forAll(curPatch, facei)
            {
                label faceCelli = curPatch.faceCells()[facei];

                omega_[faceCelli] = 0.0;
                G[faceCelli] = 0.0;
            }
        }
    }

    //- Accumulate the wall face contributions to omega and G
    //  Increment cellBoundaryFaceCount for each face for averaging
    forAll(patches, patchi)
    {
        const fvPatch& curPatch = patches[patchi];
```

```
        if (isType<wallFvPatch>(curPatch))
        {
//#           include "checkkOmega_LowRePatchFieldTypes.H"

            //const scalarField& nuw = nu().boundaryField()[patchi];
              const tmp<volScalarField> tnu = nu();
              const volScalarField& nu = tnu();
              const scalarField& nuw = nu.boundaryField()[patchi];
            const scalarField& nutw = nut_.boundaryField()[patchi];

            scalarField magFaceGradU =
                mag(U_.boundaryField()[patchi].snGrad());

            forAll(curPatch, facei)
            {
                label faceCelli = curPatch.faceCells()[facei];

                // For corner cells (with two boundary or more faces),
                // omega and G in the near-wall cell are calculated
                // as an average

                cellBoundaryFaceCount[faceCelli]++;

                omega_[faceCelli] += scalar(60.0)*nuw[facei]
                    /(beta1_.value()*sqr(y_[faceCelli]));

                G[faceCelli] +=
                    (nutw[facei] + nuw[facei])*magFaceGradU[facei]
                    *Cmu25*sqrt(k_[faceCelli])/(kappa_.value()*y_[faceCelli]);
            }
        }
    }


    // Perform the averaging

    forAll(patches, patchi)
    {
        const fvPatch& curPatch = patches[patchi];

        if (isType<wallFvPatch>(curPatch))
        {
            forAll(curPatch, facei)
            {
                label faceCelli = curPatch.faceCells()[facei];

                omega_[faceCelli] /= cellBoundaryFaceCount[faceCelli];
                G[faceCelli] /= cellBoundaryFaceCount[faceCelli];
            }
        }
    }
}


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

setWallOmega.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright held by original author
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software; you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by the
    Free Software Foundation; either version 2 of the License, or (at your
    option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM; if not, write to the Free Software Foundation,
    Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

\*---------------------------------------------------------------------------*/

{
    const fvPatchList& patches = mesh_.boundary();

    forAll(patches, patchi)
    {
        const fvPatch& p = patches[patchi];

        if (isType<wallFvPatch>(p))
        {
            omegaEqn().setValues
            (
                p.faceCells(),
                omega_.boundaryField()[patchi].patchInternalField()
            );
        }
    }
}

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