Spring 5-21-2021

Development of a New Transitional Flow Model Integrating the One-Equation Wray-Agarwal Turbulence Model with an Algebraic Intermittency Transport Term

YAN XUE
Washington University in St. Louis

Washington University in St. Louis

James McKelvey School of Engineering

Department of Mechanical Engineering & Material Science

Thesis Examination Committee:

Ramesh Agarwal, Chair

David Peters

Swami Karunamoorthy

Development of a New Transitional Flow Model Integrating the One-Equation Wray-Agarwal Turbulence Model with an Algebraic Intermittency Transport Term

By

Yan Xue

A thesis presented to the McKelvey School of Engineering of Washington University in St. Louis in partial fulfillment of the requirements for the degree of Master of Science

May 2021

St. Louis, Missouri

Dedication

I gracefully dedicate this master's thesis with my love and sincerity to all people who once helped and inspired me.

Acknowledgements

I would like to express my deepest appreciation to my researcher advisor, Dr. Ramesh Agarwal for his generous assistance and helpful suggestions. Without his professional and continuous help and instruction, it would have never been possible to be complete this thesis.

I also appreciate the effort and time devoted by the committee members, Dr. David Peters and Dr. Swami Karunamoorthy to read the thesis and attend the defense.

I also want to thank the former CFD Lab members, Wenjie Shang and Tianshu Wen, who helped me and provided me a detailed introduction on using the OpenFOAM software when I was new in CFD Lab.

Finally, I would like to thank my mother for her continuous love and financial support. Her constant support always helped me to overcome all difficulties in my life.

Yan Xue

May 2021

Table of Contents

List of Tables

List of Figures

Abstract

Development of a New Transitional Flow Model Integrating the One-Equation Wray-Agarwal Turbulence Model with an Algebraic Intermittency Transport Term

By

Yan Xue

Master of Science in Mechanical Engineering

Washington University in St. Louis, 2021

Research Advisor: Professor Ramesh K. Agarwal

In last five decades, Computational Fluid Dynamics (CFD) has become a mature technology and CFD solvers are now routinely employed in the analysis and design of automobiles, airplanes and in many other industrial applications. In industrial applications, the widely used method is to solve the Reynolds Averaged Navier-Stokes equations (RANS) in conjunction with a turbulence model since the other high-fidelity methods namely the Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS) are highly compute intensive as well as expensive for 3D complex flows at high Reynolds numbers although LES and DNS methods can provide better accuracy. A large number of turbulence models have been proposed in past 50 years; most of them are linear eddy viscosity models based on Boussinesq's hypothesis. While there has been significant progress in turbulence modeling for prediction of

fully turbulent flows, progress has been relatively limited in modeling of laminar to turbulent transition because of flow intermittency in the transitional flow region. The purpose of this research is to establish an accurate and efficient one-equation transition model to predict the transition in fluid flow. The development of new one-equation transition model is based on the recently proposed one-equation linear eddy viscosity turbulence model, called the Wray-Agarwal (WA) model by incorporating an algebraic intermittency term in the model for predicting transition. The new transition model is designated as WA-T. The model is applied to compute benchmark ERCOFTAC test cases, namely T3A, T3B and T3A- for transitional flow over a flat plate with different free stream turbulent intensity and zero pressure gradient, and T3C series cases of flow over a flat plate under pressure gradient and different free stream turbulence intensity, and transitional flow past S809 airfoil. The computed results show an excellent agreement with experimental data.

Chapter 1: Introduction

1.1 Background and Motivation

Turbulent flows are the most prevalent flow phenomenon in everyday surroundings. Most industrial and consumer products involve fluids. However, the prediction of turbulent flows has remained one of the most difficult problems in classical physics due to random variations in flow variables with an infinite number of lengths and time scales. Since the advent of modern computers, Computational Fluid Dynamics (CFD) tools developed over the past fifty years have made great progress in solving the Navier-Stokes equations. By decomposing each flow quantity into a mean and a fluctuating part, the Reynolds-Averaged Navier-Stoke (RANS) equations can be obtained from the Navier-Stokes equation by substituting the decomposed flow variables into the Navier-Stokes equation and averaging them over time. Even though the instantaneous flow properties in turbulent flow are very sensitive to initial conditions and boundary conditions, the time averaged properties are still quite regular on the length and time scale of interest. However, the RANS equations have closure problem because of the appearance of unknown turbulent stress during averaging. The turbulent stresses need to be modeled by a turbulence model. This has resulted in an area of study called “Turbulence Modeling.” Because of modeling of turbulent stresses, RANS predictions are not very accurate for many turbulent flows with complex flow features. For more accurate simulations, two methods called the Direct Numerical Simulation (DNS) (which requires no modeling) and the Large Eddy Simulation (LES) (which requires modeling of sub-grid scale (SGS) eddies) have been developed which are more accurate than RANS; however they are highly compute intensive and costly. Therefore, due to the high

computational cost and CPU requirements of DNS and LES, both of them can currently only be used for very simple applications and low Reynolds numbers.

While there has been significant progress in turbulence modeling for prediction of fully turbulent flows, progress has been relatively limited in modeling of laminar to turbulent transition because of flow intermittency in the transitional flow region. Accurate prediction of laminar to turbulent transition has remained a challenging problem in CFD for decades. Transitional flows frequently occur in many industrial applications such as flow past airplanes, automobiles, wind turbines and turbomachines to name a few. The relatively moderate Reynolds number can result in flow past a body in all three flow regimes --- laminar, transitional and turbulent which makes its prediction very difficult and challenging. Due to large uncertainty in the prediction of transition location, the current transition prediction approaches based on empirical methods need to be improved. Accurate prediction of transition will greatly help the design of airfoils that are widely used in airplanes, turbomachinery and wind turbines. The purpose of this research is to establish an accurate and efficient one-equation transition model to predict the transition in fluid flow.

1.2 Scope of the Thesis

The goal of this work is to develop a new one-equation transition model, WA-T, based on the recently proposed one-equation linear eddy viscosity turbulence model called the Wray-Agarwal (WA) model by incorporating an algebraic intermittency term in the model for predicting transition.

A brief summary of each chapter and its contents are given below:

Chapter 2: Introduction to Turbulence and Transition Modeling: This chapter introduces turbulent flows and turbulence and transition modeling. The main approach used in Computational Fluid Dynamics (CFD), namely the solution of Reynolds Averaged Navier-Stokes (RANS) equations is briefly described. The linear eddy viscosity turbulence models are explained and the one equation Wray-Agarwal (WA2018) turbulence model and algebraic intermittency term are briefly described.

Chapter 3: Development of a New Algebraic Transitional Flow Model: This chapter describes an extension of one-equation WA2018 model by developing and including an algebraic transitional flow model. The extension is accomplished by coupling the one-equation WA model with an algebraic intermittency γ term. The new model is validated by 2D benchmark cases.

Chapter 4: Summary: This chapter provides the summary of all the results and the overall potential of the new transition model.

Chapter 2: Introduction to Turbulence and Transition Modeling

2.1 Introduction to Turbulent Flow and Turbulence Modeling

Turbulent flows can be observed in our everyday surroundings, such as heavy smoke from chimneys or water flowing down from rivers. Turbulence occurs in almost all practical engineering problems. The fluid motion is governed by the conservation of momentum equations, called the Navier-stokes equations which employ the Stokes' hypothesis for the viscous stresses to relate them to the strain tensor via a dynamic viscosity; these equations in principle are applicable to all three categories of flows – laminar, turbulent and transitional. In laminar flow each fluid element moves along a smooth path without fluctuations; however very few flows are laminar in nature and in various industrial products. On the other hand, "turbulent fluid motion is an irregular condition of flow in which various flow quantities show a random variation with time and space coordinates so that statistically distinct average values can be discerned. Turbulence has a wide range of scales." By Fourier analysis of the time history of turbulence, analysis shows that the time and length scale of turbulence can be represented by frequency and wavelength, respectively. Compared to laminar flow, the turbulent flow has the characteristics of instability, nonlinearity, vortex stretching, and violent mixing. Turbulence is a continuous phenomenon. Due to its complex nature and characteristics, turbulence is still an unsolved problem in classical physics.

In past few decades, computing power has made tremendous progress, and now the Navier-Stokes equations can be solved numerically. Three calculation methods have been developed to solve Navier-Stokes equations, namely the Direct Numerical Simulation (DNS), Large Eddy Simulation (LES), and Reynold-Averaged Navier-Stokes (RANS) equations. DNS

does not require any modeling, but because computing power is limited even on the largest supercomputers, it is currently limited to low Reynolds numbers and simple geometric shapes. LES only needs to model small eddies and is relatively less compute intensive compared to DNS and therefore it is being increasing used more, but it is still very compute intensive and expensive for industrial applications. At present, due to relatively less requirements of computing resources, the RANS equation are most widely used for calculation of industrial turbulent flows providing acceptable solution accuracy. In RANS equations, the turbulent stresses are unknown and must be modeled. This is called the closure problem. Reynolds Stresses require modeling; the modeling of the Reynolds stress tensor is called turbulence modeling.

2.2 Turbulence and Transition Modeling

2.2.1 Introduction

Although significant progress has been made in turbulence modeling to predict fully turbulent flows, the progress in prediction of laminar to turbulent transition has been relatively limited due to flow intermittency in the transition region. For decades, accurate prediction of transition from laminar flow to turbulent flow has been a challenging problem in CFD. Transitional flows are common in many industrial applications, such as airplanes, automobiles, wind turbines, and fluids flowing through turbines. At relatively moderate Reynolds number, the blood flow in the cardiovascular system of human body goes through regions of: laminar flow, transition and turbulent flow, which makes it very difficult predict. Due to large uncertainty in predicting the transition position, it is necessary to improve the current transition prediction methods which are based on empirical methods. Accurate transition prediction is needed for good airfoil/wing design for in aircraft, turbomachinery, and wind turbine applications.

The most widely used model for computing transitional flow in industrial applications is the Langtry-Menter four-equation transitional model, which is also known as γ-Re_θ-SST model developed by Menter et al. [1]. The model couples Menter's SST k-ω two-equation turbulence model with an additional intermittency equation "γ" and a "Re_θ" transport equation. To remove the lack of Galilean invariance and to reduce the computational cost, a three-equation γ-SST model was developed by Menter et al. which is independent of the Re_θ equation [2].

The next two subsections describe the one equation wall-distance free Wray-Agarwal (WA2018) one-equation turbulence model and an algebraic intermittency term used for predicting the transitional flow.

2.2.2 Algebraic Intermittency Transport Term in SA Model

The basic experimental correlations of the original γ-$Re\theta$ model were not released until 2009; therefore many researchers tried to create alternative formulas for these correlations. Bas and Cakmakcioglu [3] introduced an algebraic or zero-equation model (the BC model), which successfully reproduced the results of the two-equation and three-equation models. At the same time, Cakmakcioglu et al. [3] proposed a one-equation γ-model, which is a simplified version of the early γ-Re_θ model with similar quality of results as the original model with less number of transport equations. Kubacki et al. [4] also proposed a new algebraic transition model; the results once again proved that as long as the physics is correctly modeled, the same predictive ability can be obtained compared to the models with several transport equations like γ-$Re\theta$ SST model.

In the BC transition model developed by Bas and Cakmakcioglu [3], by using the new algebraic intermittency function formulation instead of the intermittency transport equation, the use of vorticity Reynolds number to trigger the transition is avoided, thereby avoiding the use of

non-local information. In this way, the intermittency equation is directly fed into the production term of the one-equation SA turbulence model.

The SA-BC transition model is coupled with the SA-noft2 model as follows:

$$\frac{\partial \tilde{v}}{\partial t} + u_j \frac{\partial \tilde{v}}{\partial x_j} = \gamma_{BC} c_{b1} \tilde{S} \tilde{v} - c_{w1} f_w \left(\frac{\tilde{v}}{d}\right)^2 + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left((\tilde{v} + v)\frac{\partial \tilde{v}}{\partial x_j}\right) + c_{b2}\frac{\partial \tilde{v}}{\partial x_j}\frac{\partial \tilde{v}}{\partial x_j}\right] \quad (2.1)$$

The γ intermittency function is defined in the Spalart-Allmaras one-equation BC transition model as:

$$\gamma = 1 - \exp\left(-\sqrt{Term_1} - \sqrt{Term_2}\right) \quad (2.2)$$

$Term_1$ is given by:

$$Term_1 = \frac{\max(Re_\theta - Re_{\theta c}, 0.0)}{\chi_1 Re_{\theta c}} \quad (2.3)$$

where

$$Re_\theta = \frac{Re_v}{2.193} \text{ and } Re_v = \frac{\rho d^2}{\mu}\Omega \quad (2.4)$$

and

$$Re_{\theta c} = 803.73(Tu_\infty + 0.6067)^{-1.027} \quad (2.5)$$

$Term_2$ is given by:

$$Term_2 = \max\left(\frac{v_t}{v}\chi_2, 0.0\right) \quad (2.6)$$

χ_1 and χ_2 are calibrated constants. χ_1 and χ_2 are given by:

$$\chi_1 = 0.02 \text{ and } \chi_2 = 50$$

The freestream kinematic eddy viscosity $\tilde{v}$ is set to

$$\tilde{v}_{farfield} = 0.015 v_\infty : to : 0.025 v_\infty \quad (2.7)$$

(or 0.005 times the original SA boundary condition in far field)

2.2.1 Wall-Distance Free Wray-Agarwal (WA2018) Turbulence Model

The WA one equation turbulence model was first proposed by Wray and Agarwal; it is a one-equation linear eddy viscosity model; "WA" in the model stands for the two authors' last names. The latest version of WA model is WA2018 developed by Han et al [5]. The WA2018 is a wall-distance-free model, which has been shown to improve the accuracy near curved surfaces. The model solves for the variable $R=k/\omega$, and its transport equation is given as follows:

$$\frac{\partial R}{\partial t}+\frac{\partial u_j R}{\partial x_j}=\frac{\partial}{\partial x_j}\left[(\sigma_R R+\nu)\frac{\partial R}{\partial x_j}\right]+C_1 RS+f_1 C_{2kw}\frac{R}{S}\frac{\partial R}{\partial x_j}\frac{\partial S}{\partial x_j}$$
$$-(1-f_1)min\left[C_{2k\varepsilon}R^2\left(\frac{\frac{\partial S}{\partial x_j}\frac{\partial S}{\partial x_j}}{S^2}\right),C_m\frac{\partial R}{\partial x_j}\frac{\partial R}{\partial x_j}\right] \qquad (2.8)$$

The value of γ will be 0 in laminar flow, and it turns into 1 in fully turbulent flow. In Wray-Agarwal (WA2018) model, the eddy viscosity is given by:

$$\mu_t=\rho f_\mu R \qquad (2.9)$$

In WA model, the damping function f_μ is designed to account for wall blocking effect, which is given by:

$$f_\mu=\frac{\chi^3}{\chi^3+C_w^3},\qquad \chi=\frac{R}{\nu} \qquad (2.10)$$

where ν is the kinematic viscosity and $R=k/\omega$. S and W are the mean strain rate and mean vorticity given by:

$$S=\sqrt{2S_{ij}S_{ij}},\qquad S_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right) \qquad (2.11)$$

$$W=\sqrt{2W_{ij}W_{ij}},\qquad W_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}-\frac{\partial u_j}{\partial x_i}\right) \qquad (2.12)$$

The WA model combines the features of standard *k-ω* and *k-ε* models. The switching function f_1 triggers the behavior of a one-equation *k-ω* or a one-equation *k-ε* model. The switching function f_1 in WA model is given by:

$$f_1 = \tanh(arg_1^4), \quad arg_1 = \frac{\nu + R}{2} \frac{\eta^2}{C_\mu k\omega} \tag{2.13}$$

where

$$k = \frac{\nu_T S}{\sqrt{C_\mu}} \tag{2.14}$$

$$\omega = \frac{S}{\sqrt{C_\mu}} \tag{2.15}$$

$$\eta = S \max\left(1, \left|\frac{W}{S}\right|\right) \tag{2.16}$$

The model constants are:

$$C_{1k\omega} = 0.0829, \quad C_{1k\varepsilon} = 0.1284$$

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

$$\sigma_{kw} = 0.72, \quad \sigma_{k\varepsilon} = 1.0$$

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

$$C_{2k\omega} = \frac{C_{1k\omega}}{\kappa^2} + \sigma_{kw}, \quad C_{2k\varepsilon} = \frac{C_{1k\varepsilon}}{\kappa^2} + \sigma_{k\varepsilon}$$

$$\kappa = 0.41, \quad C_\omega = 8.54$$

$$C_\mu = 0.09, \quad C_m = 8.0$$

The boundary conditions at solid smooth walls are:

$$R_{\text{wall}} = 0 \tag{2.17}$$

And for the freestream, the authors recommend:

$$R_{\text{farfield}} = 3v_{\infty} : to : 5v_{\infty} \qquad (2.18)$$

Chapter 3: Development of a New Algebraic Transitional Flow Model

3.1 Introduction

This work is based on the recent work by Cakmakcioglu et al. [3]. They developed an algebraic transition model for the Spalart-Allmaras (SA) turbulence model. Instead of using a transport equation for solving the intermittency γ, the γ is solved by an algebraic equation which reduces the computational cost and still shows very good accuracy in several benchmark test cases.

The starting point of this research is to try to develop a new one-equation transition model, WA-T, based on algebraic intermittency gamma equation of Cakmakcioglu et al. [3] and one equation Wray-Agarwal (WA2018) turbulence model [5]. Since it has only one transport equation, it will have lower computational cost than the existing two-equation WA-γ transition model [6]. The newly developed model is validated by computing the benchmark transitional flow cases of flow over the zero-pressure gradient T3 series of flat plates. The computational results are compared to available experimental data and results from other transition models.

The one equation Wray-Agarwal (WA2018) model is a newly proposed turbulence model developed by Han et al. [5]. The one-equation WA model inherits the advantages of both the two-equation k-ω model and k-ε model with lower computational cost, and it sometimes performs even better than both the k-ω model and the k-ε model in some of the benchmark validation cases from NASA Turbulence Modeling Resource (TMR). Without any modification, this model alone cannot predict the transition from laminar flow to turbulent flow. Based on Cakmakcioglu et al.'s work [3], coupling the turbulence model with the turbulence intermittency γ obtained from an algebraic equation can provide the capability to predict the transition. The

integration of one-equation Wray-Agarwal turbulence model with γ term can provide an accurate and efficient one-equation transition prediction model with capability similar to the four equation Shear-Stress Transport (SST) transition model proposed by Menter et al. [1] but being at least two to three times more efficient.

3.2 Integration of Algebraic Transition Model with WA2018 Model

The baseline one-equation WA2018 model by Han et al. [5] is modified to include γ term as shown in Eq. (3.2). The WA model is coupled with intermittency term γ by multiplying γ with the kinetic energy production term C_1RS as shown below:

$$\frac{\partial R}{\partial t}+\frac{\partial u_j R}{\partial x_j}=\frac{\partial}{\partial x_j}\left[(\sigma_R R+\nu)\frac{\partial R}{\partial x_j}\right]+C_1\gamma RS+f_1 C_{2kw}\frac{R}{S}\frac{\partial R}{\partial x_j}\frac{\partial S}{\partial x_j}$$
$$-(1-f_1)min\left[C_{2k\omega}R^2\left(\frac{\frac{\partial S}{\partial x_j}\frac{\partial S}{\partial x_j}}{S^2}\right),C_m\frac{\partial R}{\partial x_j}\frac{\partial R}{\partial x_j}\right] \quad (3.2)$$

The value of γ will be 0 in laminar flow, and it turns into 1 in fully turbulent flow. In Wray-Agarwal (WA2018) model, the eddy viscosity is given by:

$$\nu_t=f_\mu R$$

In Eq. (3.2), the intermittency term γ is formulated as:

$$\gamma=1-\exp\left(-\sqrt{Term_1}-\sqrt{Term_2}\right) \quad (3.3)$$

Here $Term_1$ is designed to trigger the transition location, and $Term_2$ helps the intermittency to penetrate into the boundary layer [3]. $Term_1$ is given by:

$$Term_1=\frac{\max(1.2Re_\theta-Re_{\theta c},0.0)}{\chi_1 Re_{\theta c}} \quad (3.4)$$

where

$$Re_\theta = \frac{Re_v}{2.193} \text{ and } Re_v = \frac{\rho d^2}{\mu}\Omega \qquad (3.5)$$

and d is the wall distance. In Eq. (3.6), the local turbulence intensity is different from the work by Menter et al [2] where it is calculated by using k and ω. Instead, it is set to a constant value in Eq. (3.6)

$$Re_{\theta c} = 803.73(Tu_\infty + 0.6067)^{-1.027} \qquad (3.6)$$

$Term_2$ is given by:

$$Term_2 = \max\left(\frac{v_t}{v}\chi_2, 0.0\right) \qquad (3.7)$$

χ_1 and χ_2 are calibrated constants. χ_1 and χ_2 are given by:

$$\chi_1 = 0.02 \text{ and } \chi_2 = 50 \qquad (3.8)$$

In WA model, the damping function f_μ is designed to account for wall blocking effect, which is given by:

$$f_\mu = \frac{\chi^3}{\chi^3 + C_w^3}, \qquad \chi = \frac{R}{\nu} \qquad (3.9)$$

where ν is the kinematic viscosity and $R = k/\omega$. S and W are the mean strain rate and mean vorticity given by:

$$S = \sqrt{2S_{ij}S_{ij}}, \qquad S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \qquad (3.10)$$

$$W = \sqrt{2S_{ij}S_{ij}}, \qquad W_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \qquad (3.11)$$

The WA model combines the features of standard k-ω and k-ε models. The switching function f_1 triggers the behavior of a one-equation k-ω or a one-equation k-ε model. The switching function f_1 in WA model is given by:

$$f_1 = \tanh(arg_1^4), \quad arg_1 = \frac{\nu + R}{2}\frac{\eta^2}{C_\mu k\omega} \tag{3.12}$$

where

$$k = \frac{\nu_T S}{\sqrt{C_\mu}} \tag{3.13}$$

$$\omega = \frac{S}{\sqrt{C_\mu}} \tag{3.14}$$

$$\eta = S\max\left(1, \left|\frac{W}{S}\right|\right) \tag{3.15}$$

The model constants are:

$$C_{1k\omega} = 0.0829, \quad C_{1k\varepsilon} = 0.1284$$

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

$$\sigma_{kw} = 0.72, \quad \sigma_{k\varepsilon} = 1.0$$

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

$$C_{2k\omega} = \frac{C_{1k\omega}}{\kappa^2} + \sigma_{kw}, \quad C_{2k\varepsilon} = \frac{C_{1k\varepsilon}}{\kappa^2} + \sigma_{k\varepsilon}$$

$$\kappa = 0.41, \quad C_\omega = 8.54$$

$$C_\mu = 0.09, \quad C_m = 8.0$$

The boundary conditions for *R* are set to be:

$$R_\infty = 0.002\nu_\infty \tag{3.16}$$

and

$$R_{wall} = 0 \tag{3.17}$$

3.3 Validation Cases

The one-equation WA-T model was implemented in the open source computational fluid dynamics (CFD) solver OpenFOAM. The validation cases include ERCOFTAC T3 series of flat plate (T3A, T3B, and T3A-) in zero pressure gradient, T3C series of flat plates in pressure gradient and transitional flow over S809 airfoil. The computational grids were generated as structured grids using ICEM CFD, and the SIMPLE algorithm was utilized in all simulations.

3.3.1 Zero-Pressure Gradient Boundary-Layer Flow Past a Flat Plate

The first set of validation cases is flow past zero pressure gradient T3 series of flat plates (T3A, T3B and T3A-), which employ different inlet velocities and turbulence intensities as given in Table 1. A typical grid for the computational domain is shown in Fig. 1. The computational results from one-equation WA-T model are compared with the results from the four-equation SST-Transition model and the experimental data [6].

Table 1 Inlet flow conditions for T3 series of flat plates.

	U_∞ (m/s)	Tu_∞(%)	μ_T/μ	ρ (kg/m3)	μ (kg/m.s)
T3A	5.4	3.5	13.3	1.2	1.8e-5
T3B	9.4	6.5	100	1.2	1.8e-5
T3A-	19.8	0.874	8.72	1.2	1.8e-5

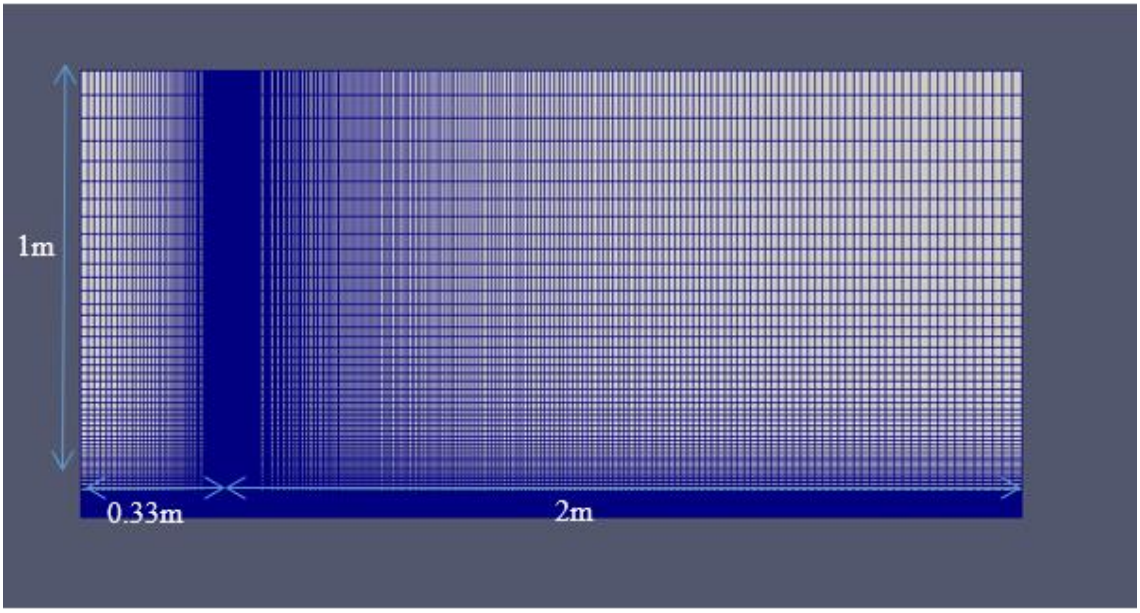

Figure 1*:* Mesh (291x191) for T3 series of flat plates.

Figures 2-4 show the comparison of computed skin friction coefficient Cf along the flat plate using the Langtry-Menter transition model (SST-T) and WA-T model with experimental data. In Fig. 2, for flow past T3A flat plate, the SST transition model identically matches the experimental data in the laminar flow region and in the transition regime. However, it cannot reach the peak value in the fully turbulent region. The WA-T model successfully predicts the peak value. In Fig. 3 in particular, WA-T model shows much better agreement with experimental data especially in transition region compared to SST-T model. In Fig. 4, WA-T has better agreement with experimental data compared to SST-T model especially in downstream part of the transition region. Overall, WA-T model shows better agreement with experimental data compared to STT-T model and is computationally three to four times more efficient than SST-T Model.

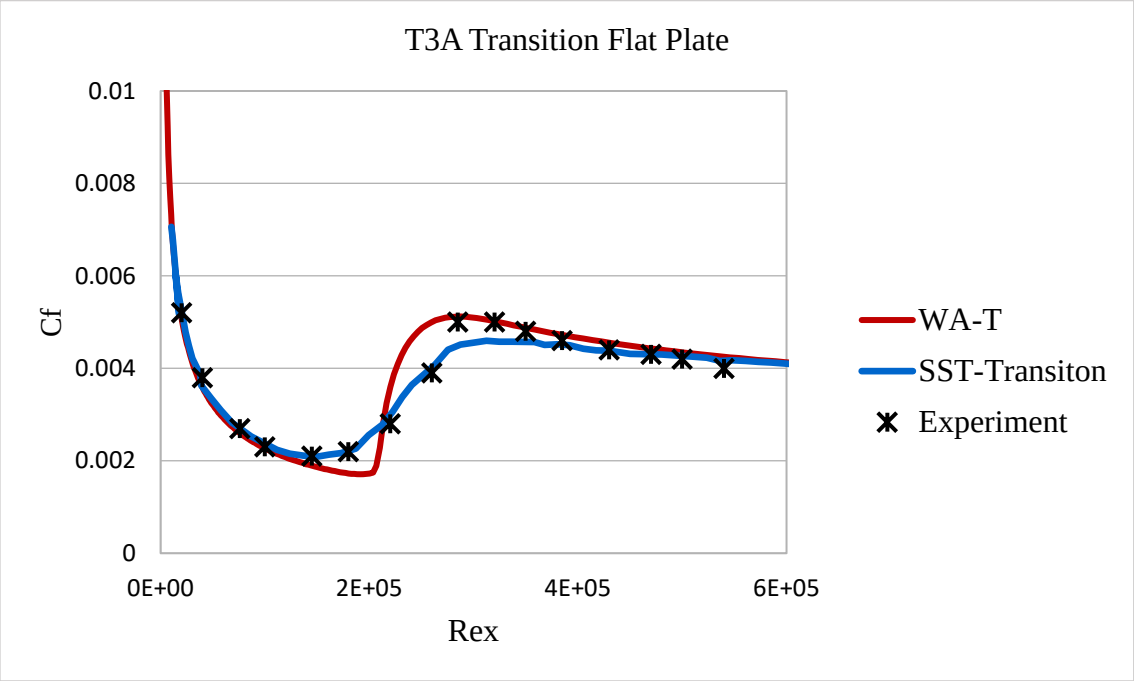

Figure 2: Transitional flow past T3A flat plate

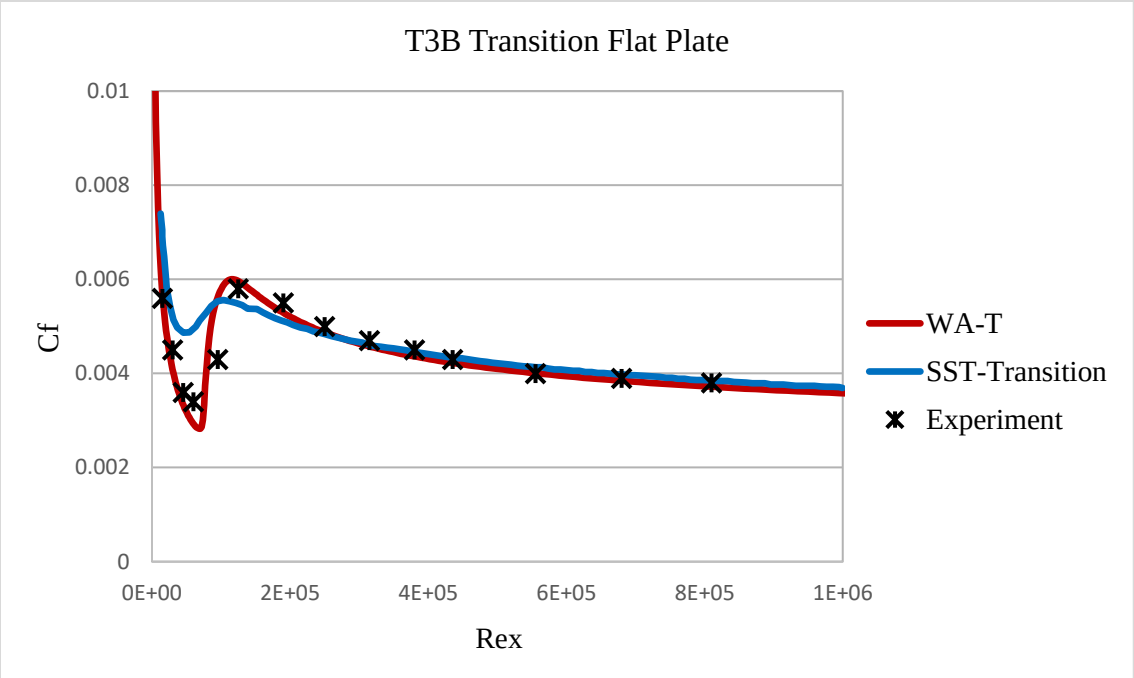

Figure 3: Transitional flow past T3B flat plate

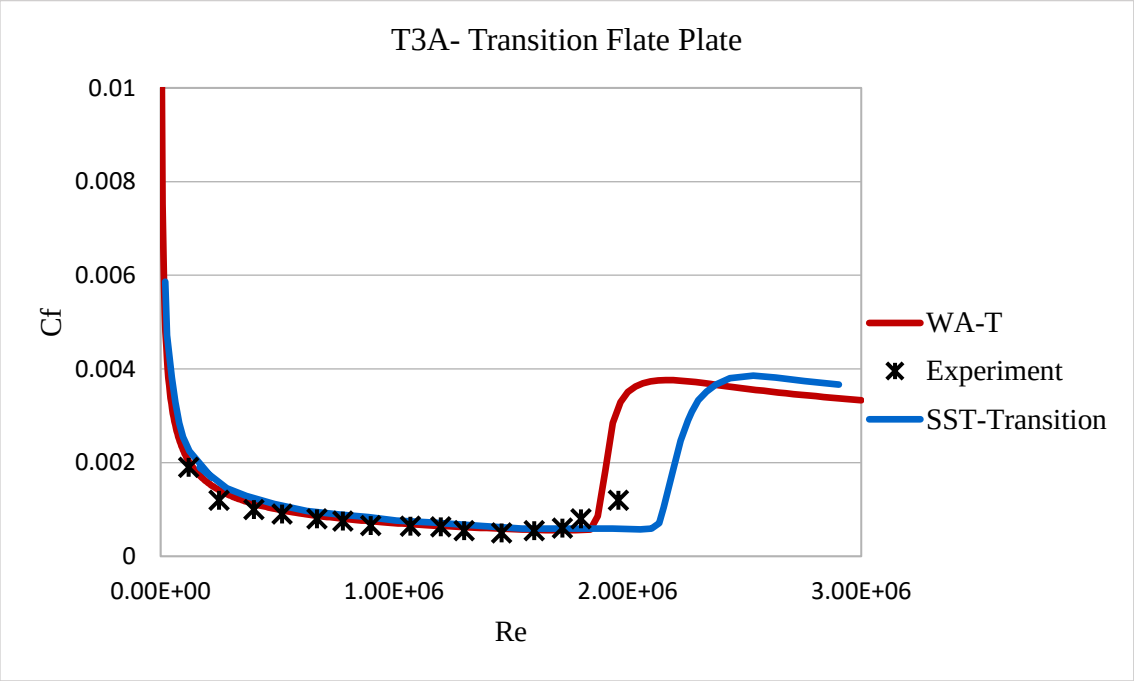

Figure 4*:* Transitional flow past T3A- flat plate

3.3.2 Non-Zero-Pressure Gradient Boundary-Layer Flow past a Flat Plate

The ERCOFTAC test cases T3C take into account the effect of pressure gradient and free-stream turbulence decay on transition prediction. A non-zero pressure gradient effect in the simulations was achieved by using the polynomial expressions from Suluskna et al [6] to modify the shape of the duct upper boundary. T3C4 was the only case that required a different shape. The polynomial expression for the domain width as a function of x for the T3C cases is expressed by Eq. (3.18), and the domain width for T3C4 is expressed by Eq. (3.19) as follows:

$$\frac{h}{D} = \min(1.231x^6 - 6.705x^5 + 14.061x^4 - 14.113x^3 + 7.109x^2 - 1.9x + 0.95, 1.0) \quad (3.18)$$

$$\frac{h}{D} = \min(1356x^6 - 7.591x^5 + 16.513x^4 - 17.510x^3 + 9.486x^2 - 2.657x + 0.991, 1.0) \quad (3.19)$$

In Eq. (3.18) and Eq. (3.19), h is the upper boundary height, D is the inlet height (0.3m), and x is distance along the plate from the leading edge. Each T3C case uses different free-stream velocity U_∞ and free-stream turbulence intensity Tu_∞ as shown in Table 2.

Table 2 Inlet flow conditions for T3C series of flat plates

	U_∞ (m/s)	Tu_∞(%)	μ_T/μ	ρ (kg/m3)	μ (kg/m.s)
T3C3	3.7	3.10	6.0	1.2	1.8e-5
T3C4	1.28	3.10	2.5	1.2	1.8e-5

Skin friction results for the T3C flat plate cases given in Table 2 are shown in Figs. 5-6. The computed results from the WA-T model for the two T3C flat plate cases are in reasonably good agreement with the experimental values [6].

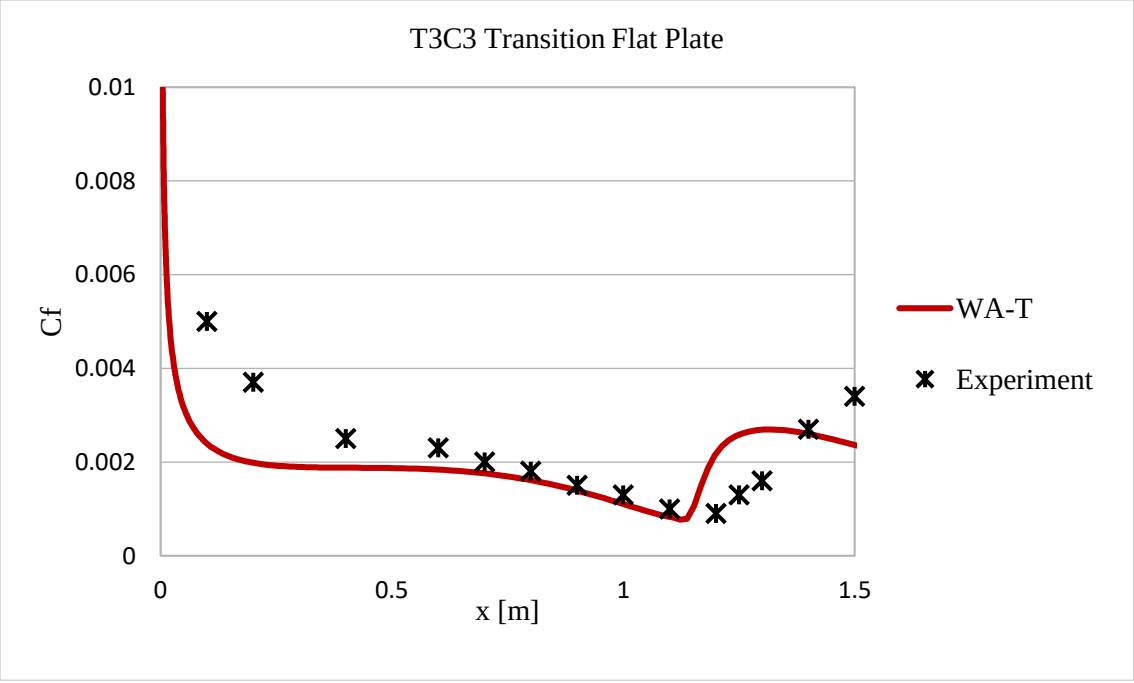

Figure 5*:* Transitional flow past T3C3 flat plate

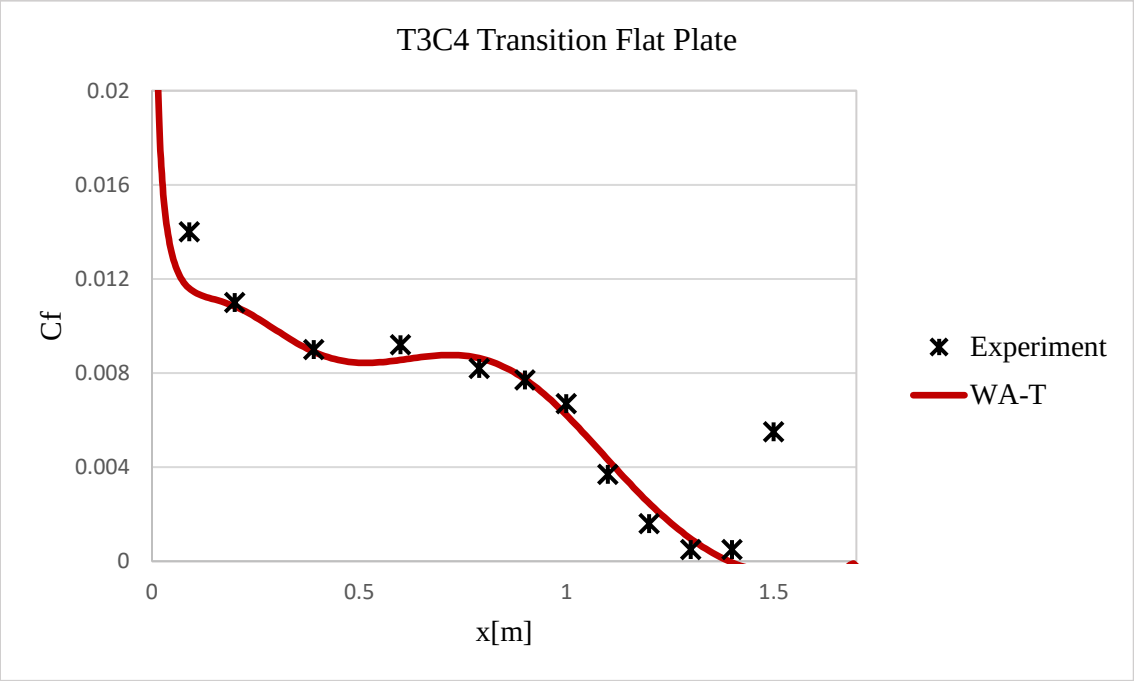

Figure 6*:* Transitional flow past T3C4 flat plate

3.3.2 Flow past a 2D S809 Airfoil

The S809 airfoil was designed for wind turbines; it is thick airfoil with 21% thick chord. It is a laminar-flow airfoil for horizontal-axis wind-turbine applications. Accurate computational results may help in greatly improving the design of the airfoil in other energy-generating applications. In this case, the Reynolds number based on the chord length is 2 million and three angles of attack are considered --- 0°, 5° and 10°. For every AOA, the inlet turbulence intensity is Tu_∞ = 0.2% and the viscosity ratio $\mu_t/\mu = 10$.

Figures 7-9 show the comparison of pressure coefficients on the airfoil. The WA-T transition model is compared with experimental data and the results from SST-T model. When AOA = 0 and 5 degrees, the WA-T model accurately predicts C_p near the trailing edge compared to SST-T model and the overall performance of WA-T model in these cases is very good. For

AOA = 10 degree, the results from both WA-T and SST-T model are in good agreement with the experimental data.

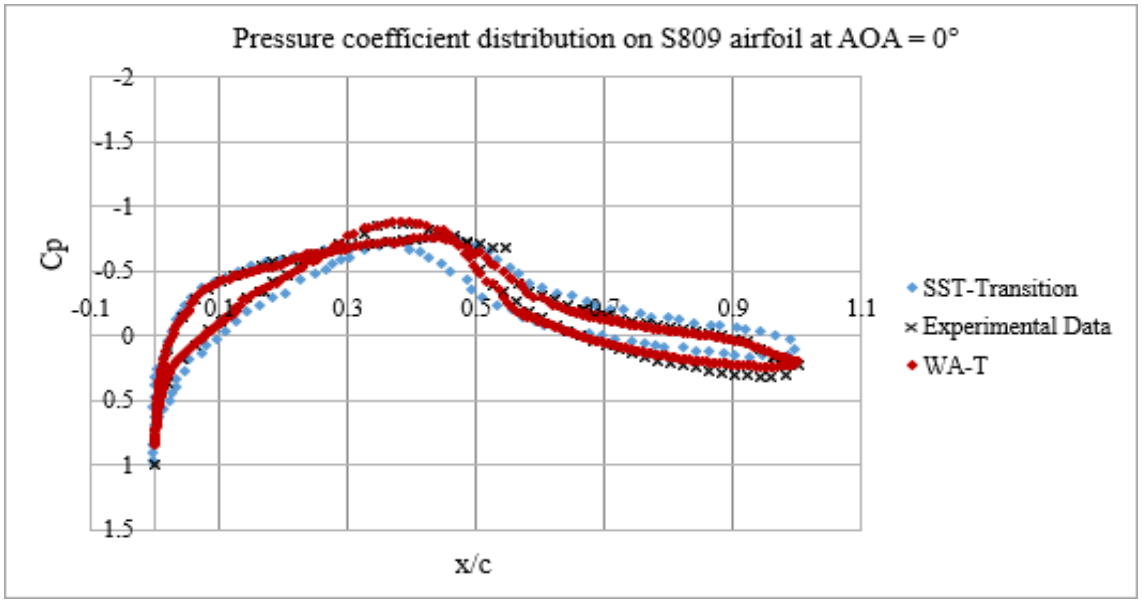

Figure 7: Pressure coefficient distribution on S809 airfoil at AOA = 0°

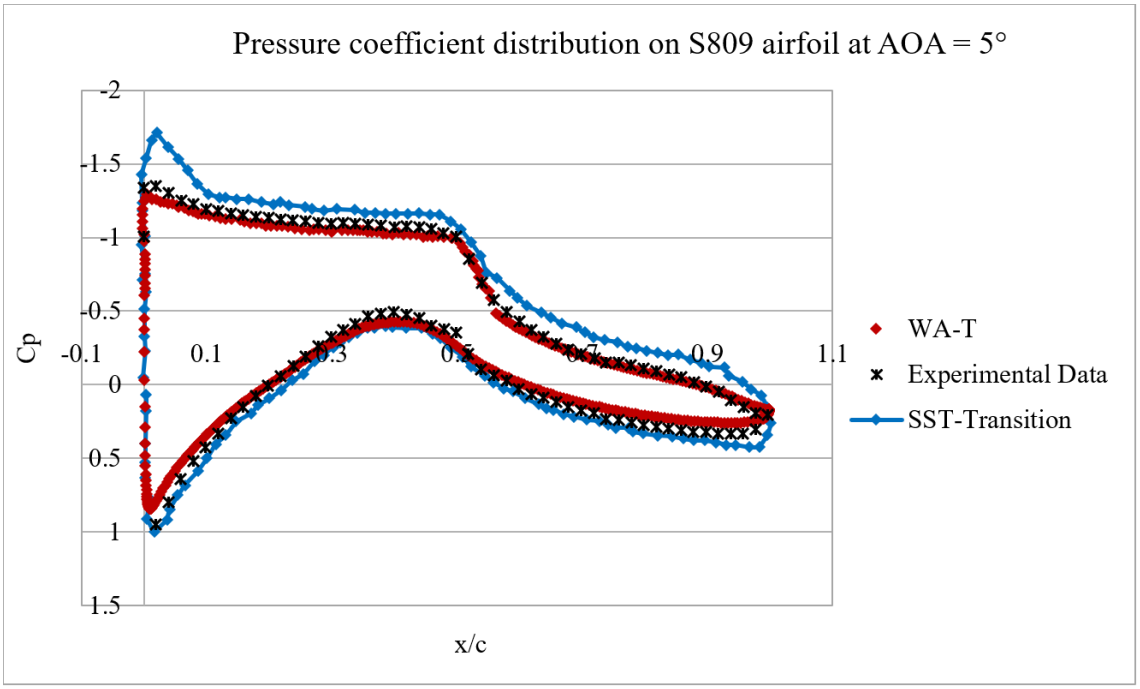

Figure 8: Pressure coefficient distribution on S809 airfoil at AOA = 5°

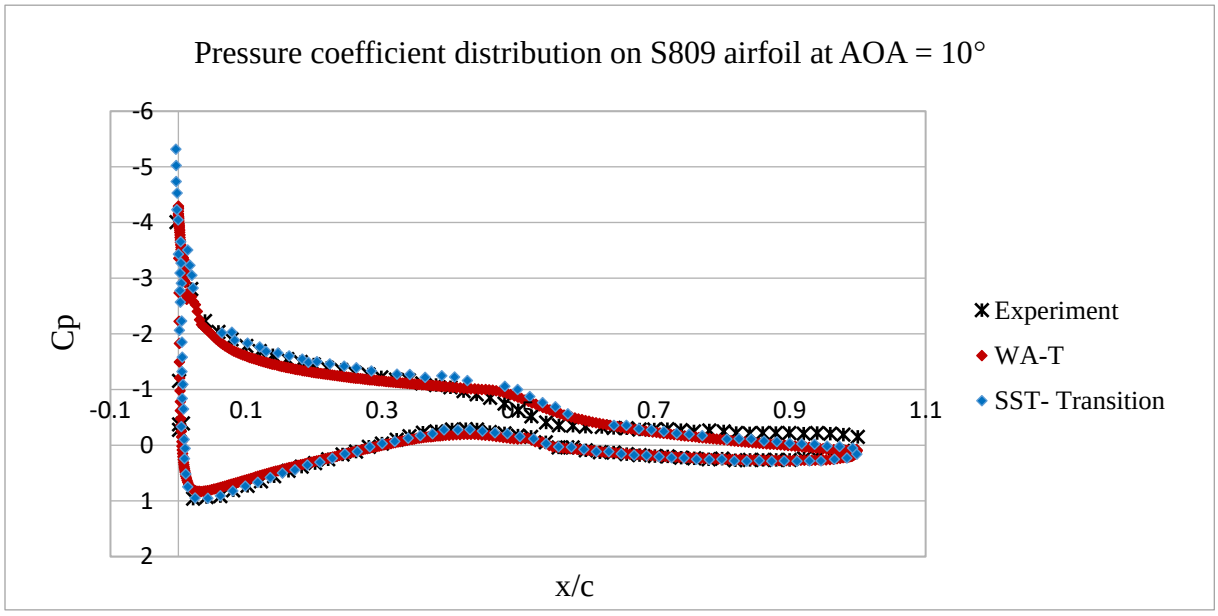

Figure 9: Pressure coefficient distribution on S809 airfoil at AOA = 10°

Chapter 4: Summary

This present work employs the recent research idea by Cakmakcioglu et al [3] which couples an algebraic transition model with the Spalart-Allmaras (SA) one-equation turbulence model. A new algebraic transition model is developed which is coupled with the one-equation Wall-Distance Free Wray-Agarwal (2018) model. The model is applied to compute benchmark ERCOFTAC test cases, namely T3A, T3B and T3A- for transitional flow over a flat plate with different free stream turbulent intensity and zero pressure gradient, and T3C series cases of flow over a flat plate under pressure gradient and different free stream turbulence intensity, and transitional flow past S809 airfoil. WA-T shows improved accuracy compared to the four-

equation Langtry-Menter SST k-ω-Re_θ-γ transition model. As a one-equation model, WA-T is very efficient which has lower computational cost and gives reasonably accurate results. The model still needs further validation e.g. transition flow past Aerospatiale-A airfoil and NRL-7301 two-element airfoil among other benchmark cases for transitional flow. Based on the current work, it can be concluded that the WA-T model has great potential and it is worth investigating it more in-depth. This model is currently being further developed and applied to airfoil/wing applications.

References

[1]Menter, F. R., Langtry, R. B., Likki, S. R., Suzen, Y. B., Huang, P. G., and Völker, S., "A Correlation-Based Transition Model Using Local Variables—Part I: Model Formulation," J. of Turbomachinery, Vol. 128, 2006, p. 413.

[2]Menter, F. R., Smirnov, P. E., Liu, T., and Avancha, R., "A One-Equation Local Correlation-Based Transition Model," Flow, Turbulence and Combustion, Vol. 95, 2015, pp. 583–619.

[3]Cakmakcioglu, S. C., Bas, O., Mura, R. and Kaynak, U., "A Revised One-Equation Transitional Model for External Aerodynamics," AIAA Paper 2020-2076, AIAA Aviation Forum 2020, June 2020.

[4]Menter, F. R., Langtry, R., and Völker, S., "Transition Modelling for General Purpose CFD Codes," Flow, Turbulence and Combustion, Vol. 77, No. 1-4, 2006, pp. 277–303. https://doi.org/10.1007/s10494-006-9047-1

[5]Han, X., Rahman, M. M., and Agarwal, R. K., "Development and Application of a Wall Distance Free Wray-Agarwal Turbulence Model," AIAA Paper 2018-0593, AIAA SciTech Forum, Kissimmee, FL, 8-12 January 2018.

[6] Nagapetyan, H., and Agarwal, R. K., "Development of a New Transitional Flow Model Integrating the Wray-Agarwal Turbulence Model with an Intermittency Transport Equation," AIAA Paper 2018-3384, AIAA 2018 Fluid Dynamics Conference, Atlanta, Georgia, 25-29 June 2018.

[7] ERCOFTAC (European Research Community on Flow, Turbulence and Combustion), http://ercoftac.mech.surrey.ac.uk/ [retrieved 20 June 2020]

Appendix A: Source Code of WA-T Transition Model

A1. WA-T.C

```
/*---------------------------------------------------------------------------*\
 =========                 |
 \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
  \\    /   O peration     |
   \\  /    A nd           | Copyright (C) 2011-2015 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 "WAone.H"
#include "bound.H"
#include "wallDist.H"
#include "wallFvPatch.H"

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

namespace Foam
{
namespace RASModels
{

// * * * * * * * * * * * * Protected Member Functions  * * * * * * * * * * * //
template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::chi() const
{
    return Rnu_/this->nu();
```

```
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::fmu
(
    const volScalarField& chi
) const
{
    const volScalarField chi3(pow3(chi));
    return chi3/(chi3 + pow3(Cw_));
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::WDF_R
(
    const volScalarField& S,
    const volScalarField& W
) const
{
    return mag(W/S);
}
```

```
template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::WDF_omega
(
    const volScalarField& S
) const
{
    return S/sqrt(Cmu_);
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::WDF_k
(
    const volScalarField& omega
) const
{
    return this->nut_*omega;
        //return Rnu_*omega;
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::arg1
```

```
(
    const volScalarField& S,
    const volScalarField& W
) const
{
    const volScalarField R = WDF_R(S, W);
    const volScalarField omega = WDF_omega(S);
    const volScalarField k = WDF_k(omega);

    const volScalarField eta = S*max(1.0, R);

    return (this->nu()+Rnu_)/2 * sqr(eta)/max(Cmu_*k*omega,
                                    dimensionedScalar("SMALL",
                                                dimensionSet(0, 2, -3, 0, 0),
                                                SMALL)
                                   );
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::calcSwitch
(
    const volScalarField& S,
```

```
    const volScalarField& W
) const
{
    return tanh(pow(Cs1_*arg1(S, W), Cs2_));
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::blend
(
    const volScalarField& Switch,
    const dimensionedScalar& psi1,
    const dimensionedScalar& psi2
) const
{
    return Switch*(psi1 - psi2) + psi2;
}

template<class BasicTurbulenceModel>
tmp<volScalarField>  WAone<BasicTurbulenceModel>::sigmaR(const  volScalarField&
Switch) const
{
    return blend(Switch, sigmakw_, sigmake_);
```

```
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::C1(const volScalarField&
Switch) const
{
    return blend(Switch, C1kw_, C1ke_);
}

//////////////////////////////////////////////////////////////////transition
template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::Rev
(
  const volScalarField& W
) const
{
   return this->rho_*sqr(y_)*W/this->mu();
   //return this->rho_*sqr(y_)*S/this->mu();
}

template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::Retheta
```

```
(
  const volScalarField& Rev
) const
{
   return Rev/2.193;
}




//////////////////////////////////////////////////////////////////////

template<class BasicTurbulenceModel>
void WAone<BasicTurbulenceModel>::correctNut
(
    const volScalarField& fmu
)
{
    this->nut_ = Rnu_*fmu;
    this->nut_.correctBoundaryConditions();


    BasicTurbulenceModel::correctNut();
}
```

```
template<class BasicTurbulenceModel>
void WAone<BasicTurbulenceModel>::correctNut()
{
    correctNut(fmu(this->chi()));
}

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

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

```
eddyViscosity<RASModel<BasicTurbulenceModel> >
(
    type,
    alpha,
    rho,
    U,
    alphaRhoPhi,
    phi,
    transport,
    propertiesName
),

kappa_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "kappa",
        this->coeffDict_,
        0.41
    )
),
Cw_
```

```
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cw",
        this->coeffDict_,
        8.54
    )
),

C1ke_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "C1ke",
        this->coeffDict_,
                    0.1284
    )
),

C1kw_
(
    dimensioned<scalar>::lookupOrAddToDict
```

```
    (
        "C1kw",
        this->coeffDict_,
                        0.0829
    )
),

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

sigmakw_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "sigmakw",
```

```
        this->coeffDict_,
                        0.72
    )
),

C2ke_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "C2ke",
        this->coeffDict_,
                        C1ke_.value()/sqr(kappa_.value()) + sigmake_.value()
    )
),

C2kw_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "C2kw",
        this->coeffDict_,
                        C1kw_.value()/sqr(kappa_.value()) + sigmakw_.value()
```

```
    )
),

Cmu_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cmu",
        this->coeffDict_,
                    0.09
    )
),

Cs1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cs1",
        this->coeffDict_,
                    1.0
    )
),
```

```
Cs2_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cs2",
        this->coeffDict_,
                        4.0
    )
),

Cm_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Cm",
        this->coeffDict_,
                        8.0
    )
),

Rnu_
```

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

utau_
(
    IOobject
    (
        "utau",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
```

```
    this->mesh_,
    dimensionedScalar("0.0", dimensionSet(0, 1, -1, 0, 0), 0.0)
),

wGUx_
(
    IOobject
    (
        "wGUx",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_,
    dimensionedScalar("0.0", dimensionSet(0, 0, -1, 0, 0), 0.0)
),
wallGradUTest_
(
    IOobject
    (
        "wallGradUTest",
```

```
            this->runTime_.timeName(),
            this->mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        this->mesh_,
        dimensionedVector
        (
            "wallGradU",
            U.dimensions()/dimLength,
            vector::zero
        )
    ),
    f1_
    (
        IOobject
        (
            "f1",
            this->runTime_.timeName(),
            this->mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
```

```
    ),
    this->mesh_,
    dimensionedScalar("0.0", dimensionSet(0, 0, 0, 0, 0), 0.0)
),
S_
(
    IOobject
    (
        "S",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    this->mesh_,
    dimensionedScalar("0.0", dimensionSet(0, 0, -1, 0, 0), 0.0)
),
W_
(
    IOobject
    (
        "W",
```

```
            this->runTime_.timeName(),
            this->mesh_,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        this->mesh_,
        dimensionedScalar("0.0", dimensionSet(0, 0, -1, 0, 0), 0.0)
    ),
//////////////////transition
    chi1_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "chi1",
            this->coeffDict_,
            0.02
        )
    ),
    /*
    chi2_
    (
        dimensioned<scalar>::lookupOrAddToDict
```

```
        (
            "chi2",
            this->coeffDict_,
            50.0
        )
    ),

*/
    Tu_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "Tu",
            this->coeffDict_,
            1.0
        )
    ),

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

```
        "RethetaC",
        this->coeffDict_,
        //803.73*pow((Tu_.value()+0.6067),-1.027)
        803.73*pow((Tu_.value()+0.6067),-1.027)
    )
),

gamma_
(
    IOobject
    (
        "gamma",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
this->mesh_,
dimensionedScalar("gamma", dimless, scalar(1.0))
),

term1m_
```

```
(
    IOobject
    (
        "term1m",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
this->mesh_,
dimensionedScalar("1", dimless, scalar(0.0))
),

term2m_
(
    IOobject
    (
        "term2m",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
```

```
    ),
this->mesh_,
dimensionedScalar("2", dimless, scalar(0.0))
),

nuBCm_
(
    IOobject
    (
        "nuBCm",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
this->mesh_,
dimensionedScalar("0.0", dimensionSet(0, 0, 0, 0, 0), 1.0)
),

chi2m_
(
    IOobject
```

```
    (
        "chi2m",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
this->mesh_,
dimensionedScalar("0.0", dimensionSet(0, 0, 0, 0, 0), 1.0)
),

Rethetam_
(
    IOobject
    (
        "Rethetam",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
this->mesh_,
```

```
dimensionedScalar("0.0", dimensionSet(0, 0, 0, 0, 0), 1.0)
),

tm_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "tm",
        this->coeffDict_,
        50
    )
),

plim_
(
    IOobject
    (
        "plim",
        this->runTime_.timeName(),
        this->mesh_,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
```

```
    ),
    this->mesh_,
    dimensionedScalar("0.0", dimensionSet(0, 2, -2, 0, 0), 0.0)
),

Ct1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "Ct1",
        this->coeffDict_,
        1.0
    )
),

CP1_
(
    dimensioned<scalar>::lookupOrAddToDict
    (
        "CP1",
        this->coeffDict_,
        1.2
```

```
        )
    ),

    CP2_
    (
        dimensioned<scalar>::lookupOrAddToDict
        (
            "CP2",
            this->coeffDict_,
            1.0
        )
    ),

            y_(wallDist::New(this->mesh_).y())

{
    if (type == typeName)
    {
        this->printCoeffs(type);
    }
}
```

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

template<class BasicTurbulenceModel>
bool WAone<BasicTurbulenceModel>::read()
{
    if (eddyViscosity<RASModel<BasicTurbulenceModel> >::read())
    {
        return true;
    }
    else
    {
        return false;
    }
}
template<class BasicTurbulenceModel>
tmp<volScalarField>            WAone<BasicTurbulenceModel>::DRnuEff(volScalarField
Switch) const
{
    return tmp<volScalarField>
    (
        new volScalarField("DRnuEff", Rnu_*sigmaR(Switch) + this->nu())
```

```
    );
}



template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::k() const
{
    return tmp<volScalarField>
    (
        new volScalarField
        (
            IOobject
            (
                "k",
                this->runTime_.timeName(),
                this->mesh_
            ),
            this->mesh_,
            dimensionedScalar("0", dimensionSet(0, 2, -2, 0, 0), 0)
        )
    );
}
```

```
template<class BasicTurbulenceModel>
tmp<volScalarField> WAone<BasicTurbulenceModel>::epsilon() const
{
    WarningInFunction
        << "Turbulence kinetic energy dissipation rate not defined for "
        << "Spalart-Allmaras model. Returning zero field"
        << nl;

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

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

    // Local references
    const alphaField& alpha = this->alpha_;
    const rhoField& rho = this->rho_;
    const surfaceScalarField& alphaRhoPhi = this->alphaRhoPhi_;
        const volVectorField& U = this->U_;
    eddyViscosity<RASModel<BasicTurbulenceModel> >::correct();
    // Calculate strain rate magnitude S
        volScalarField S2(2.0*magSqr(symm(fvc::grad(this->U_))));
        volScalarField S = sqrt(S2);
        bound(S, dimensionedScalar("0", S.dimensions(), SMALL)); // SMALL = 1e-15
        bound(S2, dimensionedScalar("0", S2.dimensions(), SMALL));
        S_ = S;
```

```
    // Calculate vorticity magnitude W
    volScalarField W2(2.0*magSqr(skew(fvc::grad(this->U_))));
    volScalarField W = sqrt(W2);
        bound(W, dimensionedScalar("0", W.dimensions(), SMALL));
        bound(W2, dimensionedScalar("0", W2.dimensions(), SMALL));
        W_ = W;

        volScalarField magU = mag(U);
        volScalarField nueff = this->nut_+this->nu();
        const volScalarField Rev(this->Rev(W));
    const volScalarField Retheta(this->Retheta(Rev));

        //volScalarField nuBC = this->nut_/(mag(this->U_)*y_);

        volScalarField nuBC = Rnu_/(mag(this->U_)*y_);
        //nuBCm_ = nuBC;
        volScalarField Term2 = nuBC;

        forAll(nuBC, cellI)
{
        //nuBC[cellI] = (0.3*Rnu_[cellI])/(S[cellI]*sqr(y_[cellI]));
        nuBC[cellI] = (CP2_.value()*Rnu_[cellI])/(S[cellI]*sqr(y_[cellI]));
```

```
            //Term2[cellI] = max(nuBC[cellI]-0.0005,0.0)/0.0005;
        }
            nuBCm_=nuBC;
            term2m_= max(tm_*this->nut_/this->nu(),0.0);

            term1m_ = max(CP1_*Retheta-RethetaC_,0.0)/(chi1_*RethetaC_); //2.5

            gamma_ = 1.0-exp(-sqrt(Ct1_*term1m_)-sqrt(term2m_)); //0.01term1

          gamma_=min(gamma_,scalar(1.0));
          bound(gamma_,scalar(0));
          eddyViscosity<RASModel<BasicTurbulenceModel> >::correct();
         //////////transitionBC local reference

            chi2m_ = this->nut_/this->nu();
            Rethetam_ = Retheta;

          // Calculate switch function (f1)
          f1_ = calcSwitch(S, W);
          plim_  =  0.5*max(gamma_-0.2,  0.0)*(1.0-gamma_)*min(max((Rev/2420.0)-1.0,  0.0),
3.0)*max(3.0*this->nu()-this->nut_, dimensionedScalar("0", dimensionSet(0, 2, -1, 0, 0), 0))*W;
```

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


// Define and solve R-Equation
tmp<fvScalarMatrix> RnuEqn
(
    fvm::ddt(alpha, rho, Rnu_)
  + fvm::div(alphaRhoPhi, Rnu_)
  - fvm::laplacian(alpha*rho*DRnuEff(f1_), Rnu_)
 ==
    alpha*rho*C1(f1_)*gamma_*fvm::Sp(S, Rnu_)
  + alpha*rho*f1_*C2kw_*fvm::Sp((fvc::grad(Rnu_)&fvc::grad(S))/S, Rnu_)
  - alpha*rho*(1.0-f1_)*min(C2ke_*Rnu_*Rnu_*magSqr(fvc::grad(S))/S2,
                            Cm_*magSqr(fvc::grad(Rnu_)))

);

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

```
    }

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


    } // End namespace RASModels

    } // End namespace Foam


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

A2. WA-T.H

```
/*---------------------------------------------------------------------------*\
 =========                 |
 \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
  \\    /   O peration     |
   \\  /    A nd           | Copyright (C) 2011-2015 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::RASModels::WAone

Group

grpRASTurbulence

Description

Wray-Agarwal Turbulence Model with Rahman's wall distance free modification with type-2 switch function, and Han's coefficient adjustment.

SourceFiles

WAone.C

---/

#ifndef WAone_H

#define WAone_H

#include "RASModel.H"

#include "eddyViscosity.H"

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

namespace Foam
{
namespace RASModels
{

/*---------------------------------------------------------------------------*\
                    Class WAone Declaration
\*---------------------------------------------------------------------------*/

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

        // Disallow default bitwise copy construct and assignment
        WAone(const WAone&);
        WAone& operator=(const WAone&);
```

```
protected:

    // Protected data

        // Model coefficients

            dimensionedScalar kappa_;
            dimensionedScalar ks_;
            dimensionedScalar Cw_;
            dimensionedScalar Cr1_;
            dimensionedScalar Cr2_;
            dimensionedScalar Cr3_;
            dimensionedScalar C1ke_;
                        dimensionedScalar C1kw_;
            dimensionedScalar sigmake_;
            dimensionedScalar sigmakw_;
                        dimensionedScalar C2ke_;
            dimensionedScalar C2kw_;
            dimensionedScalar Cmu_;
            dimensionedScalar Cs1_;
            dimensionedScalar Cs2_;
```

```
        dimensionedScalar Cm_;
    // Fields

        volScalarField Rnu_;
        volScalarField utau_;
        volScalarField wGUx_;
        volVectorField wallGradUTest_;
                    volScalarField f1_;
        volScalarField S_;
        volScalarField W_;
///tran
        dimensionedScalar chi1_;
        //dimensionedScalar chi2_;
        dimensionedScalar Tu_;
        dimensionedScalar RethetaC_;
                    volScalarField gamma_;
                    volScalarField term1m_;
                    volScalarField term2m_;
                    volScalarField nuBCm_;
                    volScalarField chi2m_;
                    volScalarField Rethetam_;
        dimensionedScalar tm_;
```

```
        volScalarField plim_;
        dimensionedScalar Ct1_;
        dimensionedScalar CP1_;
        dimensionedScalar CP2_;

                        const volScalarField& y_;

    // Protected Member Functions

        tmp<volScalarField> chi() const;

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

        tmp<volScalarField> WDF_R
        (
            const volScalarField& S,
            const volScalarField& W
        ) const;

        tmp<volScalarField> WDF_omega
        (
            const volScalarField& S
```

```
) const;

tmp<volScalarField> WDF_k
(
    const volScalarField& omega
) const;

tmp<volScalarField> arg1
(
    const volScalarField& S,
    const volScalarField& W
) const;

tmp<volScalarField> calcSwitch
(
    const volScalarField& S,
    const volScalarField& W
) const;

tmp<volScalarField> blend
(
    const volScalarField& Switch,
```

```
            const dimensionedScalar& psi1,
            const dimensionedScalar& psi2
        ) const;

        tmp<volScalarField> sigmaR(const volScalarField& Switch) const;

        tmp<volScalarField> C1(const volScalarField& Switch) const;

        tmp<volScalarField> Rev(const volScalarField& W) const;

        tmp<volScalarField> Retheta(const volScalarField& Rev) const;

        void correctNut(const volScalarField& fmu);
        virtual void correctNut();

public:

    typedef typename BasicTurbulenceModel::alphaField alphaField;
    typedef typename BasicTurbulenceModel::rhoField rhoField;
    typedef typename BasicTurbulenceModel::transportModel transportModel;
    //- Runtime type information
    TypeName("WAone");
```

```
// Constructors
    //- Construct from components
    WAone
    (
        const alphaField& alpha,
        const rhoField& rho,
        const volVectorField& U,
        const surfaceScalarField& alphaRhoPhi,
        const surfaceScalarField& phi,
        const transportModel& transport,
        const word& propertiesName = turbulenceModel::propertiesName,
        const word& type = typeName
    );
//- Destructor
virtual ~WAone()
{}
// Member Functions
    //- Re-read model coefficients if they have changed
    virtual bool read();

    //- Return the effective diffusivity for Rnu
    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;

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

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

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

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

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

```
#endif

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

Curriculum Vita

Yan Xue

Degrees **M.S. in Mechanical Engineering**, Washington University in St. Louis, St. Louis, Missouri, USA May 2021

B.S. in Mechanical Engineering, Xi'an Jiaotong University, Xi'an, Shanxi, China June 2018