INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Semi-discrete Galerkin solution of the compressible boundary-layer equations with viscous-inviscid interaction

Day, Brad Allen, M.S.

Rice University, 1993
RICE UNIVERSITY

Semi-Discrete Galerkin Solution of the Compressible Boundary-Layer Equations with Viscous-Inviscid Interaction

by

Brad Allen Day

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Andrew J. Merle, Jr.
Assistant Professor of Mechanical Engineering and Materials Science, Chairman

Yildiz Bayazitoglu
Professor of Mechanical Engineering and Materials Science

Balasubramaniam Ramaswamy
Assistant Professor of Mechanical Engineering and Materials Science

Houston, Texas

December, 1992
Abstract

Semi-Discrete Galerkin Solution of the Compressible Boundary-Layer Equations with Viscous-Inviscid Interaction

by

Brad Allen Day

A method is developed to solve the two-dimensional, steady, compressible, turbulent boundary-layer equations and is coupled to an existing Euler solver for attached transonic airfoil analysis problems. The boundary-layer formulation utilizes the semi-discrete Galerkin (SDG) method to model the spatial variable normal to the surface with linear finite elements and the time-like variable with finite differences. A Dorodnitsyn transformed system of equations is used to bound the infinite spatial domain thereby permitting the use of a uniform finite element grid which provides high resolution near the wall and automatically follows boundary-layer growth. The second-order accurate Crank-Nicholson scheme is applied along with a linearization method to take advantage of the parabolic nature of the boundary-layer equations and generate a non-iterative marching routine. The SDG code can be applied to any smoothly-connected airfoil shape without modification and can be coupled to any inviscid flow solver. In this analysis, a direct viscous-inviscid interaction is accomplished between the Euler and boundary-layer codes through the application of a transpiration velocity boundary condition. Results are presented for compressible turbulent flow past NACA 0012 and RAE 2822 airfoils at various freestream Mach numbers, Reynolds numbers, and angles of attack. All results show good agreement with experiment, and the coupled code has proven to be a computationally-efficient and accurate airfoil analysis tool.
Acknowledgments

I would first like to thank my research advisor, Dr. Andrew J. Meade, Jr., for providing me the opportunity to pursue an advanced degree at Rice University, and to express my gratitude for the patience, support, and encouragement he has given me over the past two years. Also, I would like to acknowledge Mr. Richard Campbell of NASA Langley Research Center for the support given under Grant No. NAG-1-1196.

I would like to thank Dr. Peter Hartwich and project monitor Dr. Dennis Allison of NASA Langley Research Center for the use of GAUSS2 and seemingly endless access to their aerodynamic expertise. My personal thanks to office-mates Scott Miller, Stuart Strong, and Daniel Badders for their collective work as an invaluable sounding board.

I am eternally grateful to my parents, Brenda and Connie, for their guidance throughout the years, and for instilling in me the belief that hard work and dedication is an unexceptionable substitute for natural ability.

Finally, I would like to thank my wife, Karen. Her untiring support and encouragement during this period has been uplifting and a source of motivation for me.
# Table of Contents

Abstract \hspace{1cm} ii  
Acknowledgments \hspace{1cm} iii  
List of Figures \hspace{1cm} vi  
List of Tables \hspace{1cm} viii  
Nomenclature \hspace{1cm} ix  

## 1 Introduction

## 2 Finite Element Method

2.1 Finite Element Approximation \hspace{1cm} 3  
2.2 Interpolation Functions \hspace{1cm} 4  
2.3 Method of Weighted Residuals \hspace{1cm} 6  
2.3.1 Petrov-Galerkin Method \hspace{1cm} 7  
2.4 Semi-Discrete Galerkin Method \hspace{1cm} 7  
2.5 Group Finite Element Method \hspace{1cm} 10  

## 3 Boundary Layer Formulation

3.1 Governing Equations \hspace{1cm} 12  
3.2 Traditional Dorodnitsyn Formulation \hspace{1cm} 14  
3.3 Semi-Discrete Galerkin Formulation \hspace{1cm} 18  
3.4 Self-Similar Solutions \hspace{1cm} 25  
3.4.1 Momentum Similarity \hspace{1cm} 26  
3.4.2 Energy Similarity \hspace{1cm} 27  
3.5 Turbulence Models \hspace{1cm} 28  

## 4 Viscous-Inviscid Interaction

4.1 Viscous-Inviscid Coupling \hspace{1cm} 30  
4.1.1 Displacement Thickness \hspace{1cm} 31  
4.1.2 Transpiration Velocity \hspace{1cm} 32  

4.2 Euler Equation Solver .......................... 33
  4.2.1 Viscous-Inviscid Interfaces .......... 34

5 Numerical Results ................................. 35
  5.1 Convergence Properties .................... 35
  5.2 Flat Plate .................................. 36
  5.3 NACA 0012 Airfoil ......................... 37
    5.3.1 NACA 0012 - Case A ................ 40
    5.3.2 NACA 0012 - Case B ................ 40
    5.3.3 Aerodynamic Characteristics ....... 44
  5.4 RAE 2822 Airfoil ........................... 44
    5.4.1 RAE 2822 - Case A .................. 46
    5.4.2 RAE 2822 - Case B .................. 47
    5.4.3 RAE 2822 - Case C .................. 47
  5.5 Grid Refinement Study ..................... 51
  5.6 Aerodynamic Force Coefficients ......... 53

6 Conclusions ...................................... 56

A Galerkin Integrals ................................. 57

B Boundary Layer Source Code .......................... 61

Bibliography ........................................ 150
List of Figures

2.1 Finite Element Representation of $\phi(x)$ .............................................. 4
2.2 Linear Interpolation Function $N_i^{(e)}(x)$ ............................................... 5

3.1 Airfoil Coordinate System ................................................................. 19
3.2 Computational Plane .............................................................................. 19

4.1 Viscous-Inviscid Interaction Scheme ...................................................... 31
4.2 Viscous Flow Momentum Defect ............................................................ 32

5.1 Nondimensional Shear Stress Convergence Results for a Flat Plate at $M_\infty = 0.800$, $Re = 5000$ .................................................. 36
5.2 Comparison of Computed and Exact Nondimensional Shear Stress Profiles for a Flat Plate at $x = 0.5$, $M_\infty = 0.800$, $Re = 5000$ ........... 37
5.3 3 and 9-Node Comparison of Computed and Exact Skin Friction for a Flat Plate at $M_\infty = 0.800$, $Re = 5000$ ..................................................... 38
5.4 NACA 0012 C-Grid used by GAUSS2 ...................................................... 39
5.5 Comparison of Pressure Coefficient Distribution for the NACA 0012 Airfoil at $M_\infty = 0.499$, $\alpha_{num} = -0.14$, $Re = 9.0 \times 10^6$ .................. 41
5.6 Skin Friction Coefficient Distribution for the NACA 0012 Airfoil Upper-Surface at $M_\infty = 0.499$, $\alpha_{num} = -0.14$, $Re = 9.0 \times 10^6$ ...... 41
5.7 Upper-Surface Displacement Thickness for the NACA 0012 Airfoil at $M_\infty = 0.499$, $\alpha_{num} = -0.14$, $Re = 9.0 \times 10^6$ ......................... 42
5.8 Comparison of Pressure Coefficient Distribution for the NACA 0012 Airfoil at $M_\infty = 0.700$, $\alpha_{num} = 1.37$, $Re = 9.0 \times 10^6$ ............ 42
5.9 Skin Friction Coefficient Distribution for the NACA 0012 Airfoil Upper-Surface at $M_\infty = 0.700$, $\alpha_{num} = 1.37$, $Re = 9.0 \times 10^6$ ...... 43
5.10 Upper-Surface Displacement Thickness for the NACA 0012 Airfoil at $M_\infty = 0.700$, $\alpha_{num} = 1.37$, $Re = 9.0 \times 10^6$ ....................... 43
5.11 Comparison of Lift Coefficient vs. Angle of Attack for the NACA 0012 Airfoil at $M_\infty = 0.700$, $Re = 9.0 \times 10^6$ .................................................. 45

5.12 Comparison of Lift vs. Drag Polars for the NACA 0012 Airfoil at $M_\infty = 0.700$, $Re = 9.0 \times 10^6$ .................................................. 45

5.13 Comparison of Computed and Measured Transonic Drag-Rise Characteristics for the NACA 0012 Airfoil at $\alpha_{num} = 0$, $Re = 9.0 \times 10^6$ 46

5.14 Comparison of Pressure Coefficient Distribution for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = -2.18$, $Re = 5.7 \times 10^6$ ................ 48

5.15 Comparison of Upper-Surface Skin Friction Coefficient for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = -2.18$, $Re = 5.7 \times 10^6$ ................ 48

5.16 Upper-Surface Displacement Thickness Comparison for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = -2.18$, $Re = 5.7 \times 10^6$ ................ 49

5.17 Comparison of Computed and Measured Velocity Profiles for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = -2.18$, $Re = 5.7 \times 10^6$ .... 49

5.18 Comparison of Pressure Coefficient Distribution for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = 1.90$, $Re = 5.7 \times 10^6$ .......... 50

5.19 Comparison of Upper-Surface Skin Friction Coefficient for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = 1.90$, $Re = 5.7 \times 10^6$ .......... 50

5.20 Upper-Surface Displacement Thickness Comparison for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = 1.90$, $Re = 5.7 \times 10^6$ .......... 51

5.21 Comparison of Pressure Coefficient Distribution for the RAE 2822 Airfoil at $M_\infty = 0.725$, $\alpha_{num} = 2.10$, $Re = 6.5 \times 10^6$ .......... 52

5.22 Comparison of Upper-Surface Skin Friction Coefficient for the RAE 2822 Airfoil at $M_\infty = 0.725$, $\alpha_{num} = 2.10$, $Re = 6.5 \times 10^6$ .......... 52

5.23 Upper-Surface Displacement Thickness Comparison for the RAE 2822 Airfoil at $M_\infty = 0.725$, $\alpha_{num} = 2.10$, $Re = 6.5 \times 10^6$ .......... 53

5.24 Comparison of Lift Coefficient vs. Boundary-Layer Space Width for the NACA 0012 Airfoil - Case A ............................................................. 54

5.25 Comparison of Drag Coefficient vs. Average Grid Spacing for the NACA 0012 Airfoil - Case A ............................................................. 54
List of Tables

4.1 Viscous-Inviscid Variable Conversion  ......................... 34

5.1 NACA 0012 Test Cases  ...................................... 40
5.2 RAE 2822 Test Cases  ........................................ 47
5.3 Force Coefficient Comparison  ................................. 55
## Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>speed of sound</td>
</tr>
<tr>
<td>$b_1, \ldots, b_6$</td>
<td>nodal variables defined by Eqns. 3.37-3.42</td>
</tr>
<tr>
<td>$c_p$</td>
<td>specific heat at constant pressure</td>
</tr>
<tr>
<td>$c_v$</td>
<td>specific heat at constant volume</td>
</tr>
<tr>
<td>$f$</td>
<td>weighting function</td>
</tr>
<tr>
<td>$h$</td>
<td>height</td>
</tr>
<tr>
<td>$m$</td>
<td>Falkner-Skan pressure gradient coefficient</td>
</tr>
<tr>
<td>$p$</td>
<td>pressure</td>
</tr>
<tr>
<td>$s$</td>
<td>dependent variable defined by Eqn. 3.21</td>
</tr>
<tr>
<td>$u$</td>
<td>velocity component tangential to body surface</td>
</tr>
<tr>
<td>$\dot{u}$</td>
<td>normalized velocity component tangential to body surface</td>
</tr>
<tr>
<td>$v$</td>
<td>velocity component normal to body surface</td>
</tr>
<tr>
<td>$v_t$</td>
<td>transpiration velocity normal to body surface</td>
</tr>
<tr>
<td>$\dot{w}$</td>
<td>intermediate dependent Dorodnitsyn variable</td>
</tr>
<tr>
<td>$x$</td>
<td>coordinate tangential to body surface</td>
</tr>
<tr>
<td>$y$</td>
<td>coordinate normal to body surface</td>
</tr>
<tr>
<td>$C_0$</td>
<td>Chapman-Rubesin constant</td>
</tr>
<tr>
<td>$C_1, \ldots, C_4$</td>
<td>variable coefficients defined by Eqns. 3.47-3.48, 3.57-3.58</td>
</tr>
<tr>
<td>$C_f$</td>
<td>coefficient of friction</td>
</tr>
<tr>
<td>$C_L$</td>
<td>coefficient of lift</td>
</tr>
<tr>
<td>$C_{D_p}$</td>
<td>coefficient of drag due to pressure</td>
</tr>
<tr>
<td>$C_{D_v}$</td>
<td>coefficient of drag due to friction</td>
</tr>
<tr>
<td>$C_D$</td>
<td>total coefficient of drag</td>
</tr>
<tr>
<td>$C_p$</td>
<td>coefficient of pressure</td>
</tr>
<tr>
<td>$F_1, F_2$</td>
<td>right-hand-side of Eqns. 3.61, 3.68</td>
</tr>
<tr>
<td>$F_T, F_S$</td>
<td>load vectors defined by Eqns. 3.66, 3.72</td>
</tr>
<tr>
<td>$H$</td>
<td>stagnation enthalpy</td>
</tr>
<tr>
<td>$K_1, \ldots, K_{10}$</td>
<td>Galerkin integrals defined by Eqns. 3.44-3.46, 3.50-3.56</td>
</tr>
</tbody>
</table>
\[ KT, KS \] stiffness matrices defined by Eqns. 3.65, 3.71
\[ L \] characteristic body dimension
\[ N \] interpolation function
\[ M \] total number of nodes
\[ Pr \] Prandtl number
\[ Re \] Reynolds number based on length
\[ T \] temperature
\[ \alpha \] similarity parameter, angle of attack in degrees
\[ \beta \] Falkner-Skan pressure-gradient parameter
\[ \delta \] boundary-layer thickness
\[ \delta^* \] displacement thickness
\[ \gamma \] ratio of specific heats
\[ \eta \] independent Dorodnitsyn transformation variable
\[ \theta \] implicitness factor, local body angle
\[ \lambda \] error tolerance
\[ \mu \] kinematic viscosity
\[ \mu_t \] eddy viscosity
\[ \xi \] independent Dorodnitsyn transformation variable
\[ \rho \] density
\[ \sigma \] viscosity parameter defined in Eqn. 3.19
\[ \tau \] dependent variable defined by Eqn. 3.26
\[ \omega \] viscous-coupling relaxation parameter
\[ \Delta \] space width
\[ \Psi \] dependent momentum similarity variable
\[ \Omega \] dependent energy similarity variable

**Superscript** | **Definition**
---|---
\[ n \] | streamwise location
\[ q \] | viscous-inviscid iteration location
\[ - \] | dimensional variable
\[ \sim \] | time-averaged variable
\[ * \] | critical condition
\[ ' \] | fluctuating variable, derivative
\[ + \] | eddy-viscosity mixing-length parameters
<table>
<thead>
<tr>
<th>Subscript</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>boundary-layer edge condition</td>
</tr>
<tr>
<td>exact</td>
<td>exact value</td>
</tr>
<tr>
<td>exp</td>
<td>experimental value</td>
</tr>
<tr>
<td>i, j, k</td>
<td>indices</td>
</tr>
<tr>
<td>num</td>
<td>numerical value</td>
</tr>
<tr>
<td>o</td>
<td>stagnation condition</td>
</tr>
<tr>
<td>rms</td>
<td>average discrete root-mean-square</td>
</tr>
<tr>
<td>w</td>
<td>wall condition</td>
</tr>
<tr>
<td>$\infty$</td>
<td>freestream condition</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The successful application of computational fluid dynamics to the design and analysis of two-dimensional airfoils in transonic flow regimes has been accomplished by a large number of researchers in the past fifteen years. Analysis of the entire flowfield around wing sections can generally be performed through the use of two main techniques. The first technique is to solve some practical form of the Navier-Stokes equations for the entire flowfield. The second technique involves the solution of the boundary-layer equations in the viscous flowfield region and either the potential or Euler equations in the inviscid flowfield region. Iteratively solving the viscous and inviscid equations while enforcing a compatibility condition then yields a solution for the entire flowfield.

The Navier-Stokes methods, while modelling most of the physical flow mechanisms and providing accurate results, require large amounts of computer time and storage. The coupled viscous-inviscid methods are generally 30-500 times faster than the Navier-Stokes methods and generate results of adequate accuracy [33]. Advances in computer hardware technology are constantly narrowing the computational advantage held by coupled methods over the Navier-Stokes methods. However, coupled methods should continue to be particularly important in an interactive airfoil design environment where near real-time flowfield analysis is desired.

The numerical solution of the classical boundary-layer equations has traditionally been accomplished through the use of finite differences [1]. The finite element method has been used only since 1972 to obtain numerical boundary-layer solutions, even though the method itself has been in existence since 1915 [11]. Traditionally, the finite element treatment of two-dimensional boundary-layer flow has involved the use of finite differences in the streamwise direction to capitalize on the parabolic nature of the boundary-layer equations. Coordinate transformations, which reduce the number of dependent variables or generate computational grids suited for finite elements, have also been used.

In 1960, A.A. Dorodnitsyn developed and applied a set of boundary-layer transformations which are well-adapted to the finite element method of solution [15]. Initially,
the transformations were used along with the method of integral relations to solve various classes of supersonic boundary-layer flows [2]. Fletcher and Fleet successfully applied the finite element method to the laminar and turbulent incompressible Dorodnitsyn form of the boundary-layer equations [3], [18]. Meade and Strong have extended the method to solve laminar compressible flow about cones and airfoils, respectively [17], [4].

Prandtl proposed that the inviscid pressure distribution could be determined to a higher-order accuracy by recalculating the potential flow while accounting for the displacement thickness of the boundary layer [5]. Perhaps the most notable transonic airfoil analysis code, employing Prandtl's direct interaction procedure to couple Green's [6] lag-entrainment boundary-layer code with an inviscid full-potential code, is the viscous Garabedian and Korn program developed by Collyer and Lock [7]. Another notable example is the GRUMFOIL program developed by Melnik, Chow, Mead, and Jameson [8]. A great deal of research has been performed to develop viscous-inviscid coupling mechanisms which do not have the disadvantages associated with the direct displacement thickness approach. The transpiration velocity boundary condition, as suggested by Lighthill [24], has been used as the viscous-inviscid coupling mechanism by Van Dalsem, Steger, and Rao with success [26].

The viscous, compressible, transonic airfoil analysis method presented in this text is based on the direct viscous-inviscid interaction of finite element boundary layer and finite difference Euler codes. The present work extends the semi-discrete Galerkin (SDG) boundary-layer formulation to include turbulent flow. A direct transpiration velocity viscous-inviscid interaction approach will be used to couple the SDG method with an innovative Euler solver (GAUSS2) which employs a shock fitting technique. The coupled codes will be used to analyze flow about a NACA 0012 and RAE 2822 airfoil for attached, turbulent, compressible flow.
Chapter 2

Finite Element Method

The finite element method is a numerical analysis technique for obtaining piecewise approximate solutions to the governing equations of a wide variety of engineering problems. The principle of the finite element method is to replace a continuum, having an infinite number of unknowns, with a discretized domain of assembled elements, having a finite number of unknowns. The unknown field variable is expressed in terms of assumed approximating functions within each element. The approximating, or interpolation, functions are defined in terms of field variable values at specific points or nodes. Each element has a prescribed number of nodes which may be on the boundary, where connections to other elements are made, or in the interior of the element. Thus, the nodal values of the field variable and the interpolation functions completely define the behavior of the field variable within the elements.

The solution to any continuum problem by the finite element method is accomplished in the following steps [10]:

- Discretize the continuum
- Select interpolation functions
- Find the element properties
- Assemble the element properties to obtain the system of equations
- Solve the system equations

2.1 Finite Element Approximation

Consider $\hat{\phi}(x)$ as an approximate, or trial, solution to the one-dimensional field variable $\phi(x)$, which can be written as

$$\hat{\phi}(x) = \sum_{i=1}^{M} N_i(x) \phi_i,$$  \hspace{1cm} (2.1)
where \( \phi_i \) are the nodal unknowns, \( N_i(x) \) are the interpolation functions, and \( M \) is the total number of nodes or nodal unknowns. The derivative of \( \phi(x) \) is approximated in this finite element representation by

\[
\frac{d\phi}{dx} = \sum_{i=1}^{M} \frac{dN_i}{dx} \phi_i.
\] (2.2)

Referring to Figure 2.1, it can be seen that when linear interpolation functions \( N_i \) are used, \( \phi(x) \) interpolates the function \( \phi(x) \) linearly over each element.

![Finite Element Representation of \( \phi(x) \)](image)

**Figure 2.1** Finite Element Representation of \( \phi(x) \)

### 2.2 Interpolation Functions

Interpolation functions are normally chosen to be locally defined polynomials within each element. It can be seen from Figure 2.2 that linear interpolation functions fall from a maximum value of one at a particular node to zero at the two neighboring nodes and are zero throughout the rest of the domain. Therefore, even though Equation 2.1 is a global equation, only two nodal unknowns and two interpolation functions make
a nonzero contribution in any particular element. As shown in Figure 2.2, the local shape functions satisfy the following conditions in each element \((e)\):

\[ N_i^{(e)}(x) = 0 \quad \text{if } x \text{ not in element } (e) \]

\[ \sum_{i=1}^{M} N_i^{(e)}(x) = 1 \quad \text{for all } x \text{ in element } (e) \]

It is also necessary that the interpolation functions be chosen in such a way that the

\[ \begin{align*}
\text{Figure 2.2 Linear Interpolation Function } N_i^{(e)}(x) \\
\end{align*} \]

field variable \(\phi(x)\) and any of its derivatives, up to one order less than the highest order derivative appearing in the weak form of the equation, be continuous at the element boundaries [11]. The linear interpolation functions can be determined by using Lagrange polynomials and, for elements A and B, take the form

\[ \begin{align*}
N_{i-1}^A &= \frac{x - x_i}{x_{i-1} - x_i} \\
N_i^A &= \frac{x - x_{i-1}}{x_i - x_{i-1}} \\
N_i^B &= \frac{x - x_{i+1}}{x_i - x_{i+1}} \\
N_{i+1}^B &= \frac{x - x_i}{x_{i+1} - x_i}.
\end{align*} \]  

(2.3)
2.3 Method of Weighted Residuals

The method of weighted residuals is a technique for obtaining approximate solutions to partial differential equations. The method of weighted residuals is one of many approaches, namely the direct approach, the variational approach, or the energy-balance approach, used to determine the finite element matrix equations which express the properties of individual elements.

Applying the method of weighted residuals involves basically two steps. The first step is to assume the general functional behavior of the dependent field variable so that the given differential equation and boundary conditions are approximately satisfied. A residual, which is required to vanish over the entire solution domain, results when the approximation is substituted into the original differential equation and boundary conditions. The second step is to solve the equation(s) resulting from step one for a particular function.

Consider finding an approximate functional representation for the field variable \( \phi \) which is governed by the differential equation

\[
L(\phi) - f = 0, \tag{2.4}
\]

where \( L \) represents the differential operator and \( f \) is a known function of the independent variables. The differential equation resides in the domain \( D \) bounded by the surface \( S \), where proper boundary conditions are prescribed. The unknown exact solution for \( \phi \) can be approximated by \( \phi \) as

\[
\phi \approx \hat{\phi} = \sum_{i=1}^{M} N_i \phi_i, \tag{2.5}
\]

where \( N_i \) are the assumed functions and \( \phi_i \) are the unknown parameters. The \( M \) functions \( N_i \) are usually chosen to satisfy the global boundary conditions.

When \( \hat{\phi} \) is substituted into Equation 2.4, a residual \( R \) results from the approximation and is given by

\[
R = L(\hat{\phi}) - f. \tag{2.6}
\]

The method of weighted residuals seeks to determine the \( M \) unknowns \( \phi_i \) in such a way that the error \( R \) over the entire solution domain is minimized. This is accomplished by forming a weighted average of the error and specifying that this weighted average vanish over the solution domain. Therefore, choosing \( M \) linearly independent weighting functions \( W_j \) and insisting that if

\[
\int_{D} W_j [L(\hat{\phi}) - f] dD = \int_{D} W_j R dD = 0, \quad j = 1, 2, ..., M \tag{2.7}
\]
then $R \approx 0$. The form of error distribution principle used in Equation 2.7 is dependent on the choice of weighting function. There are a variety of weighted-residual techniques which can be employed; the most popular error distribution principle is the Galerkin Method. The Bubnov-Galerkin (classical Galerkin) method uses the interpolation functions $N_j$ as the weighting functions $W_j$, while the Petrov-Galerkin method specifies $W_j$ as some modification of $N_j$.

### 2.3.1 Petrov-Galerkin Method

The Petrov-Galerkin method is used in applications of the method of weighted residuals when specific requirements must be imposed on the finite element solution. The weighting function in this method is represented by

$$W_j = P_j,$$  \hspace{1cm} (2.8)

where $P_j$ is an analytic function similar to the Bubnov-Galerkin interpolation function $N_j$ but with additional terms or factors to impose the specific solution requirements. The use of the Petrov-Galerkin method has been based in part on its ability to produce asymmetric weighting functions which force diagonal dominance of the finite element matrix equations and reduce the oscillatory solution behavior in convection-dominated fluid flows [11].

### 2.4 Semi-Discrete Galerkin Method

The semi-discrete Galerkin method is a hybrid finite element and finite difference numerical-analysis technique which uses a finite element representation for the spatial variables and models the time or time-like variables by finite differences. The semi-discrete Galerkin (SDG) method is best demonstrated by its application to the one-dimensional, unsteady, nondimensionalized Heat Conduction equation:

$$\frac{\partial \phi}{\partial t} - \frac{\partial^2 \phi}{\partial x^2} = 0$$  \hspace{1cm} (2.9)

The initial and boundary conditions for $\phi(x,t)$, over the interval $0 \leq x \leq 1$, are given by

$$\phi(x,0) = \phi_0(x), \quad \phi(0,t) = 0, \quad \phi(1,t) = 1.$$  \hspace{1cm} (2.10)
Substituting an approximate solution for \( \phi \),

\[
\phi(x, t) = \sum_{i=1}^{M} N_i(x) \phi_i(t),
\]

(2.11)

into Equation 2.9 and applying the method of weighted residuals gives

\[
\int_0^1 W_j \left( \frac{\partial \phi}{\partial t} - \frac{\partial^2 \phi}{\partial x^2} \right) dx = 0,
\]

(2.12)

where \( j = 1, 2, ..., M \). The classical Galerkin method can be used by taking the weighting function as the interpolation function:

\[
W_j = N_j(x)
\]

(2.13)

Therefore, Equation 2.12 becomes

\[
\int_0^1 N_j \frac{\partial \phi}{\partial t} dx - \int_0^1 N_j \frac{\partial^2 \phi}{\partial x^2} dx = 0.
\]

(2.14)

In order to satisfy the continuity requirements for linear interpolation functions, it is necessary to reduce the second partial differential by applying the Green-Gauss theorem. The Green-Gauss theorem in one-dimension is simply an integration by parts which gives

\[
\int_0^1 N_j \frac{\partial \phi}{\partial t} dx = - \left[ \int_0^1 N_j \frac{\partial \phi}{\partial x} dx \right]_0^1 + \left[ N_j \frac{\partial \phi}{\partial x} \right]_0^1.
\]

(2.15)

The last term in Equation 2.15 represents the natural boundary conditions. If Neumann boundary conditions are present, it would be appropriate to replace the boundary term with the given condition. However, Dirichlet boundary conditions are specified so the boundary term remains incorporated in the governing equation. Substituting the approximate solutions for \( \dot{\phi} \) and \( \frac{\partial \phi}{\partial x} \) into Equation 2.15 produces the following finite element equation:

\[
\int_0^1 N_j \sum_{i=1}^{M} N_i \frac{d\phi_i}{dt} dx = - \left[ \int_0^1 \frac{dN_j}{dx} \frac{dN_i}{dx} \phi_i dx \right]_0^1 + \left[ N_j \frac{dN_i}{dx} \frac{\partial \phi_i}{\partial x} \right]_0^1
\]

(2.16)

Rewriting in a more convenient form by introducing the coefficients,

\[
A_{1ji} = \int_0^1 N_j N_i dx, \quad A_{2ji} = \int_0^1 \frac{dN_j}{dx} \frac{dN_i}{dx} dx,
\]
gives
\[ \sum_{i=1}^{M} A_{1ji} \frac{d\phi_i}{dt} = -\sum_{i=1}^{M} A_{2ji} \phi_i + N_j \sum_{i=1}^{M} \frac{dN_i}{dx} \phi_i \bigg|_0^1. \] (2.17)

A time-independent discretization in \( x \) (uniform) allows the solution of \( A_1 \) and \( A_2 \) to be obtained prior to the rest of the solution process. The elements of matrices \( A_1 \) and \( A_2 \) can be solved exactly by Gaussian quadrature if lower-order polynomials are chosen as interpolation functions. Also, the matrices become tridiagonal if the interpolation functions are chosen to be linear. Modifying \( A_2 \) to absorb the natural boundary conditions and renaming as \( A_3 \) result in the following system of ODE's:
\[ \sum_{i=1}^{M} A_{1ji} \frac{d\phi_i}{dt} = -\sum_{i=1}^{M} A_{3ji} \phi_i \] (2.18)

The boundary conditions in nodal form are given by
\[ \phi_1 = 0, \quad \phi_M = 1. \]

To utilize the parabolic nature of the governing equations, a marching routine is invoked by using a finite difference discretization for \( \frac{d\phi_i}{dt} \),
\[ \frac{\phi_i^{n+1} - \phi_i^n}{t^{n+1} - t^n} = \frac{\Delta \phi_i^{n+1}}{\Delta t}, \] (2.19)
where \( n \) denotes the time level. The resulting \( \Delta \phi_i^{n+1} \) may then be solved for using the theta method:
\[ \sum_{i=1}^{M} A_{1ji} \Delta \phi_i^{n+1} = -\Delta t \left[ \theta \sum_{i=1}^{M} A_{3ji} \phi_i^{n+1} + (1 - \theta) \sum_{i=1}^{M} A_{3ji} \phi_i^n \right] \] (2.20)

The value of \( \theta \) controls the degree of implicitness. A number of schemes which depend on the value of theta may be employed:
\[ \theta = 0.0 \quad \text{fully-explicit Euler forward} \]
\[ \theta = 1.0 \quad \text{fully-implicit Euler backward} \]
\[ \theta = 0.5 \quad \text{second-order accurate Crank-Nicholson} \]

Choosing the Crank-Nicholson technique and linearizing Equation 2.20 converts the system of ordinary differential equations into a system of linear algebraic equations given by
\[ \sum_{i=1}^{M} (A_{1ji} + \theta \Delta t A_{3ji}) \Delta \phi_i^{n+1} = -\Delta t \sum_{i=1}^{M} A_{3ji} \phi_i^n. \] (2.21)
A detailed discussion of the linearization is given in Section 3.3.

The right-hand side of Equation 2.21 is known for a given time increment $\Delta t$ and initial condition $\phi_i^n$. The left-hand side array forms a tridiagonal matrix of size $(M \times M)$ that may be efficiently solved by the Thomas Algorithm which is ideally suited for equations of this type [13]. The Thomas Algorithm requires an order of $M$ operations ($O(M)$) which is more efficient than the $O(M^2)$ operations necessary for Gaussian elimination. After the system of equations has been solved, $\Delta \phi_i^{n+1}$ is added to the known value $\phi_i^n$ to give $\phi_i^{n+1}$. The marching routine continues as $\phi_i^{n+1}$ is used in the right-hand side of the equation to solve for $\Delta \phi_i^{n+1}$ at the next time step. The time step is varied to obtain a desired accuracy, instead of performing iterations, which results in a computationally efficient algorithm.

### 2.5 Group Finite Element Method

Finite element treatment of the nonlinear convective terms, which are present in most flow problems, is traditionally accomplished by the introduction of a separate approximate solution for each contributing variable in the nonlinear terms. The group finite element method permits the nonlinear convective terms to be represented without introducing a separate approximate solution for each variable. Since the added connectivity of separate approximate solutions is avoided, the group finite element formulation can model nonlinear terms and avoid having products of nodal values over all connected nodes in a particular element [12].

The group finite element method consists of transforming any convective terms into a divergence form and then employing supplementary solutions for these terms. The group method is best demonstrated by its application to the one-dimensional, unsteady, nondimensionalized Burger’s equation:

$$\frac{\partial \phi}{\partial t} + \phi \frac{\partial \phi}{\partial x} = \frac{1}{Re} \frac{\partial^2 \phi}{\partial x^2}$$

(2.22)

Transforming the convective term into a divergence form,

$$\phi \frac{\partial \phi}{\partial x} = \frac{1}{2} \frac{\partial (\phi^2)}{\partial x},$$

(2.23)

and substituting into Equation 2.22 gives

$$\frac{\partial \phi}{\partial t} + \frac{1}{2} \frac{\partial (\phi^2)}{\partial x} = \frac{1}{Re} \frac{\partial^2 \phi}{\partial x^2}.$$  

(2.24)
The supplemental approximate solution is then introduced for the group $\phi^2$ as

$$\phi^2(x,t) = \sum_{i=1}^{M} N_i(x) \phi_i^2(t).$$  \hspace{1cm} (2.25)

After substituting the supplemental approximate solution into Equation 2.24 and applying Galerkin’s method with linear elements on a uniform grid, the following system of ODE’s is produced:

$$\frac{\partial}{\partial t} \left( \frac{1}{6} \phi_{i-1} + \frac{2}{3} \phi_i + \frac{1}{6} \phi_{i+1} \right) + \frac{1}{2} \left( \phi_{i-1} + \phi_{i+1} \right) \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} = \frac{1}{Re} \left( \frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{\Delta x^2} \right)$$  \hspace{1cm} (2.26)

The traditional finite element formulation produced the following similar system of ODE’s:

$$\frac{\partial}{\partial t} \left( \frac{1}{6} \phi_{i-1} + \frac{2}{3} \phi_i + \frac{1}{6} \phi_{i+1} \right) + \frac{1}{3} \left( \phi_{i-1} + \phi_i + \phi_{i+1} \right) \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} = \frac{1}{Re} \left( \frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{\Delta x^2} \right)$$  \hspace{1cm} (2.27)

As seen in the second term of Equations 2.26 and 2.27, the group method produces a computationally more economical finite element form of Burger’s equation by reducing the nodal connectivity of the convective term. The group finite element method becomes progressively more economical as the order of nonlinearity or the number of dimensions increases and generally produces a more accurate finite element scheme [12].
Chapter 3

Boundary Layer Formulation

The boundary-layer approximation of the Navier-Stokes equations is valid if the viscous flow region, prior to separation from a body, is assumed to be thin. That is, the boundary-layer thickness is much smaller than the characteristic length of the body in question. Prandtl proposed that the following assumptions could be made about thin shear layer flows [14]:

- negligible body forces
- negligible normal viscous stresses
- negligible normal pressure gradient
- normal velocity $\ll$ tangential velocity
- tangential velocity gradients $\ll$ normal velocity gradients

3.1 Governing Equations

The equations of motion for steady, compressible, and turbulent boundary-layer flow are given in terms of dimensional variables:

Continuity:

$$\frac{\partial}{\partial x} (\bar{\rho} \bar{u}) + \frac{\partial}{\partial y} (\bar{\rho} \bar{v}) = 0$$  \hspace{1cm} (3.1)

Momentum:

$$\bar{\rho} \bar{u} \frac{\partial \bar{u}}{\partial x} + \bar{\rho} \bar{v} \frac{\partial \bar{u}}{\partial y} = - \frac{\partial \bar{p}}{\partial x} + \frac{\partial}{\partial y} \left[ (\bar{\mu} + \bar{\mu}_t) \frac{\partial \bar{u}}{\partial y} \right]$$  \hspace{1cm} (3.2)

Energy:

$$\bar{\rho} \bar{u} \frac{\partial \bar{H}}{\partial x} + \bar{\rho} \bar{v} \frac{\partial \bar{H}}{\partial y} = \frac{1}{Pr} \frac{\partial}{\partial y} \left[ (\bar{\mu} + \bar{\mu}_t) \frac{\partial \bar{H}}{\partial y} \right] + \left( 1 - \frac{1}{Pr} \right) \frac{\partial}{\partial y} \left[ (\bar{\mu} + \bar{\mu}_t) \frac{\partial \left( \bar{u}^2 \right)}{\partial y} \right]$$  \hspace{1cm} (3.3)
The Prandtl number is assumed constant, and the perfect gas assumption serves as the equation of state of the fluid. The gross effects of turbulence in the boundary layer are accounted for using the eddy-viscosity model

\[
\tilde{\mu}_t \frac{\partial \tilde{u}}{\partial \tilde{y}} = -\tilde{\rho} \tilde{u} \tilde{v}',
\]

where the tilde denotes average values and the prime denotes fluctuating values [9]. Dropping the tilde and prime notation, the dimensional velocity components are denoted as \(\bar{u}\) in the local \(\bar{x}\) coordinate and \(\bar{v}\) in the local \(\bar{y}\) coordinate. The dimensional pressure, density, laminar viscosity, turbulent viscosity, and total enthalpy are shown as \(\bar{p}, \bar{\rho}, \bar{\mu}, \bar{\mu}_t,\) and \(\bar{H},\) respectively.

The initial conditions for velocity and temperature are provided at \(\bar{x} = \bar{x}_0\) and represented by

\[
\bar{u}(\bar{x}_0, \bar{y}) = \bar{u}_o(\bar{y}), \quad \bar{v}(\bar{x}_0, \bar{y}) = \bar{v}_o(\bar{y}), \quad \text{and} \quad \bar{T}(\bar{x}_0, \bar{y}) = \bar{T}_o(\bar{y}),
\]

where \(\bar{u}_o(\bar{y}), \bar{v}_o(\bar{y}),\) and \(\bar{T}_o(\bar{y})\) are known quantities.

Boundary conditions are prescribed at \(\bar{y} = 0\) and \(\bar{y} = \infty\). The boundary conditions for velocity are obtained by noting the no-slip condition at the surface of the body and that the tangential velocity approaches the magnitude of the inviscid velocity at the edge of the boundary layer. The velocity boundary conditions are given by

\[
\bar{u}(\bar{x}, 0) = \bar{v}(\bar{x}, 0) = 0 \quad \text{and} \quad \bar{u}(\bar{x}, \infty) = \bar{u}_e(\bar{x}),
\]

where \(\bar{u}_e(\bar{x})\) is the velocity at the edge of the boundary layer obtained from a solution of the inviscid equations of motion. The heat-transfer boundary conditions for temperature are given by

\[
\bar{T}(\bar{x}, 0) = \bar{T}_w(\bar{x}) \quad \text{and} \quad \bar{T}(\bar{x}, \infty) = \bar{T}_e(\bar{x}),
\]

where \(w\) and \(e\) denote wall and boundary-layer edge conditions, respectively. If an adiabatic wall is prescribed, the following boundary conditions for temperature are required:

\[
\frac{\partial \bar{T}}{\partial \bar{x}} \bigg|_{\bar{y}=0} = 0 \quad \text{and} \quad \bar{T}(\bar{x}, \infty) = \bar{T}_e(\bar{x})
\]

The equations of motion, Equations 3.1-3.3, can be nondimensionalized by employing a characteristic length \(L,\) critical speed of sound \(a^*\), and free-stream density
The dimensionless variables are defined as follows:

\[ x = \frac{\bar{x}}{L}, \quad y = \frac{\bar{y}}{L}, \quad u = \frac{\bar{u}}{a^*}, \quad v = \frac{\bar{v}}{a^*}, \]

\[ p = \frac{\bar{p}}{\bar{p}_\infty a^{*2}}, \quad \rho = \frac{\bar{\rho}}{\bar{\rho}_\infty}, \quad \mu = \frac{\bar{\mu}}{\bar{\rho}_\infty a^* L}, \quad H = \frac{c_p T + \frac{1}{2}(\bar{u}^2 + \bar{v}^2)}{a^{*2}} \]  

(3.9)

After substitution of the nondimensionalized parameters and simplifying, the continuity equation becomes

\[ \frac{\partial}{\partial x} (\rho u) + \frac{\partial}{\partial y} (\rho v) = 0. \]  

(3.10)

Since pressure is only a function of \( x \) and is transmitted unchanged through the boundary layer,

\[ \frac{\partial p}{\partial x} = \frac{dp}{dx} = \frac{dp_e}{dx}, \]

where \( p_e \) is the dimensionless pressure at the edge of the boundary layer. Substituting the pressure term along with the nondimensionalized parameters into the momentum equation gives

\[ \rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = -\frac{dp_e}{dx} + \frac{\partial}{\partial y} \left[ (\mu + \mu_t) \frac{\partial u}{\partial y} \right]. \]

(3.11)

Similarly, the energy equation takes the form

\[ \rho u \frac{\partial H}{\partial x} + \rho v \frac{\partial H}{\partial y} = \frac{1}{Pr} \frac{\partial}{\partial y} \left[ (\mu + \mu_t) \frac{\partial H}{\partial y} \right] + \left( 1 - \frac{1}{Pr} \right) \frac{\partial}{\partial y} \left[ (\mu + \mu_t) \frac{\partial u^2}{\partial y} \right]. \]

(3.12)

### 3.2 Traditional Dorodnitsyn Formulation

A. A. Dorodnitsyn applied the method of integral relations to the compressible boundary-layer equations and transformed them into a form resembling the incompressible equations [15]. Dorodnitsyn used the following expressions to transform the body-normal \( x \) and \( y \) coordinate system to a \( \xi \) and \( \eta \) computational plane and smooth both velocity and density over the boundary layer:

\[ \xi = \frac{1}{u_\infty p_\infty} \int_0^x u_e p_e dx \]  

(3.13)

\[ \eta = \frac{u_e}{(u_\infty \mu_\infty)^{\frac{1}{2}}} \int_0^y \rho dy \]  

(3.14)

The Dorodnitsyn formulation utilized a normalized velocity \( \hat{u} \)

\[ \hat{u} = \frac{u}{u_e}, \]  

(3.15)
and an intermediate variable \( \dot{\omega} \) given by

\[
\dot{\omega} = \hat{u} \frac{u_\infty p_\infty}{u_e p_e} \frac{\partial \eta}{\partial x} + \left( \frac{u_\infty}{\mu_\infty} \right)^{\frac{1}{2}} \frac{v}{u_e} \frac{T_\infty}{T}.
\]  

(3.16)

Employing the Dorodnitsyn relations results in the following form of the continuity equation:

\[
\frac{\partial \dot{u}}{\partial \xi} + \frac{\partial \dot{\omega}}{\partial \eta} = 0
\]  

(3.17)

It is assumed that the variation of laminar viscosity across the boundary layer can be represented by the linear relation

\[
\frac{\mu}{\mu_\infty} = C_0 \frac{T}{T_\infty},
\]  

(3.18)

where \( C_0 \) is the Chapman-Rubesin constant. A full explanation of the assumption is given in Reference [16]. Applying the Chapman-Rubesin relation, ideal-gas law, Dorodnitsyn relations, and a viscosity parameter \( \sigma \)

\[
\sigma = 1 + \frac{\mu_i}{\mu},
\]  

(3.19)

results in the following form of the momentum equation:

\[
\hat{u} \frac{\partial \dot{u}}{\partial \xi} + \hat{\omega} \frac{\partial \dot{u}}{\partial \eta} = \frac{1}{u_e} \frac{du_e}{d\xi} \left( 1 - \dot{u}^2 \right) + C_0 \frac{\partial}{\partial \eta} \left( \frac{\sigma}{\partial \eta} \right) \]  

(3.20)

Using a new variable \( s \) which relates enthalpy and the ratio of specific heats \( \gamma = c_p/c_v \)

\[
s = \left[ 1 - 2H \left( \frac{\gamma - 1}{\gamma + 1} \right) \right] \left( \frac{1}{u_e^2} \right),
\]  

(3.21)

with the Dorodnitsyn relations results in the following form of the energy equation:

\[
\hat{u} \frac{\partial s}{\partial \xi} + \hat{\omega} \frac{\partial s}{\partial \eta} + \frac{1}{u_e} \frac{d u_e}{d \xi} \dot{s} = C_0 \frac{\partial}{\partial \eta} \left( \sigma \frac{\partial \dot{u}}{\partial \eta} \right) + 2C_0 \frac{(\gamma - 1)}{\gamma + 1} \frac{1}{Pr} \frac{1}{\sigma} \frac{\partial}{\partial \eta} \left( \hat{u} \frac{\partial \dot{u}}{\partial \eta} \right) \]  

(3.22)

The Dorodnitsyn formulation, representing the two-dimensional, compressible, boundary-layer equations in incompressible form with respect to the independent variables, \( \xi \) and \( \eta \), is therefore given by Equations 3.17, 3.20, and 3.22. An integral form of these equations can be obtained by applying the method of weighted residuals.
After using a general weighting function \( f(\hat{u}) \) and summing the products of Equation 3.17 \( \times f \) and Equation 3.20 \( \times \frac{df}{d\hat{u}} \), the first Dorodnitsyn equation appears as

\[
\frac{\partial(\hat{u}f)}{\partial \xi} + \frac{\partial(\hat{w}f)}{\partial \eta} = \frac{1}{u_e} \frac{du_e}{d\xi} \left(1 - \hat{u}^2\right) \frac{df}{d\hat{u}} + C_0 \frac{df}{d\hat{u}} \frac{\partial}{\partial \eta} \left(\sigma \frac{\partial \hat{u}}{\partial \eta}\right).
\]  
(3.23)

Integrating the above equation with respect to \( \eta \) and reducing gives the following:

\[
\frac{d}{d\xi} \left[ \hat{u}f \right]_0^\infty + \hat{w}f \bigg|_0^\infty = \frac{1}{u_e} \frac{du_e}{d\xi} \int_0^\infty \left(1 - \hat{u}^2\right) \frac{df}{d\hat{u}} d\eta + C_0 \int_0^\infty \frac{df}{d\hat{u}} \frac{\partial}{\partial \eta} \left(\sigma \frac{\partial \hat{u}}{\partial \eta}\right) d\eta
\]  
(3.24)

Applying the Crocco transformation to the integral equation changes the independent variable \( \eta \) to \( \hat{u} \) [17]. The transformation makes use of the nondimensionalized shear stress and is given by

\[
\hat{u} = \int_0^\eta \tau d\eta,
\]  
(3.25)

or

\[
\tau = \frac{\partial \hat{u}}{\partial \eta}.
\]  
(3.26)

Substituting the Crocco transformation into Equation 3.24 and noting that the limits of integration are changed from \( \eta = 0 \rightarrow \infty \) to \( \hat{u} = 0 \rightarrow 1 \), gives

\[
\frac{d}{d\xi} \left[ \hat{u}f \right]_0^1 + \hat{w}f \bigg|_0^1 = \frac{1}{u_e} \frac{du_e}{d\xi} \int_0^1 \left(1 - \hat{u}^2\right) \frac{df}{d\hat{u}} \tau d\hat{u} + C_0 \int_0^1 \frac{df}{d\hat{u}} \frac{\partial}{\partial \eta} \left(\sigma \tau \frac{\partial \hat{u}}{\partial \eta}\right) d\hat{u}.
\]  
(3.27)

Assuming that there is no surface injection, \( v \), and hence \( \hat{w} \), is zero at the wall. Therefore, if \( f(\hat{u}) \) is chosen to vanish at the edge of the boundary layer, the traditional Dorodnitsyn formulation for the first integral equation reduces to

\[
\frac{d}{d\xi} \left[ \hat{u}f \right]_0^1 + \hat{w}f \bigg|_0^1 = \frac{1}{u_e} \frac{du_e}{d\xi} \int_0^1 \left(1 - \hat{u}^2\right) \frac{df}{d\hat{u}} \tau d\hat{u} + C_0 \int_0^1 \frac{df}{d\hat{u}} \frac{\partial}{\partial \eta} \left(\sigma \tau \frac{\partial \hat{u}}{\partial \eta}\right) d\hat{u}.
\]  
(3.28)

After using a general weighting function \( f(\hat{u}) \) and summing the products of Equation 3.17 \( \times s \times f \), Equation 3.20 \( \times s \times \frac{df}{d\hat{u}} \), and Equation 3.22 \( \times f \), the second Dorodnitsyn equation appears as

\[
\frac{\partial(\hat{u}fs)}{\partial \xi} + \frac{\partial(\hat{w}fs)}{\partial \eta} = -2 \frac{1}{d\xi} \frac{du_e}{d\xi} \hat{u}fs + \frac{1}{u_e} \frac{du_e}{d\xi} \left(1 - \hat{u}^2\right) \frac{df}{d\hat{u}} s
\]  
\[+ C_0 \frac{df}{d\hat{u}} \frac{\partial}{\partial \eta} \left(\sigma \frac{\partial \hat{u}}{\partial \eta}\right) s + \frac{C_0}{Pr} f \frac{\partial}{\partial \eta} \left(\sigma \frac{\partial \hat{u}}{\partial \eta}\right)
\]  
\[+ 2C_0 \frac{\gamma - 1}{\gamma + 1} \left(\frac{1}{Pr} - 1\right) f \frac{\partial}{\partial \eta} \left(\hat{u} \sigma \frac{\partial \hat{u}}{\partial \eta}\right).
\]  
(3.29)
Integrating the above equation with respect to η and reducing gives the following:

\[
\frac{d}{d\xi} \int_0^{\infty} \hat{u}f s d\eta + \hat{\dot{u}} s \bigg|_0^{\infty} = -2 \frac{1}{u_e} \frac{du_e}{d\xi} \int_0^{\infty} \hat{u}f s d\eta + \frac{1}{u_e} \frac{du_e}{d\xi} \int_0^{\infty} (1 - \hat{u}^2) \frac{df}{d\hat{u}} s d\eta \\
+ C_0 \int_0^{\infty} \frac{df}{d\hat{u}} \frac{\partial}{\partial \eta} (\frac{\sigma}{\sigma \hat{u}} \frac{\partial \hat{u}}{\partial \eta}) s d\eta + \frac{C_0}{Pr} \int_0^{\infty} f \frac{\partial}{\partial \eta} (\frac{\sigma}{\sigma \hat{u}} \frac{\partial s}{\partial \eta}) d\eta \\
+ 2C_0 \frac{(\gamma - 1)}{(\gamma + 1)} \frac{1}{Pr - 1} \int_0^{\infty} f \frac{\partial}{\partial \eta} (\hat{u} \frac{\partial \hat{u}}{\partial \eta}) d\eta \tag{3.30}
\]

Using the Green-Gauss theorem to reduce the second-order derivative of s with respect to η, and again applying the Crocco transformation while noting the previous assumptions for f(\hat{u}) and \hat{\dot{u}}, gives the modified Dorodnitsyn formulation for the second integral equation:

\[
\frac{d}{d\xi} \int_0^{1} \hat{u} \frac{f s}{\tau} d\hat{u} = -2 \frac{1}{u_e} \frac{du_e}{d\xi} \int_0^{1} \hat{u} \frac{f s}{\tau} d\hat{u} + \frac{1}{u_e} \frac{du_e}{d\xi} \int_0^{1} (1 - \hat{u}^2) \frac{df}{d\hat{u}} s d\hat{u} \\
+ C_0 \int_0^{1} \frac{df}{d\hat{u}} s \frac{\partial}{\partial \eta} (\sigma \hat{u}) d\hat{u} + \frac{C_0}{Pr} f \sigma \hat{u} \frac{\partial s}{\partial \eta} \bigg|_0^{1} - \frac{C_0}{Pr} \int_0^{1} \frac{df}{d\hat{u}} s \frac{\partial \hat{u}}{\partial \eta} d\hat{u} \\
+ 2C_0 \frac{(\gamma - 1)}{(\gamma + 1)} \frac{1}{Pr - 1} \int_0^{1} \hat{u} f \frac{\partial}{\partial \eta} (\sigma \tau) d\hat{u} \\
+ 2C_0 \frac{(\gamma - 1)}{(\gamma + 1)} \frac{1}{Pr - 1} \int_0^{1} f \sigma \tau d\hat{u} \tag{3.31}
\]

In order to permit a more physically-oriented solution procedure, it is desirable to use the independent variable x in place of \xi. Making use of the fact that

\[
\frac{d}{d\xi} = \frac{d}{dx} \frac{dx}{d\xi}, \quad \text{where} \quad \frac{dx}{d\xi} = \frac{u_{\infty} p_{\infty}}{u_e p_e},
\]

Equations 3.28 and 3.31 become

\[
\frac{d}{dx} \int_0^{1} \hat{u} \frac{f \frac{1}{\tau}}{\tau} d\hat{u} = \frac{1}{u_e} \frac{du_e}{dx} \int_0^{1} (1 - \hat{u}^2) \frac{df}{d\hat{u}} \frac{1}{\tau} d\hat{u} + \frac{C_0}{u_{\infty} p_{\infty}} \int_0^{1} \frac{df}{d\hat{u}} \frac{\partial}{\partial \hat{u}} (\sigma \tau) d\hat{u} \tag{3.32}
\]
and

\[
\frac{d}{dx} \int_0^{1} \hat{u} \frac{f \frac{s}{\tau}}{\tau} d\hat{u} = -2 \frac{1}{u_e} \frac{du_e}{dx} \int_0^{1} \hat{u} \frac{f \frac{s}{\tau}}{\tau} d\hat{u} \\
+ \frac{1}{u_e} \frac{du_e}{dx} \int_0^{1} (1 - \hat{u}^2) \frac{df}{d\hat{u}} \frac{s}{\tau} d\hat{u} + \frac{C_0}{u_{\infty} p_{\infty}} \int_0^{1} \frac{df}{d\hat{u}} \frac{\partial}{\partial \hat{u}} (\sigma \tau) d\hat{u} \\
+ \frac{C_0}{Pr} \frac{u_e p_e}{u_{\infty} p_{\infty}} f \sigma \tau \frac{\partial s}{\partial \hat{u}} \bigg|_0^{1} - \frac{C_0}{Pr} \frac{u_e p_e}{u_{\infty} p_{\infty}} \int_0^{1} \frac{df}{d\hat{u}} \frac{\partial \hat{u}}{\partial \eta} d\hat{u}.
\]


\[ + \ 2C_0 \left( \frac{\gamma - 1}{\gamma + 1} \right) \left( \frac{1}{Pr} - 1 \right) \frac{u_{\infty} L}{p_{\infty}} \int_0^1 \hat{u} \hat{f} \frac{\partial}{\partial \hat{u}} (\sigma \tau) \, d\hat{u} \]

\[ + \ 2C_0 \left( \frac{\gamma - 1}{\gamma + 1} \right) \left( \frac{1}{Pr} - 1 \right) \frac{u_{\infty} L}{p_{\infty}} \int_0^1 f \sigma \tau \, d\hat{u}, \quad (3.33) \]

respectively.

The boundary conditions for \( \tau \) and \( s \) in terms of the Dorodnitsyn formulation are given by

\[ \tau(1) = 0, \quad s(0) = s_w = \left( 1 - \frac{T_w}{T_0} \right) \frac{1}{u_{\infty}^2}, \quad \text{and} \quad s(1) = 0, \quad (3.34) \]

in the case of heat transfer at the wall. For an adiabatic wall, the boundary conditions for \( s \) are specified as

\[ \left. \frac{\partial s}{\partial \hat{u}} \right|_{\hat{u}=0} = 0 \quad \text{and} \quad s(1) = 0. \quad (3.35) \]

Numerical experiments have shown that the solution for \( \tau \) was generally inferior when a wall boundary condition was imposed; therefore, the wall boundary condition available from Equation 3.2 is not used [18].

It has been shown that the Dorodnitsyn formulation reduces the nonlinear partial differential equations governing two-dimensional, compressible, and turbulent flow to a set of uncoupled integral equations by seeking the proper weighted combinations of variables and equations. The integral equations are explicitly independent of density and represented in the independent variables \( x \) and \( \hat{u} \). The transformation from a body-normal coordinate system \( (x, y) \) to a computational plane \( (x, \hat{u}) \) is depicted in Figures 3.1 and 3.2. One benefit of this coordinate transformation is that the infinite domain in the \( y \) direction has been replaced by a finite domain in the \( \hat{u} \) direction. A greater benefit is that the uniform grid in \( \hat{u} \) automatically captures downstream boundary-layer growth and spatially provides high resolution near the wall.

### 3.3 Semi-Discrete Galerkin Formulation

In order to apply the semi-discrete Galerkin method to the integral Dorodnitsyn equations developed above, the general weighting function \( f(\hat{u}) \) must be replaced by a set of linearly independent functions \( f_j(\hat{u}) \). The weighting function is introduced as

\[ f_j(\hat{u}) = (1 - \hat{u}) N_j(\hat{u}), \quad j = 1, 2, \ldots, M \quad (3.36) \]
Figure 3.1 Airfoil Coordinate System

Figure 3.2 Computational Plane
where \( N_j(\hat{u}) \) is the linear interpolation function at a particular node \( j \), and \( M \) is the total number of nodes. The Petrov-Galerkin finite element method is utilized by introducing the term \((1 - \hat{u})\) to satisfy the requirement that \( f_j(\hat{u}) \) equals zero at the outer edge of the boundary layer. The trial solutions for the dependent variables \( \frac{1}{\tau} \), \( \tau \), \( s \), \( \frac{s}{\tau} \), \( \sigma \), and \( \sigma \tau \) are introduced using the group finite element method and take the following form:

\[
\frac{1}{\tau} = \frac{1}{(1 - \hat{u})} \sum_{i=1}^{M} N_i(\hat{u}) b_{1i}(x) \tag{3.37}
\]

\[
\tau = (1 - \hat{u}) \sum_{i=1}^{M} N_i(\hat{u}) b_{2i}(x) \tag{3.38}
\]

\[
s = (1 - \hat{u}) \sum_{i=1}^{M} N_i(\hat{u}) b_{3i}(x) \tag{3.39}
\]

\[
\frac{s}{\tau} = \sum_{i=1}^{M} N_i(\hat{u}) b_{4i}(x) \tag{3.40}
\]

\[
\sigma = \sum_{i=1}^{M} N_i(\hat{u}) b_{5i}(x) \tag{3.41}
\]

\[
\sigma \tau = (1 - \hat{u}) \sum_{i=1}^{M} N_i(\hat{u}) b_{6i}(x) \tag{3.42}
\]

Substituting the approximate solutions and the weighting function into Equation 3.32 produces the following finite element equation:

\[
\frac{d}{dx} \int_0^1 \hat{u} [(1 - \hat{u}) N_j] \left[ \frac{1}{(1 - \hat{u})} \sum_{i=1}^{M} N_i b_{1i} \right] d\hat{u} \\
= \frac{1}{u_e} \frac{du_e}{dx} \int_0^1 (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ \frac{1}{(1 - \hat{u})} \sum_{i=1}^{M} N_i b_{1i} \right] d\hat{u} \\
+ C_0 \frac{u_e p_e}{u_\infty p_\infty} \int_0^1 [(1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j] \left[ (1 - \hat{u}) \sum_{i=1}^{M} \frac{dN_i}{d\hat{u}} b_{6i} - \sum_{i=1}^{M} N_i b_{6i} \right] d\hat{u} \tag{3.43}
\]

Removing all terms that are independent of \( \hat{u} \) from the integral simplifies the Equation 3.43 as follows:

First term:

\[
\frac{d}{dx} \int_0^1 \hat{u} [(1 - \hat{u}) N_j] \left[ \frac{1}{(1 - \hat{u})} \sum_{i=1}^{M} N_i b_{1i} \right] d\hat{u} = \sum_{i=1}^{M} \int_0^1 \hat{u} N_j N_i d\hat{u} \frac{db_{1i}}{dx}
\]
Second term:
\[
\frac{1}{u_e} \frac{du_e}{dx} \int_0^1 (1 - \hat{u}^2) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ \frac{1}{(1 - \hat{u})} \sum_{i=1}^M N_i b_{1i} \right] d\hat{u}
\]
\[
= \frac{1}{u_e} \frac{du_e}{dx} \sum_{i=1}^M \int_0^1 (1 + \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_i d\hat{u} b_{1i}
\]

Third term:
\[
C_0 \frac{u_e p_e}{u_\infty p_\infty} \int_0^1 \left[ (1 - \hat{u}) \sum_{i=1}^M \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_i}{d\hat{u}} b_{6i} - \sum_{i=1}^M N_i b_{6i} \right] d\hat{u}
\]
\[
= C_0 \frac{u_e p_e}{u_\infty p_\infty} \sum_{i=1}^M \int_0^1 \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_i}{d\hat{u}} - N_i \right] d\hat{u} b_{6i}
\]

Therefore, after introducing
\[
K_{1ji} = \int_0^1 \hat{u} N_j N_i d\hat{u},
\]
\[
K_{2ji} = \int_0^1 \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_i}{d\hat{u}} - N_i \right] d\hat{u},
\]
\[
K_{3ji} = \int_0^1 (1 + \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_i d\hat{u},
\]
\[
C_1 = \frac{1}{u_e} \frac{du_e}{dx},
\]
and
\[
C_2 = C_0 \frac{u_e p_e}{u_\infty p_\infty}
\]

into Equation 3.43, the following system of first-order ordinary differential equations is produced:
\[
\sum_{i=1}^M K_{1ji} \frac{db_{1i}}{dx} = C_1 \sum_{i=1}^M K_{3ji} b_{1i} + C_2 \sum_{i=1}^M K_{2ji} b_{6i}
\]

After substituting the approximate solutions into Equation 3.33, removing all terms that are independent of \( \hat{u} \) from the integral, and introducing
\[
K_{4ji} = \int_0^1 \hat{u} (1 - \hat{u}) N_j N_i d\hat{u},
\]
\[
K_{5ji} = \int_0^1 (1 - \hat{u}^2) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_i d\hat{u},
\]
\[ K_{6ji} = \int_0^1 (1 - \dot{\mu})^2 N_j N_i d\dot{\mu}, \quad (3.52) \]

\[ K_{7ji} = \int_0^1 \dot{\mu}(1 - \dot{\mu}) N_j \left[ (1 - \dot{\mu}) \frac{dN_i}{d\dot{\mu}} - N_i \right] d\dot{\mu}, \quad (3.53) \]

\[ K_{8jik} = \int_0^1 (1 - \dot{\mu}) \left[ (1 - \dot{\mu}) \frac{dN_i}{d\dot{\mu}} - N_i \right] \left[ (1 - \dot{\mu}) \frac{dN_j}{d\dot{\mu}} - N_j \right] N_k d\dot{\mu}, \quad (3.54) \]

\[ K_{9jik} = \int_0^1 (1 - \dot{\mu}) \left[ (1 - \dot{\mu}) \frac{dN_i}{d\dot{\mu}} - N_i \right] N_k \left[ (1 - \dot{\mu}) \frac{dN_j}{d\dot{\mu}} - N_j \right] d\dot{\mu}, \quad (3.55) \]

\[ K_{10jik} = (1 - \dot{\mu})^2 N_j N_i \left[ (1 - \dot{\mu}) \frac{dN_k}{d\dot{\mu}} - N_k \right], \quad (3.56) \]

\[ C_3 = C_2 \frac{1}{Pr}, \quad (3.57) \]

and

\[ C_4 = 2C_2 \frac{(\gamma - 1)}{(\gamma + 1)} \left( \frac{1}{Pr} - 1 \right), \quad (3.58) \]

the following system of first-order ordinary differential equations is produced:

\[
\sum_{i=1}^{M} K_{4ji} \frac{db_{4i}}{dx} = -2C_1 \sum_{i=1}^{M} K_{4ji} b_{4i} + C_1 \sum_{i=1}^{M} K_{5ji} b_{4i} + C_2 \sum_{i=1}^{M} \sum_{k=1}^{M} K_{8jik} b_{6i} b_{3k} + C_3 \sum_{i=1}^{M} \sum_{k=1}^{M} K_{10jik} b_{6i} b_{3k} \bigg|_0^1 - C_3 \sum_{i=1}^{M} \sum_{k=1}^{M} K_{9jik} b_{6i} b_{3k} + C_4 \sum_{i=1}^{M} K_{6ji} b_{6i} + C_4 \sum_{i=1}^{M} K_{7ji} b_{6i} \quad (3.59)
\]

The Galerkin integrals \((K_1, \ldots, K_{10})\) are further defined in Appendix A.

The second-order Crank-Nicholson scheme, obtained by using \(\theta = 0.5\), produces an efficient implicit algorithm for marching the solution in the \(x\) direction. Equation 3.49 is approximated by

\[
\sum_{i=1}^{M} K_{1ji} \Delta b_i^{n+1} = \Delta x \left[ \theta F_{1j}^{n+1} + (1 - \theta) F_{1j}^{n} \right], \quad (3.60)
\]

where

\[
F_{1j} = C_1 \sum_{i=1}^{M} K_{3ji} b_{1i} + C_2 \sum_{i=1}^{M} K_{2ji} b_{2i}, \quad (3.61)
\]
\[ \Delta b_{l_i}^{n+1} = b_{l_i}^{n+1} - b_{l_i}^n. \] (3.62)

The superscript \( n \) denotes a particular streamwise, or \( x \), position.

In order to develop a non-iterative marching routine, it is necessary to linearize \( F_{l_j}^{n+1} \) by an expansion about the \( n \)th level following the approach of Briley and MacDonald [19]:

\[
F_{l_j}^{n+1} = F_{l_j}^n + \Delta x \left( \frac{\partial F_{l_j}}{\partial b_{l_i}} \right)^n \frac{\partial b_{l_i}}{\partial x} + \ldots \\
\approx F_{l_j}^n + \left( \frac{\partial F_{l_j}}{\partial b_{l_i}} \right)^n \Delta b_{l_i}^{n+1} \tag{3.63}
\]

Substitution of the linearization into Equation 3.60 produces the following system of equations in \( b_l \):

\[
\sum_{i=1}^{M} K_{Tji} \Delta b_{l_i}^{n+1} = F_{Tj} \tag{3.64}
\]

The variables \( K_{T} \) and \( F_{T} \) are defined as

\[
\sum_{i=1}^{M} K_{Tji} = \sum_{i=1}^{M} K_{1ji} - \theta \Delta x \left[ C_1^{n+1} \sum_{i=1}^{M} K_{3ji} - C_2^{n+1} \sum_{i=1}^{M} K_{2ji} \left( \frac{b_{5i}^n}{b_{1i}^n} - \frac{1}{\eta(l_i)} \frac{\partial b_{5i}^n}{\partial b_{1i}^n} \right) \right], \tag{3.65}
\]

and

\[
F_{Tj} = \Delta x \left\{ \left[ \theta C_1^{n+1} + (1-\theta) C_3^n \right] \sum_{i=1}^{M} K_{3ji} b_{1i}^n \\
+ \left[ \theta C_2^{n+1} + (1-\theta) C_3^n \right] \sum_{i=1}^{M} K_{2ji} b_{6i}^n \right\}. \tag{3.66}
\]

Similarly, applying the Crank-Nicholson scheme to Equation 3.59 gives

\[
\sum_{i=1}^{M} K_{4ji} \Delta b_{4i}^{n+1} = \Delta x \left[ \theta F_{2j}^{n+1} + (1-\theta) F_{2j}^n \right], \tag{3.67}
\]

where

\[
F_{2j} = -2C_1 \sum_{i=1}^{M} K_{4ji} b_{4i} + C_1 \sum_{i=1}^{M} K_{5ji} b_{4i} \\
+ C_2 \sum_{i=1}^{M} \sum_{k=1}^{M} K_{8ijk} b_{5i} b_{3k} + C_3 \sum_{i=1}^{M} \sum_{k=1}^{M} K_{10ijk} b_{5i} b_{3k}, \tag{3.68}
\]
\[- C_3 \sum_{i=1}^{M} \sum_{k=1}^{M} K_{9jk} b6_i b3_k + C_4 \sum_{i=1}^{M} K6_{ji} b6_i + C_4 \sum_{i=1}^{M} K7_{ji} b6_i, \quad (3.68)\]

and

\[
\Delta b4_i^{n+1} = b4_i^{n+1} - b4_i^n. \quad (3.69)
\]

Performing the same linearization technique for \( F2_j^{n+1} \) as was done for \( F1_j^{n+1} \) produces the following system of equations in \( b4 \):

\[
\sum_{i=1}^{M} KS_{ji} \Delta b4_i^{n+1} = FS_j \quad (3.70)
\]

The variables \( KS \) and \( FS \) are defined as

\[
\sum_{i=1}^{M} KS_{ji} = \sum_{i=1}^{M} K4_{ji} - \theta \Delta x \left[ -2C_1^{n+1} \sum_{i=1}^{M} K4_{ji} + C_1^{n+1} \sum_{i=1}^{M} K5_{ji} \right.
\]

\[
+ C_2^{n+1} \sum_{i=1}^{M} \sum_{k=1}^{M} K8_{jik} b6_i b2_k^n + C_3^{n+1} \sum_{i=1}^{M} \sum_{k=1}^{M} K10_{jik} b6_i b2_k^n \bigg|_0
\]

\[- C_3^{n+1} \sum_{i=1}^{M} \sum_{k=1}^{M} K9_{jik} b6_i b2_k^n \right], \quad (3.71)
\]

and

\[
FS_j = \Delta x \left\{ -2 \left[ \theta C_1^{n+1} + (1 - \theta)C_1^n \right] \sum_{i=1}^{M} K4_{ji} b4_i \right\}
\]

\[
+ \left[ \theta C_1^{n+1} + (1 - \theta)C_1^n \right] \sum_{i=1}^{M} K5_{ji} b4_i
\]

\[
+ \left[ \theta C_2^{n+1} + (1 - \theta)C_2^n \right] \sum_{i=1}^{M} \sum_{k=1}^{M} K8_{jik} b6_i b3_k \bigg|_0
\]

\[
+ \left[ \theta C_3^{n+1} + (1 - \theta)C_3^n \right] \sum_{i=1}^{M} \sum_{k=1}^{M} K10_{jik} b6_i b3_k \bigg|_0
\]

\[- \left[ \theta C_3^{n+1} + (1 - \theta)C_3^n \right] \sum_{i=1}^{M} \sum_{k=1}^{M} K9_{jik} b6_i b3_k
\]

\[
+ \left[ \theta C_4^{n+1} + (1 - \theta)C_4^n \right] \sum_{i=1}^{M} K6_{ji} b6_i
\]

\[
+ \left[ C_4^{n+1} + (1 - \theta)C_4^n \right] \sum_{i=1}^{M} K7_{ji} b6_i \right\}. \quad (3.72)
\]
Boundary conditions for $\tau$ and $s$ at the boundary-layer edge are automatically imposed through the use of the $(1 - \hat{u})$ term in the trial solutions. As note previously, no wall boundary condition for $\tau$, and hence $\Delta b_{1_i}^{n+1}$, is imposed. The boundary condition at the wall for $s$, and hence $\Delta b_{4_i}^{n+1}$, is given by

$$\Delta b_{4_i}^{n+1} = \Delta b_{1_i}^{n+1} s_w.$$  \hfill (3.73)

The system of equations given by Equations 3.64 and 3.70 can be solved without iteration by the Thomas algorithm at each streamwise location using a variable step size. The variable step size is based on a comparison of the linearized and nonlinearized values of $F_{1_j}^{n+1}$ and $F_{2_j}^{n+1}$. The maximum relative error expression is given by

$$\lambda = \max \left| \frac{F_{1_j}^{n+1}({\text{linearized}}) - F_{1_j}^{n+1}({\text{nonlinearized}})}{F_{1_j}^{n+1}({\text{nonlinearized}})} \right|, \quad \left| \frac{F_{2_j}^{n+1}({\text{linearized}}) - F_{2_j}^{n+1}({\text{nonlinearized}})}{F_{2_j}^{n+1}({\text{nonlinearized}})} \right|. \hfill (3.74)$$

Alternately, the linearization error can be efficiently controlled by enforcing a maximum allowable ratio of $\Delta b_{1_i}^{n+1}$ and $\Delta b_{4_i}^{n+1}$. The maximum ratio expression is given by

$$\lambda = \max \left| \frac{\Delta b_{1_i}^{n+1}}{b_{1_i}^n}, \frac{\Delta b_{4_i}^{n+1}}{b_{4_i}^n} \right|. \hfill (3.75)$$

Given a maximum and minimum tolerance $\lambda_{\text{min}}$ and $\lambda_{\text{max}}$, respectively, the step size is modified according to the following procedure:

$$\lambda_{\text{min}} \leq \lambda \leq \lambda_{\text{max}} \quad \text{step size is unchanged}$$

$$\lambda > \lambda_{\text{max}} \quad \text{step size is halved}$$

$$\lambda < \lambda_{\text{min}} \quad \text{step size is doubled}$$

### 3.4 Self-Similar Solutions

The self-similar solution may be used near the leading edge to obtain the needed initial conditions. In order to solve Equation 3.64 for the shear stress at the $(n + 1)^{th}$ streamwise location, it is necessary to know the shear stress at the $n^{th}$ streamwise location. A number of techniques have been used to obtain self-similar solutions for
specific geometries within certain regions. The classic Falkner-Skan and Illingworth-Stewartson series can produce self-similar solutions for stagnation point and wedge flows with heat transfer [20], [21]. This same method may be applied to the traditional Dorodnitsyn equations of motion (laminar form of Equations 3.17, 3.20, and 3.22) to provide the initial \( \tau \) and \( s \) values. Even though the governing equations of interest are turbulent, the laminar self-similar solutions are valid in the region near the stagnation point where transition to turbulence has not yet occurred.

At a small distance from the stagnation point, the velocity \( u_\varepsilon \) can be approximated by

\[
    u_\varepsilon = u_\infty \xi^m, \tag{3.76}
\]

where \( m \) is related to the pressure gradient coefficient \( \beta \) as follows:

\[
    \beta = \frac{2m}{m + 1} \tag{3.77}
\]

Also, near the stagnation point \( \Delta \xi = \xi - 0 \) and \( \Delta x = x - 0 \) such that Equation 3.13 can be written as

\[
    \xi = \frac{u_\varepsilon p_\varepsilon}{u_\infty p_\infty} x. \tag{3.78}
\]

The similarity parameter \( \alpha \), used to reduce the number of independent variables, is given by

\[
    \alpha = \eta \left( \frac{m + 1}{2C_0 \xi} \right)^{\frac{1}{2}}. \tag{3.79}
\]

3.4.1 Momentum Similarity

The development of the momentum similarity equation takes the same form as the classical development. Using the no-slip boundary condition and integrating with respect to \( \eta \), the continuity equation becomes

\[
    \dot{\psi} = -\frac{\partial}{\partial \xi} \int_0^\eta \dot{u} d\eta. \tag{3.80}
\]

Representing the nondimensional velocity \( \dot{u} \) as

\[
    \dot{u} = \frac{d\Psi}{d\alpha} = \Psi', \tag{3.81}
\]

and substituting into Equation 3.80 gives

\[
    \dot{\psi} = \left( \frac{\partial \alpha}{\partial \eta} \right)^2 \Psi - \frac{\partial \alpha}{\partial \eta} \Psi'. \tag{3.82}
\]
Substituting Equations 3.81 and 3.82 into the laminar form of Equation 3.20 generates a momentum similarity equation of the following form:

\[
\Psi'''' + \frac{\frac{1}{u_c} \frac{du_c}{dx}}{C_0 \left( \frac{\partial \alpha}{\partial \eta} \right)^3} \left( 1 - \Psi^2 \right) - \frac{\frac{\partial^2 \alpha}{\partial x \partial \eta}}{C_0 \left( \frac{\partial \alpha}{\partial \eta} \right)^3} \Psi \Psi'' = 0
\]  
(3.83)

The coefficients are evaluated by obtaining the same form of similarity equation as White [20], which requires

\[
\beta = \frac{\frac{1}{u_c} \frac{du_c}{dx}}{C_0 \left( \frac{\partial \alpha}{\partial \eta} \right)^2},
\]  
(3.84)

and

\[
m = \frac{C_0 \left( \frac{\partial \alpha}{\partial \eta} \right)^3}{\frac{\partial^2 \alpha}{\partial x \partial \eta}} - 1.
\]  
(3.85)

Therefore, the final momentum similarity equation can be written as

\[
\Psi'''' + \beta \left( 1 - \Psi^2 \right) + \left( \frac{1}{m + 1} \right) \Psi \Psi'' = 0,
\]  
(3.86)

with the following boundary conditions:

\[
\Psi(0) = \Psi'(0) = 0 \text{ and } \Psi'(\infty) = 1
\]  
(3.87)

### 3.4.2 Energy Similarity

Representing the nondimensional enthalpy parameter \( s \) as

\[
s = \Omega(\alpha),
\]  
(3.88)

and substituting into the laminar form of Equation 3.22 along with Equations 3.81 and 3.82 generates an energy similarity equation of the following form:

\[
\Omega'' + \frac{Pr}{m + 1} \Psi \Omega' - 2Pr \beta \Psi' \Omega + 2 \frac{\gamma - 1}{\gamma + 1} (1 - Pr) \left[ \Psi''^2 + \Psi' \Psi''' \right] = 0
\]  
(3.89)

The energy similarity coefficients are evaluated according to the previous definitions for \( \beta \) and \( m \). Heat-transfer boundary conditions are given by

\[
\Omega(0) = s_w = \left( 1 - \frac{T_w}{T_a} \right) \frac{1}{u_c^2} \text{ and } \Omega(\infty) = 0,
\]  
(3.90)
where $T_w$ and $T_o$ denote wall and stagnation temperatures, respectively. If an adiabatic wall condition is prescribed, the boundary conditions are given as

$$\Omega'(0) = 0 \quad \text{and} \quad \Omega(\infty) = 0.$$  \hfill (3.91)

Equations 3.86 and 3.89 define a two-point boundary-value problem (BVP). Using bisection for the shooting method along with a 4th-order Runge-Kutta scheme, the BVP can be solved to produce an initial set of $\tau$ and $s$ values that can be used to begin the marching routine. The value of $\tau$ is proportional to $\Psi''$ and is calculated by the following equation:

$$\tau = \frac{\partial \hat{u}}{\partial \eta} = \frac{\partial \alpha}{\partial \eta} \frac{\partial \hat{u}}{\partial \alpha} = \left( \frac{(m + 1) u_{\infty}^2 p_{\infty}}{2 C_0} \frac{u_{e}^2}{u_{e}^2} \right)^{\frac{1}{2}} \Psi''$$  \hfill (3.92)

### 3.5 Turbulence Models

A turbulent boundary layer can be regarded approximately as a composite layer made up of inner and outer regions which arise due to the different response of the fluid to shear and pressure gradients near the wall [9]. In order to determine the eddy-viscosity $\mu_t$ in the inner and outer regions, two models based on the mixing-length concept are used in the SDG formulation.

Prandtl proposed that each turbulent fluctuation could be related to a length scale and velocity gradient:

$$\bar{\mu}_t \approx \bar{p}_t^2 \left| \frac{\partial \hat{u}}{\partial \bar{y}} \right|$$  \hfill (3.93)

The mixing length $\bar{l}$ is related to the flow conditions and takes on different values in the inner and outer turbulent flow regions.

A single inner region eddy-viscosity model, which follows the form used by Cebeci-Smith, was considered in this analysis [9]. The van Driest modified mixing-length formula in terms of the Dobrodynsytzyn transformation is given by

$$\frac{\mu_t}{\mu} = \frac{Re^{3/2} p_{\infty} u_{e}^2}{C_0 \rho e^2} \rho^2 l^2 \tau,$$  \hfill (3.94)

where $l$ is the mixing length ($\kappa = 0.41$)

$$l = \kappa y \left( 1 - e^{y/A^+} \right).$$  \hfill (3.95)
The parameter $A^+$ is dependent on the pressure gradient according to

$$A^+ = \frac{A_o}{(1 + 10p^+)^{\frac{1}{2}}}, \quad (3.96)$$

where $A_o = 26$,

$$p^+ = \frac{C_0}{\sqrt{Re}} \frac{p_e}{\rho w^3 u_e^2 \frac{d}{dx}}, \quad \text{and} \quad u_r = u_e \left( \frac{C_0}{\sqrt{Re}} \frac{p_e \tau_w}{\rho w} \right)^{\frac{1}{2}}. \quad (3.97)$$

The mixing length parameter $l$ is related to the coordinates $y^+$ and $y$ which are obtained from the solution by

$$y^+ = \frac{Re p_\infty}{C_0 p_e} \frac{u_r^2}{u_\infty} y, \quad \text{where} \quad y = \frac{1}{\sqrt{Re}} \frac{u_\infty}{u_e} \int_0^{\hat{u}} \frac{1}{\rho \tau} \frac{d\hat{u}}{d}. \quad (3.98)$$

Two outer region eddy-viscosity formulations were considered in this analysis. The first outer region eddy-viscosity model considered is based on the form used by Cebeci and Smith [9]. The outer model is modified to account for intermittency near the edge of the boundary layer $\delta$:

$$\frac{\mu_t}{\mu} = \frac{0.0168}{[1 + 5.5(y/\delta)^6]} \frac{Re p_\infty}{C_0 p_e} \frac{u_e}{u_\infty} \rho^2 \delta^* \quad (3.99)$$

The second outer region eddy-viscosity model considered is based on the form developed by Baldwin and Lomax [22]. Instead of using the boundary layer and displacement thickness as parameters in the outer formulation, the Baldwin-Lomax model uses certain maximum functions occurring in the boundary layer. The eddy-viscosity relation is given by

$$\frac{\mu_t}{\mu} = \frac{0.0168 C_{cp}}{[1 + 5.5(C_{kleb}/y_{max})^6]} \frac{Re p_\infty}{C_0 p_e} \frac{F_{max}}{u_\infty} \rho^2 y_{max}, \quad (3.100)$$

where $y_{max}$ is the value of $y$ corresponding to $F_{max}$. The parameters $C_{cp}$ and $C_{kleb}$ are calculated by the following expressions in order to fit the known properties of Coles' wake law and equilibrium pressure gradients [23]:

$$C_{cp} = \frac{3 - 4C_{kleb}}{2C_{kleb} (2 - 3C_{kleb} + C_{kleb}^3)} \quad (3.101)$$

$$C_{kleb} = \frac{2}{3} - \frac{0.1312}{0.1724 + \beta_r} \quad \beta_r = \frac{y_{max} \frac{du_e}{dx}}{u_r} \quad (3.102)$$

The choice between the inner and outer eddy-viscosity formula is made by taking the smaller value; the changeover typically occurs at $\hat{u} \approx 0.7$. 
Chapter 4

Viscous-Inviscid Interaction

In many flows of practical aerodynamic interest, the effects of viscosity and turbulence are confined to a relatively thin shear layer near the airfoil surface and wake. The flow over an airfoil can therefore be divided into two regions: an inner region, where viscous effects are important, and an outer inviscid region. Modelling the viscous and inviscid flow regions separately while providing a mechanism by which each solution influences the other is called Viscous-Inviscid Interaction (VII). A VII method organizes the viscous and inviscid parts of the overall solution to interact in an iterative way so that convergence of the final solution is achieved as economically and accurately as possible. The principle interaction between the regions arises from the displacement effect of the shear layers which leads to a thickening of the equivalent body with a corresponding change in the surface pressure [24]. The resulting interaction is labeled as either weak or strong, depending on the change in pressure and the extent to which higher-order viscous effects influence the overall solution. VII, based on a direct relationship between the viscous and inviscid regions of flow, is applicable as long as the disturbances to the inviscid flow due to the viscous displacement effect are small [25].

4.1 Viscous-Inviscid Coupling

The classical interaction approach is to obtain an approximation to the inviscid flow, extract velocity and pressure from the inviscid solution, use the external conditions to obtain an approximation to the viscous flow, extract displacement thickness from the viscous solution, and use the viscous parameters to modify the original body geometry and obtain another estimate of the inviscid flow. An alternative to adding the displacement thickness distribution to the original body thickness is to impose a transpiration velocity boundary condition at the body surface. These iterative cycles, depicted in Figure 4.1, continue until the inviscid and viscous solutions are converged and compatible.
4.1.1 Displacement Thickness

The displacement thickness is the height by which a streamline is displaced upward by the presence of the boundary layer. Considering the flow over a flat surface as depicted in Figure 4.2, the no-slip condition at the wall causes a partial obstruction of the freestream flow. This results in an upward deflection of the streamline passing through point $\tilde{h}$ at station 1 by a distance $\delta^*$ at station 2. Because the flat surface and the streamline form the boundaries of a streamtube, the mass flow across stations 1 and 2 must be equal and is expressed by

$$
\int_0^h \bar{p}_e \bar{u}_e d\bar{y} = \int_0^h \bar{p} \bar{u} d\bar{y} + \bar{p}_e \bar{u}_e \bar{\delta}^*.
$$

(4.1)

Rearranging gives a familiar form of the compressible displacement thickness,

$$
\bar{\delta}^* = \int_0^h \left(1 - \frac{\bar{p} \bar{u}}{\bar{p}_e \bar{u}_e}\right) d\bar{y}.
$$

(4.2)

The nondimensionalized compressible displacement thickness in terms of the Dorodnitsyn formulation is given by

$$
\delta^* = \frac{(\gamma + 1)}{2\gamma} \frac{1}{u_e p_e \sqrt{Re}} \int_0^1 (1 - \bar{u}) \frac{1}{\tau} d\bar{u} + \frac{(\gamma - 1)}{2\gamma} \frac{u_e}{p_e \sqrt{Re}} \int_0^1 \bar{u}(1 - \bar{u}) \frac{1}{\tau} d\bar{u}
$$

$$
- \frac{(\gamma + 1)}{2\gamma} \frac{u_e}{p_e \sqrt{Re}} \int_0^1 \frac{s}{\tau} d\bar{u}.
$$

(4.3)
The advantage of using the displacement thickness as the coupling mechanism between the viscous and inviscid solutions is its simplicity. The disadvantage of the displacement thickness approach is that the inviscid grid must be generated after each viscous iteration because the body about which the inviscid flow is computed changes when the displacement thickness alters the effective body. An additional disadvantage of the displacement thickness approach is the possibility of supercritical VII which does not allow smooth transition to separation [26]. It has been shown that the classical displacement thickness interaction becomes supercritical when $M_{\infty} \approx 1.5 \rightarrow 2.0$, depending on the history of the turbulent boundary layer [27].

### 4.1.2 Transpiration Velocity

The transpiration velocity is an inviscid normal-velocity boundary condition which is imposed at the body surface to simulate the displacement of the inviscid flow by the viscous flow momentum defect [24]. An expression for the transpiration velocity can be obtained by integrating the difference between the inviscid and viscous continuity equations across the boundary layer while applying the Prandtl boundary-layer and uniform inviscid-flow assumptions [26]. The transpiration velocity expression is given by

$$
\bar{u}_t = \frac{1}{\bar{\rho}_e} \frac{\partial}{\partial x} \left( \bar{\rho}_e \bar{u}_e \delta^* \right).
$$

(4.4)
The nondimensionalized compressible transpiration velocity in terms of the Dorodnitsyn formulation is given by

\[ v_t = u_e \frac{d\delta^*}{dx} + \left( \frac{du_e}{dx} + \frac{u_e}{\rho_e} \frac{d\rho_e}{dx} \right) \delta^*. \]  

(4.5)

The transpiration velocity distribution should result in an inviscid streamline coincident with the effective body obtained with the displacement thickness approach. The advantages of the transpiration velocity approach are that the inviscid grid need not be regenerated after each viscous iteration and that the interaction always allows smooth transition to separation [27].

### 4.2 Euler Equation Solver

A fast and robust two-dimensional Euler code (GAUSS2), developed by Dr. Peter M. Hartwich of Vigyan Inc., is used as the inviscid flow solver in the VII scheme [28], [29]. The method uses a floating shock-fitting technique that has been combined with a second-order accurate upwind scheme based on the split-coefficient-matrix (SCM) method and with a time-implicit, diagonalized approximate-factorization (AF) algorithm. The result is a fast and robust two-dimensional Euler code that produces accurate solutions for shocked flows on crude meshes which are not adapted to the shock fronts.

The equations for two-dimensional, compressible, and nonconservative Euler equations for a polytropic gas at constant \( \gamma \) are given in general coordinates as

\[ \mathbf{Q}_t + \mathbf{A} \mathbf{Q}_x + \mathbf{B} \mathbf{Q}_{xx} = 0, \]  

(4.6)

where

\[ \mathbf{Q} = (a', u', v', s')^T, \]  

(4.7)

and the vectors \( \mathbf{A} \) and \( \mathbf{B} \) are defined by a coefficient matrix \( \mathbf{C} \) [28]. The primes denote inviscid dependent variables where \( a', u', v', \) and \( s' \) are speed of sound, cartesian velocities, and entropy, respectively. The dependent variables have been nondimensionalized as follows:

\[
\begin{align*}
x' &= \bar{x} / L \\
y' &= \bar{y} / L \\
u' &= \frac{\bar{u}}{\sqrt{\bar{p}_\infty / \bar{\rho}_\infty}} \\
v' &= \frac{\bar{v}}{\sqrt{\bar{p}_\infty / \bar{\rho}_\infty}} \\
a' &= \frac{\bar{a}}{\sqrt{\bar{p}_\infty / \bar{\rho}_\infty}} \\
p' &= \frac{\bar{p}}{\bar{\rho}_\infty} \\
p' &= \frac{\bar{\rho}}{\bar{\rho}_\infty}
\end{align*}
\]  

(4.8)
4.2.1 Viscous-Inviscid Interfaces

In order to allow the boundary-layer and Euler methods to work together in an VII scheme, it is necessary to develop a set of expressions which convert the variables from one type of nondimensionalization to another. The boundary-layer method employs a body-normal coordinate system while the Euler method employs a cartesian coordinate system, so a coordinate system conversion must be performed. Also, the methods use different reference values to nondimensionalize velocity and pressure, so a dependent variable conversion must be performed. The variable conversions which account for both the coordinate system and nondimensionalization differences are summarized in Table 4.1, where $\theta$ is the local body angle.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Euler $\rightarrow$ Boundary Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>$x = \sqrt{x'^2 + y'^2}$</td>
</tr>
<tr>
<td>Velocity</td>
<td>$u = (u'\cos\theta + v'\sin\theta)\sqrt{p_\infty}$</td>
</tr>
<tr>
<td>Pressure</td>
<td>$p = p'p_\infty$</td>
</tr>
</tbody>
</table>
Chapter 5

Numerical Results

5.1 Convergence Properties

The convergence properties of the Dorodnitsyn finite element approximation of \( \tau \) are plotted in Figure 5.1. The average discrete root-mean-square of the relative error is calculated by the following equation:

\[
\tau_{rms} = \left[ \sum_{i=1}^{M-1} \left( \frac{\tau_i}{\tau_{i,exact}} - 1 \right)^2 \right]^{1/2} / (M - 1)
\]  

(5.1)

The exact value of \( \tau \) is obtained from a solution of the similarity equations developed in Section 3.4.1 for flow over a flat plate with zero pressure gradient (\( \beta = 0 \)). The 4th-order Runge-Kutta step size was set to \( \Delta \alpha = 0.01 \). The results presented in Figure 5.1 are for 8 values of \( \Delta u \) corresponding to a discretization of 3, 4, 5, 7, 9, 11, 15, and 19 nodes at \( x = 0.5 \) along the plate. In order to evaluate the convergence properties of the spatial discretization across the boundary layer, the step size was held at a small constant value to minimize the influence of the marching routine. Examination of the figure indicates that the use of linear elements achieves the theoretically expected second-order convergence. It can be seen that the convergence rate of the scheme decreases slightly as the number of nodes increases. The decreasing convergence rate can be attributed to the decrease in accuracy of the linearized marching scheme with increasing number of nodes. An extensive analysis of convergence properties for favorable and adverse pressure-gradient cases as well as quadratic elements may be found in Reference [18]. Fletcher and Fleet showed that the accuracy of linear elements on coarse grids are comparable to quadratic elements, and the convergence properties for the adverse and favorable pressure-gradient cases are comparable to the zero-gradient case.
Figure 5.1 Nondimensional Shear Stress Convergence
Results for a Flat Plate at $M_\infty = 0.800$, $Re = 5000$

5.2 Flat Plate

An assessment of the accuracy of the semi-discrete Galerkin (SDG) method may be performed by examining the ability of the method to reproduce the well-known compressible laminar boundary-layer solution over a thermally-insulated flat plate with zero pressure gradient. The computations were performed at a Mach number of 0.8 and a Reynolds number based on plate length of 5000. A discretization of 9 nodes was used across the boundary layer. The marching routine step size was efficiently controlled by setting $\lambda_{\text{min}} = 1.0 \times 10^{-4} = 0.1\lambda_{\text{max}}$ to maintain the ratio of $\frac{\Delta h_{1}^{n+1}}{h_{1}^{n}}$ and $\frac{\Delta h_{4}^{n+1}}{h_{4}^{n}}$. Again, an exact analytical solution for this flow may be obtained by solution of the Falkner-Skan similarity equation [20]. Figure 5.2 shows excellent agreement between the exact and computed nondimensional shear-stress values at $x = 0.5$. The similarity property of the solution was also verified by examining the profiles at different stations along the length of the plate. Excellent agreement was observed except for stations close to the leading edge where effects of the stagnation-point flow are present. The skin friction along the plate was calculated using the
Figure 5.2 Comparison of Computed and Exact Nondimensional Shear Stress Profiles for a Flat Plate at \( x = 0.5, M_\infty = 0.800, Re = 5000 \)

relation

\[
C_f = \frac{2C_0}{\sqrt{Re}} \frac{u_T^2}{p_\infty} \frac{p_e}{p_\infty} \tau_w. 
\]  \hspace{1cm} (5.2)

To illustrate the high accuracy of the Dorodnitsyn formulation on extremely rough grids, both 3 and 9-node coefficient of friction solutions are plotted along with the exact solution in Figure 5.3. There is excellent agreement between the 9-node and exact solutions. The 3-node solution produced good results except in the region very close to the leading edge.

5.3 NACA 0012 Airfoil

The NACA 0012 was chosen as the primary test airfoil for the compressible turbulent VII validation. The NACA 0012 is a symmetric airfoil which has been tested both computationally and experimentally by a great number of researchers. Of particular importance to the VII scheme is that the experimental data at zero angle of attack is not affected by wall interference due to lift. The airfoil is tested for attached turbulent flow at high Reynolds numbers and transonic speeds as shown in Table 5.1. In order
to minimize the influence of wind tunnel wall-interference effects in the comparison of experimental and computed results, the numerical angle of attack was varied in each case to match the computed lift and the experimental normal force coefficients. In each of the NACA 0012 cases, the 161x31 C-grid shown in Figure 5.4 was used by GAUSS2 to solve the inviscid equations of motion. The outer boundary of the C-grid is located 5 to 6 chord lengths from the airfoil in all directions. The transpiration velocity boundary condition was used as the VII coupling mechanism and was relaxed after each global VII iteration according to

\[ v_{t,rel}^q = v_t^q + (1 - \omega)v_t^{q-1}, \]  

(5.3)

where \( q \) denotes a particular global VII iteration and \( \omega \) is a relaxation parameter. The relaxation parameter was set equal to 0.1 for all NACA 0012 calculations. Convergence of the scheme was assumed when lift and total drag coefficients changed less than 0.1% between global iterations. The maximum number of global VII iterations was set at 50, and the ratio of local inviscid to viscous iterations was set at 100.
Figure 5.4 NACA 0012 C-Grid used by GAUSS2
Table 5.1  NACA 0012 Test Cases

<table>
<thead>
<tr>
<th>Case</th>
<th>$M_\infty$</th>
<th>$Re$</th>
<th>$\alpha_{exp}$</th>
<th>$\alpha_{num}$</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.499</td>
<td>$9.0 \times 10^6$</td>
<td>-0.14</td>
<td>-0.14</td>
<td>[30]</td>
</tr>
<tr>
<td>B</td>
<td>0.700</td>
<td>$9.0 \times 10^6$</td>
<td>1.86</td>
<td>1.37</td>
<td>[30]</td>
</tr>
</tbody>
</table>

A boundary-layer discretization of 9 nodes was used and the marching routine was controlled in the same manner as the flat-plate calculations by setting $\lambda = 5.0 \times 10^{-4}$.

5.3.1  NACA 0012 - Case A

This test case consists of a NACA 0012 airfoil at a numerical angle of attack of $-0.14$, a freestream Mach number of 0.499, and a Reynolds number of $9.0 \times 10^6$. The flow is attached and subsonic over the entire airfoil surface. Transition to turbulence was numerically tripped at the leading edge. The computed coefficient of pressure compares reasonably well with experiment but is slightly underpredicted over the aft section to the trailing edge of the airfoil as shown in Figure 5.5. No experimental data was readily available to compare the computed friction coefficient and displacement thickness, but both $C_f$ and $\delta^*$ are plotted in Figures 5.6 and 5.7 for future reference.

5.3.2  NACA 0012 - Case B

This test case consists of a NACA 0012 airfoil at a numerical angle of attack of 1.37, a freestream Mach number of 0.700, and a Reynolds number of $9.0 \times 10^6$. For this case, the flow is attached and just slightly supersonic near the leading-edge upper surface. A comparison of experimental and calculated coefficient of pressure is shown in Figure 5.8. The predicted $C_p$ compares well with experiment but slightly overpredicts the peak value at the supersonic region near the leading edge and deviates at the trailing edge. Again, the computed skin friction and displacement thickness values are plotted in Figures 5.9 and 5.10 for future reference.
Figure 5.5  Comparison of Pressure Coefficient Distribution for the NACA 0012 Airfoil at $M_\infty = 0.499$, $\alpha_{num} = -0.14$, $Re = 9.0 \times 10^6$

Figure 5.6  Skin Friction Coefficient Distribution for the NACA 0012 Airfoil Upper-Surface at $M_\infty = 0.499$, $\alpha_{num} = -0.14$, $Re = 9.0 \times 10^6$
Figure 5.7  Upper-Surface Displacement Thickness for the NACA 0012 Airfoil at $M_{\infty} = 0.499$, $\alpha_{num} = -0.14$, $Re = 9.0 \times 10^6$

Figure 5.8  Comparison of Pressure Coefficient Distribution for the NACA 0012 Airfoil at $M_{\infty} = 0.700$, $\alpha_{num} = 1.37$, $Re = 9.0 \times 10^6$
Figure 5.9  Skin Friction Coefficient Distribution for the NACA 0012 Airfoil Upper-Surface at $M_\infty = 0.700$, $\alpha_{num} = 1.37$, $Re = 9.0 \times 10^6$

Figure 5.10  Upper-Surface Displacement Thickness for the NACA 0012 Airfoil at $M_\infty = 0.700$, $\alpha_{num} = 1.37$, $Re = 9.0 \times 10^6$
5.3.3 Aerodynamic Characteristics

Figure 5.11 shows a comparison of lift coefficient vs. angle of attack for the NACA 0012 airfoil at $M_\infty = 0.700$ and $Re = 9.0 \times 10^6$. The SDG VII scheme overpredicts coefficient of lift at low angles of attack, but the prediction improves at higher angles of attack. For angles of attack above 1.54, the flow is more strongly transonic and eventually separates. Drag polar comparisons are displayed in Figure 5.12 for the NACA 0012 airfoil at $M_\infty = 0.700$ and $Re = 9.0 \times 10^6$. The data again shows that the computed lift coefficient is overpredicted at lower angles of attack. It is also noted that the computed drag coefficient is underpredicted at lower angles of attack with an improved prediction at higher incidence. The flowfield is subsonic for $C_L$ values of approximately 0.2 and lower. Drag values below this point correspond to pressure plus skin friction drag. Drag values above this point have, in addition, a wave drag component. Transonic drag-rise characteristics for the NACA 0012 airfoil at zero-lift conditions are displayed in Figure 5.13. The turbulent boundary layer was numerically tripped at the leading edge, and all computations were performed at a Reynolds number of 9 million. The cross-hatch range of experimental values is based on a "best of six" set of data as described by McCroskey [32]. An underprediction of the drag coefficient is observed at lower Mach numbers with a trend toward improved results at higher Mach numbers.

5.4 RAE 2822 Airfoil

The RAE 2822 was chosen as a supplementary test airfoil for the compressible turbulent VII interaction. The RAE 2822 is a supercritical airfoil with a moderate amount of aft camber which poses a challenge in achieving VII convergence. Also, the experimental data in Reference [31] contains a number of boundary-layer and wake parameters such as displacement thickness, momentum thickness, and skin friction which are helpful in validating the boundary-layer solution. The airfoil is tested for attached turbulent flow at high Reynolds numbers and transonic speeds as shown in Table 5.2. A C-grid similar to the one use for the NACA 0012 test cases was used for all of the RAE 2822 calculations. The relaxation parameter was reduced to $\omega = 0.05$ which produced a slower but more stable VII convergence. Consequently, the maximum number of global VII iterations was increased to 75. The discretization and marching-routine controls were set at values equal to those used in the NACA 0012
Figure 5.11  Comparison of Lift Coefficient vs. Angle of Attack for the NACA 0012 Airfoil at \( M_\infty = 0.700, \; Re = 9.0 \times 10^6 \)

Figure 5.12  Comparison of Lift vs. Drag Polars for the NACA 0012 Airfoil at \( M_\infty = 0.700, \; Re = 9.0 \times 10^6 \)
test cases. For all of the RAE 2822 cases, the skin friction coefficient is based on the boundary-layer edge dynamic pressure.

5.4.1 RAE 2822 - Case A

This test case consists of a RAE 2822 airfoil at a numerical angle of attack of -2.18, a freestream Mach number of 0.676, and a Reynolds number of $5.7 \times 10^6$. The flow in this case is subsonic and attached over the entire airfoil surface. The computed coefficient of pressure shown in Figure 5.14 compares well with the experimental values. The $C_p$ is slightly underpredicted on the lower surface near the leading edge and again deviates from experiment at the trailing edge. The coefficient of friction and displacement thickness results given in Figures 5.15 and 5.16 have good agreement with the experimental values. As seen in Figure 5.15, turbulence was tripped at $x = 0.11$. Figure 5.17 shows the velocity profiles in the boundary layer at three $x$-locations. The velocity profile deviation from experimental values is attributed to
Table 5.2  RAE 2822 Test Cases

<table>
<thead>
<tr>
<th>Case</th>
<th>$M_{\infty}$</th>
<th>$Re$</th>
<th>$\alpha_{exp}$</th>
<th>$\alpha_{num}$</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.676</td>
<td>$5.7 \times 10^6$</td>
<td>-2.18</td>
<td>-2.18</td>
<td>[31]</td>
</tr>
<tr>
<td>B</td>
<td>0.676</td>
<td>$5.7 \times 10^6$</td>
<td>2.40</td>
<td>1.90</td>
<td>[31]</td>
</tr>
<tr>
<td>C</td>
<td>0.725</td>
<td>$6.5 \times 10^6$</td>
<td>2.55</td>
<td>2.10</td>
<td>[31]</td>
</tr>
</tbody>
</table>

the fact that the SDG method calculates velocity indirectly since shear stress is a dependent variable.

5.4.2  RAE 2822 - Case B

This test case consists of a RAE 2822 airfoil at a numerical angle of attack of 1.90, a freestream Mach number of 0.676, and a Reynolds number of $5.7 \times 10^6$. The flow is attached over the entire surface and slightly supersonic near the leading edge on the upper surface of the airfoil. Transition to turbulence was numerically tripped at $x = 0.03$. The computed coefficient of pressure is compared with experimental values in Figure 5.18. The $C_p$ prediction on the lower surface of the airfoil is in good agreement with experiment except at the trailing edge, and the upper-surface prediction is slightly below experimental values over the aft section of the airfoil. It should be noted that the pressure coefficient prediction in the supersonic flow region is in good agreement with experiment. Both coefficient of friction and displacement thickness values compare well with the experimental values as shown in Figures 5.19-5.20.

5.4.3  RAE 2822 - Case C

This test case consists of a RAE 2822 airfoil at a numerical angle of attack of 2.10, a freestream Mach number of 0.725, and a Reynolds number of $6.5 \times 10^6$. The transition to turbulence was again numerically tripped at $x = 0.03$. For this case, the flow is attached and supersonic on the upper surface where a moderately strong shock wave is experienced. A comparison of experimental and calculated coefficient of pressure is shown in Figure 5.21. The predicted $C_p$ compares relatively well with experiment.
Figure 5.14 Comparison of Pressure Coefficient Distribution for the RAE 2822 Airfoil at $M_{\infty} = 0.676$, $\alpha_{\text{num}} = -2.18$, $Re = 5.7 \times 10^6$

Figure 5.15 Comparison of Upper-Surface Skin Friction Coefficient for the RAE 2822 Airfoil at $M_{\infty} = 0.676$, $\alpha_{\text{num}} = -2.18$, $Re = 5.7 \times 10^6$
Figure 5.16  Upper-Surface Displacement Thickness Comparison for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = -2.18$, $Re = 5.7 \times 10^6$

Figure 5.17  Comparison of Computed and Measured Velocity Profiles for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = -2.18$, $Re = 5.7 \times 10^6$
Figure 5.18  Comparison of Pressure Coefficient Distribution for the RAE 2822 Airfoil at $M_{\infty} = 0.676$, $\alpha_{\text{num}} = 1.90$, $Re = 5.7 \times 10^6$

Figure 5.19  Comparison of Upper-Surface Skin Friction Coefficient for the RAE 2822 Airfoil at $M_{\infty} = 0.676$, $\alpha_{\text{num}} = 1.90$, $Re = 5.7 \times 10^6$
Figure 5.20 Upper-Surface Displacement Thickness Comparison for the RAE 2822 Airfoil at $M_\infty = 0.676$, $\alpha_{num} = 1.90$, $Re = 5.7 \times 10^6$

except at the shock wave which is predicted upstream of the experimental result. The coefficient of friction and displacement thickness results given in Figures 5.22 and 5.23 have good agreement with the experimental values.

5.5 Grid Refinement Study

A grid refinement study was performed to demonstrate the sensitivity of computed force coefficients to the grid spacing used in the SDG VII scheme. The NACA 0012 airfoil was solved at flow conditions given in Case A for a number of viscous finite element discretizations and inviscid grid sizes. Figure 5.24 is a plot of computed lift coefficient vs. finite element space width for coarse (81 x 17) and fine (161 x 33) inviscid grids. The trend is to decrease the lift coefficient with increasing space width. The effect of grid spacing on the computed drag coefficient is shown in Figure 5.25, where the trend is to increase the drag coefficient with increasing space width. The relatively small slope of each curve indicates the method produces reasonable drag
Figure 5.21 Comparison of Pressure Coefficient Distribution for the RAE 2822 Airfoil at $M_\infty = 0.725$, $\alpha_{num} = 2.10$, $Re = 6.5 \times 10^6$.

Figure 5.22 Comparison of Upper-Surface Skin Friction Coefficient for the RAE 2822 Airfoil at $M_\infty = 0.725$, $\alpha_{num} = 2.10$, $Re = 6.5 \times 10^6$. 
levels on coarse grids, which is a highly-desirable characteristic of any computational method.

5.6 Aerodynamic Force Coefficients

A summary of computed lift and drag coefficients for the SDG VII method is displayed in Table 5.3. All drag coefficient values are in terms of drag counts where $C_D = 1$ count is equivalent to $C_D = 0.0001$. Since the numerical angle of attack was varied in each case to match computed lift and experimental normal force coefficients, it is important to note that $\alpha_{num}$ was within the accepted corrected angle of attack range for wind tunnel wall-interference effects [33]. The computed coefficient of drag results for the SDG VII scheme were on average predicted approximately 15% lower than experimentally obtained results. The underprediction of drag is more pronounced at lower angle of attack cases. The current state-of-the-art capabilities for attached flow lift and drag predictions are $\pm 3\%$ range for lift and $\pm 5\%$ range for drag [33].
Figure 5.24 Comparison of Lift Coefficient vs. Boundary-Layer Space Width for the NACA 0012 Airfoil - Case A

Figure 5.25 Comparison of Drag Coefficient vs. Average Grid Spacing for the NACA 0012 Airfoil - Case A
Table 5.3  Force Coefficient Comparison

| NACA 0012 Case | Numerical | | Experimental | | |
|---------------|-----------|------------|---------------|-----------|
|               | $C_L$     | $C_D$     | $C_{Dp}$     | $C_{Du}$  | $C_L$  | $C_D$  |
| A             | -0.019    | 58        | -7           | 65        | -0.013 | 77     |
| B             | 0.242     | 70        | 7            | 63        | 0.241  | 79     |

| RAE 2822 Case | Numerical | | Experimental | | |
|---------------|-----------|------------|---------------|-----------|
|               | $C_L$     | $C_D$     | $C_{Dp}$     | $C_{Du}$  | $C_L$  | $C_D$  |
| A             | -0.121    | 70        | 7            | 63        | -0.115 | 79     |
| B             | 0.565     | 75        | 8            | 67        | 0.566  | 85     |
| C             | 0.658     | 86        | 21           | 65        | 0.658  | 107    |
Chapter 6

Conclusions

The application of the Dorodnitsyn transformations to the boundary-layer equations provides a spatial coordinate which automatically follows boundary-layer growth and gives high resolution near the wall which is important in turbulent flows where near-wall velocity gradients are large. The transformed spatial domain is effectively modelled with finite elements to provide accurate results on coarse grids. Modelling the group terms for viscosity by the group finite element method leads to an important economy in the formulation since $\mu_t$ need only be evaluated at the nodes. The use of a non-iterative streamwise marching routine also adds computational economy to the method. The resulting boundary layer code can be used to solve any smoothly-connected airfoil shape without modification and is easily coupled with existing inviscid flow solvers. The transpiration velocity approach used to couple the particularly fast and accurate Euler solver used in this analysis serves as a computationally efficient and easily implemented VII coupling mechanism. Convergence of the overall interaction procedure for both the NACA 0012 and RAE 2822 airfoils was achieved in relatively few global iterations while achieving results of adequate engineering accuracy.

Future work with the SDG method should include the use of a maximum reversed flow velocity concept in order to successfully model separation while still retaining a finite spatial grid. Also, the marching routine could be further improved by possibly using a trigonometric streamwise coordinate transformation to smooth external velocity gradients at the leading and trailing edges of the airfoil.
Appendix A

Galerkin Integrals

\[ K_{1ji} = \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \hat{u} N_j N_{j-1} \, d\hat{u} \]
\[ + \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \hat{u} N_j \, d\hat{u} \]
\[ + \int_{\tilde{u}_j}^{\bar{u}_{j+1}} \hat{u} N_j \, d\hat{u} \]
\[ + \int_{\tilde{u}_j}^{\bar{u}_{j+1}} \hat{u} N_j N_{j+1} \, d\hat{u} \quad (A.1) \]

\[ K_{2ji} = \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] \, d\hat{u} \]
\[ + \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \, d\hat{u} \]
\[ + \int_{\tilde{u}_j}^{\bar{u}_{j+1}} \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] \, d\hat{u} \quad (A.2) \]

\[ K_{3ji} = \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} (1 + \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_{j-1} \, d\hat{u} \]
\[ + \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} (1 + \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_j \, d\hat{u} \]
\[ + \int_{\tilde{u}_j}^{\bar{u}_{j+1}} (1 + \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_j \, d\hat{u} \]
\[ + \int_{\tilde{u}_j}^{\bar{u}_{j+1}} (1 + \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_{j+1} \, d\hat{u} \quad (A.3) \]
\[ K_{4ji} = \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \tilde{u}(1 - \tilde{u}) N_j N_{j-1} \, d\tilde{u} \]
\[ + \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \tilde{u}(1 - \tilde{u}) N_j N_j \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} \tilde{u}(1 - \tilde{u}) N_j N_j \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} \tilde{u}(1 - \tilde{u}) N_j N_{j+1} \, d\tilde{u} \]  \hspace{1cm} (A.4)

\[ K_{5ji} = \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} (1 - \tilde{u}^2) \left[ (1 - \tilde{u}) \frac{dN_j}{d\tilde{u}} - N_j \right] N_{j-1} \, d\tilde{u} \]
\[ + \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} (1 - \tilde{u}^2) \left[ (1 - \tilde{u}) \frac{dN_j}{d\tilde{u}} - N_j \right] N_j \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} (1 - \tilde{u}^2) \left[ (1 - \tilde{u}) \frac{dN_j}{d\tilde{u}} - N_j \right] N_{j+1} \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} (1 - \tilde{u}^2) \left[ (1 - \tilde{u}) \frac{dN_j}{d\tilde{u}} - N_j \right] N_{j+1} \, d\tilde{u} \]  \hspace{1cm} (A.5)

\[ K_{6ji} = \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} (1 - \tilde{u})^2 N_j N_{j-1} \, d\tilde{u} \]
\[ + \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} (1 - \tilde{u})^2 N_j N_j \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} (1 - \tilde{u})^2 N_j N_{j+1} \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} (1 - \tilde{u})^2 N_j N_{j+1} \, d\tilde{u} \]  \hspace{1cm} (A.6)

\[ K_{7ji} = \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \tilde{u}(1 - \tilde{u}) N_j \left[ (1 - \tilde{u}) \frac{dN_{j-1}}{d\tilde{u}} - N_{j-1} \right] \, d\tilde{u} \]
\[ + \int_{\tilde{u}_{j-1}}^{\tilde{u}_j} \tilde{u}(1 - \tilde{u}) N_j \left[ (1 - \tilde{u}) \frac{dN_j}{d\tilde{u}} - N_j \right] \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} \tilde{u}(1 - \tilde{u}) N_j \left[ (1 - \tilde{u}) \frac{dN_j}{d\tilde{u}} - N_j \right] \, d\tilde{u} \]
\[ + \int_{\tilde{u}_j}^{\tilde{u}_{j+1}} \tilde{u}(1 - \tilde{u}) N_j \left[ (1 - \tilde{u}) \frac{dN_{j+1}}{d\tilde{u}} - N_{j+1} \right] \, d\tilde{u} \]  \hspace{1cm} (A.7)
\[ K_{8jik} = \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] d\hat{u} \\
+ \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] d\hat{u} \\
+ \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] d\hat{u} \\
+ \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] d\hat{u} \\
+ \int_{\hat{u}_j}^{\hat{u}_{j+1}} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_j d\hat{u} \\
+ \int_{\hat{u}_j}^{\hat{u}_{j+1}} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] N_j d\hat{u} \\
+ \int_{\hat{u}_j}^{\hat{u}_{j+1}} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] N_j d\hat{u} \\
+ \int_{\hat{u}_j}^{\hat{u}_{j+1}} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] N_{j+1} d\hat{u} \] 

\[ K_{9jik} = \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_{j-1} \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] d\hat{u} \\
+ \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_j \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] d\hat{u} \\
+ \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_{j-1} \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] d\hat{u} \\
+ \int_{\hat{u}_{j-1}}^{\hat{u}_j} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_j \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] d\hat{u} \\
+ \int_{\hat{u}_j}^{\hat{u}_{j+1}} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_j \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] d\hat{u} \\
+ \int_{\hat{u}_j}^{\hat{u}_{j+1}} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_j \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] d\hat{u} \\
+ \int_{\hat{u}_j}^{\hat{u}_{j+1}} (1 - \hat{u}) \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] N_{j+1} \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] d\hat{u} \]
\begin{align*}
K_{10jk} &= N_j N_{j-1} \left[ (1 - \hat{u}) \frac{dN_{j-1}}{d\hat{u}} - N_{j-1} \right] d\hat{u} \\
&+ N_j N_j \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] d\hat{u} \\
&+ N_j N_{j-1} \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] d\hat{u} \\
&+ N_j N_j \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] d\hat{u} \\
&+ N_j N_{j+1} \left[ (1 - \hat{u}) \frac{dN_j}{d\hat{u}} - N_j \right] d\hat{u} \\
&+ N_j N_j \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] d\hat{u} \\
&+ N_j N_{j+1} \left[ (1 - \hat{u}) \frac{dN_{j+1}}{d\hat{u}} - N_{j+1} \right] d\hat{u}
\end{align*}
Appendix B

Boundary Layer Source Code

C CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC
C SEMI-DISCRETE GALERKIN METHOD.
C
C STEADY, COMPRESSIBLE, TURBULENT DORODNITSYN BOUNDARY LAYER.
C
C CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC
C Sdgtn - performs the control of the overall program including
C subroutine calls, counting, and, program termination
C
C CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC CCCC
C Global variable definition:
C
C KT - global stiffness matrix for shear stress variables
C KS - global stiffness matrix for enthalpy variables
C K1 - integral of u f 1/tau du
C K2 - integral of df/du d( sigma tau )/du du
C K3 - integral of (1-u^2) df/du 1/tau du
C K4 - integral of u f s/tau du
C K5 - integral of (1-u^2) df/du s/tau du
C K6 - integral of f sigma tau du
C K7 - integral of u f d( sigma tau )/du du
C K8 - integral of df/du d( sigma tau )/du s du
C K9 - integral of df/du sigma tau ds/du du
C D1 - integral of (1-u) 1/tau du
C D2 - integral of u (1-u) 1/tau du
C D3 - integral of s/tau du
C D4 - integral of 1/(rho tau) du
C FT - load vector for shear stress variables
C FS - load vector for enthalpy variables
C TT - field variable of delta shear stress
C TS - field variable of delta enthalpy
b1 - inverse of nodal value of shear stress variable
b2 - nodal value of shear shear stress variable
b3 - nodal value of enthalpy variable
b4 - nodal value of ratio of enthalpy and shear stress variables
b5 - nodal value of eddy-viscosity variable
b6 - nodal value of eddy-viscosity and shear stress variables
db5db1 - derivative of b5 w/r/t b1
db1dxs - derivative of b1 w/r/t xs
db4dxs - derivative of b4 w/r/t xs
c0 - Chapman Rubesin constant
c1 - 1/ue due
c2 - c0 pe ue / pinf uinf
c3 - c2 / pr
c4 - 2 c2 (gamma - 1)/(gamma + 1) ( 1/pr - 1 )
u - nodal value of xs-velocity
echk - maximum ratio of TT/b1 and TS/b4
theta - implicitness factor
gamma - ratio of specific heats
pr - prantl number
re - reynolds number
sw - boundary condition for s at the wall
twto - ratio of wall and stagnation temperatures
initx - xs distance from stagnation point
delx - xs step size
totx - xc location of trailing edge
xstep - xs distance along airfoil
transx - percent of xc that transition occurs
maxerror - maximum allowable error
delxmax - xs maximum delx
delxmin - xs minimum delx
node - number of nodes in discretization
ntot - counter for number of steps
nout - counter for number of total outputs
ntol - counter for number of outputs under tolerance
pstep - xs distance to write output
stop - flag to indicate separation and exit code
xs - body-normal surface coordinate
xc - cartesian x-coordinate
yc - cartesian y-coordinate
uei - inviscid edge velocity
dueixs - derivative of uei wrt xs
pei - inviscid edge pressure
C dpeixs - derivative of pei wrt xs
C rhoei - inviscid edge density
C drhoeixs - derivative of rhoei wrt xs
C nei - number of external inviscid data points
C ile - external data location of leading edge
C itel - external data location of the trailing edge on lower surface
C iteu - external data location of the trailing edge on upper surface
C istag - external data location of the stagnation point
C minf - freestream mach number
C uinf - freestream mach number
C pinf - freestream pressure
C aoa - angle of attack
C xe - cartesian x-distance
C ye - cartesian y-distance
C ue - velocity at the boundary layer edge
C due - derivative of velocity at the boundary layer edge wrt xs
C pe - pressure at the boundary layer edge
C dpe - derivative of pressure at the boundary layer edge wrt xs
C rhoe - density at the boundary layer edge
C drhoe - derivative of density at the boundary layer edge wrt xs
C friction - coefficient of friction
C dsthick - displacement thickness
C transvel - transpiration velocity
C blthick - boundary-layer thickness
C mothick - momentum thickness
C seprb1 - value of b1 that indicates separation
C extrb1 - value of b1 that indicates need for extrapolation
C relax - scaling parameter for relaxed transpiration velocity
C elm - flag to indicate error tolerance exception
C side - flag to indicate upper=1 or lower=2 side of airfoil
C inum - current number of vii iteration
C

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C program sdgm
C
dimension K1(50,4), K2(50,4), K3(50,4), K4(50,4), K5(50,4)
dimension K6(50,4), K7(50,4), K8(50,8), K9(50,8)
dimension KT(50,50), KS(50,50), FT(50), FS(50), TT(50), TS(50)
dimension b1(50), b2(50), b3(50), b4(50), db1dxs(50), db4dxs(50)
dimension b5(50), b6(50), db5db1(50), D1(50), D2(50), D3(50), D4(50)
dimension c0(2), c1(2), c2(2), c3(2), c4(2), u(50)
dimension xs(200), xc(200), yc(200), uei(200), dueixs(200)
dimension pei(200), dpeixs(200), rhoei(200), drhoeixs(200)
real K1, K2, K3, K4, K5, K6, K7, K8, K9
real KT, KS, FT, FS, TT, TS
real b1, b2, b3, b4, b5, b6, dbdxs, dbdys, db5dxs, db5ds, u
real D1, D2, D3, D4, xs, xc, yc
real c0, c1, c2, c3, c4, sw, twto
real uei, dueixs, pei, dpeixs, rhoei, drhoeixs
real initx, delx, totx, xstep, transx
real gamma, minf, uinf, pinf, pr, re
real xe, ye, ue, due, pe, dpe, rhoe, drhoe
real maxerror, delxmax, delxmin, echk
real theta, aoa, pstep, seprb1, extrb1, relax
real friction, dthick, transv, blthick, mothick
integer node, ntot, nout, ntol, stop, elim, nei, ile, itel, iteu
integer istag, side, inum

side = 1
continue

C

9

C

call sstart( pstep, ntot, nout, ntol, stop, elim, side )
call external( xs, xc, yc, uei, dueixs, pei, dpeixs, rhoei, drhoeixs,
1            uinf, pinf, minf, aoa, gamma, nei, ile, itel, iteu,
2            istag )
call input( theta, initx, totx, delx, transx, maxerror, pr, re,
1            seprb1, extrb1, relax, node, inum )
call calcux( u, initx, xstep, delx, delxmin, delxmax, totx, xs,
1            xc, transx, nei, istag, side, node )
call array1( K1, 1, u, node )
call array1( K2, 2, u, node )
call array1( K3, 3, u, node )
call array1( K4, 4, u, node )
call array1( K5, 5, u, node )
call array1( K6, 6, u, node )
call array1( K7, 7, u, node )
call array2( K8, 8, u, node )
call array2( K9, 9, u, node )
call array3( D1, 1, u, node )
call array3( D2, 2, u, node )
call array3( D3, 3, u, node )
call constants( xstep, delx, xs, xc, yc, uei, dueixs, pei, dpeixs,
1            rhoei, drhoeixs, xe, ye, ue, due, pe, dpe, rhoe,
2            drhoe, c0, c1, c2, c3, c4, twto, sw, gamma, pr,
call similar( b1, b2, b3, b4, c0, u, uinf, pinf, ue, pe, xstep, pr, 
    gamma, twto, node, inum )
    call displ( D1, D2, D3, b1, b4, dstick, uinf, pinf, gamma, re, ue, 
    pe, node )
    call cebsmith( D4, b2, b3, b5, b6, dbdb1, c0, u, ue, pe, dpe, 
    uinf, pinf, dstick, xstep, transx, gamma, re, 
    node )

    continue
    call assembleKT( KT, K1, K2, K3, c1, c2, b1, b2, b5, dbdb1, 
    theta, delx, node )
    call assembleFT( FT, K2, K3, c1, c2, b1, b6, theta, delx, 
    node )
    call solve( KT, FT, TT, node )
    if( pr.ne.1.0.and.twto.ne.1.0 ) then
        call assembleKS( KS, K4, K5, K8, K9, b2, b6, c1, c2, 
        c3, theta, delx, node )
        call assembleFS( TS, FS, K4, K5, K6, K7, K8, K9, b3, 
        b4, b6, c1, c2, c3, c4, theta, delx, 
        sw, node )
        call solve( KS, FS, TS, node )
    end if
    call error( TT, TS, b1, b4, echk, node )
    call calcxstep( delx, echk, maxerror, delxmin, delxmax, elim )
    if( echk.gt.maxerror.and.elim.ne.1 ) then
        call constants( xstep, delx, xs, xc, yc, uei, dueixs, 
        pei, dpeixs, rhoi, drhoeixs, xe, ye, 
        ue, dua, pe, dpe, rhoe, drhoe, c0, c1, 
        c2, c3, c4, twto, sw, gamma, pr, uinf, 
        pinf, ne, istag, side )
        goto 99
    end if
    if( pstep.ge.delxmax.or.ntot.eq.1 ) then
        call calcdbidxs( K1, K2, K3, c1, c2, b1, b6, dbidxs, 
        node )
        call calcdb4dxs( K4, K5, K6, K7, K8, K9, b3, b4, b6, 
        c1,c2, c3, c4, dbidxs, db4dxs, sw, 
        node )
        call trans( D1, D2, D3, b1, dbidxs, b4, db4dxs, 
        transvel, uinf, gamma, re, ue, dua, 
        pe, rhoe, drhoe, node )
        call bound( D4, b2, b3, b1thick, u, uinf, gamma, re, 
        node )
ue, pe, node

call momen( D2, b1, mothick, uinf, ue, rhoe, re, node )
call output( TT, TS, b1, b2, b3, b4, echk, xstep, xe,
1
ye, ue, due, pe, dpe, rhoe, drhoe, re,
2
uinf, pinf, c0, friction, dsthick,
3
transvel, blthick, mothick, u, pstep,
4
node, stop, ntot, nout, ntol )
end if

if( xstep.ge.totx.or.stop.eq.2 ) goto 999

call increment( b1, b2, b3, b4, TT, TS, xstep, delx, stop,
1
seprb1, extrb1, pstep, node, ntot )
call constants( xstep, delx, xs, xc, yc, uei, dueixs, pei,
1
dpeixs, rhoei, drhoeixs, xe, ye, ue, due, pe,
2
dpe, rhoe, drhoe, c0, c1, c2, c3, c4, twto,
3
sw, gamma, pr, uinf, pinf, neii, istag, side )
call displ( D1, D2, D3, b1, b4, dsthick, uinf, pinf, gamma,
1
re, ue, pe, node )
call cebsmith( D4, b2, b3, b5, b6, db5db1, c0, u, ue, pe,
1
dpe, uinf, pinf, dsthick, xstep, transx,
2
gamma, re, node )
goto 99

999 continue

call calcdb1dxs( K1, K2, K3, c1, c2, b1, b6, db1dxs, node )
call calcdb4dxs( K4, K5, K6, K7, K8, K9, b3, b4, b6, c1, c2,
1
c3, c4, db1dxs, db4dxs, sw, node )
call trans( D1, D2, D3, b1, db1dxs, b4, db4dxs, transvel, uinf, gamma,
1
re, due, pe, rhoe, drhoe, node )
call bound( D4, b2, b3, blthick, u, uinf, gamma, re, ue, pe, node )
call momen( D2, b1, mothick, uinf, ue, rhoe, re, node )
call output( TT, TS, b1, b2, b3, b4, echk, xstep, xe, ye, ue, due, pe,
1
dpe, rhoe, drhoe, re, uinf, pinf, c0, friction, dsthick,
2
transvel, blthick, mothick, u, pstep, node, stop,
3
ntot, nout, ntol )
call send( nout, ntol, stop )
if( side.eq.1 ) then
   side = 2
   goto 9
end if
call stagconv
call bl2ext( pinf )
call relaxv( relax, neii, inum )
call liftdrag( aoa )
C
stop 'viscous pass completed'
end

C

Sstart - subroutine to perform initializations and open data files
C

Local variable definition:
C

subroutine sstart( pstep, ntot, nout, ntol, stop, elim, side )
C
real pstep
integer ntot, nout, ntol, stop, elim, side
C

if( side.eq.1 ) then
  open( unit=15, file='ext2bl.dat', status='old' )
  open( unit=16, file='sdginput1.dat', status='old' )
  open( unit=17, file='siminput1.dat', status='old' )
  open( unit=18, file='sdgmout1.dat', status='unknown' )
  open( unit=19, file='integral1.dat', status='unknown' )
  open( unit=20, file='fcoeff1.dat', status='unknown' )
  open( unit=21, file='points1.dat', status='unknown' )
  open( unit=22, file='external.dat', status='unknown' )
  open( unit=23, file='iteration.dat', status='unknown' )
  open( unit=24, file='initial1.dat', status='unknown' )
else if( side.eq.2 ) then
  open( unit=15, file='ext2bl.dat', status='old' )
  open( unit=16, file='sdginput2.dat', status='old' )
  open( unit=17, file='siminput2.dat', status='old' )
  open( unit=18, file='sdgmout2.dat', status='unknown' )
  open( unit=19, file='integral2.dat', status='unknown' )
  open( unit=20, file='fcoeff2.dat', status='unknown' )
  open( unit=21, file='points2.dat', status='unknown' )
  open( unit=22, file='external.dat', status='unknown' )
  open( unit=23, file='iteration.dat', status='unknown' )
  open( unit=24, file='initial2.dat', status='unknown' )
end if
C
  ntot = 1
  nout = 0
  ntol = 0
  stop = 0
  elim = 0
  pstep = 0.0
C
  return
end

C
C     Send - subroutine to close data files
C
C
C     Local variable definition:
C
C
C     subroutine send( nout, ntol, stop )
C
C     integer nout, ntol, stop
C
C     write(21,*) nout, ntol
C     write(*,*) 'stop = ', stop
C
C     close(15)
C     close(16)
C     close(17)
C     close(18)
C     close(19)
C     close(20)
C     close(21)
C     close(22)
C     close(23)
C     close(24)
C
C     return
end
External subroutine reads in the external variables from data file ext2bl.dat and converts the external conditions based on a cartesian coordinate system and "other" non-dimensionalization to a body normal coordinate system based on the boundary layer non-dimensionalization

Local variable definition:

isearch = number of data points away from the leading edge to search for the stagnation point (u_tangential approx 0)
vei = cartesian y component of velocity

subroutine external( xs, xc, yc, uei, dueixs, pei, dpeixs, rhoei,
1       drhoeixs, uinf, pinf, minf, aoa, gamma,
2       nei, ile, itel, iteu, istag )

dimension xc(200), yc(200), xs(200), uei(200), dueixs(200), pei(200)
dimension dpeixs(200), rhoei(200), drhoeixs(200), vei(200)
real xc, xs, uei, dueixs, pei, dpeixs, rhoei, drhoeixs
real yc, minf, aoa, uinf, pinf, gamma, vei, ustag, theta
integer nei, ile, itel, iteu, istag, isearch, i

read(15,*) nei, ile, itel, iteu, gamma, minf, aoa
read(15,*) ( xc(i), yc(i), uei(i), vei(i), pei(i), rhoei(i),
1       i = 1, nei )

------------------------------------------------------------------------

calculate freestream conditions
------------------------------------------------------------------------

pinf = (gamma+1.0)/(2.0*gamma + gamma*(gamma-1.0)*minf**2.0))
uminf = sqrt(gamma)*minf*sqrt(pinf)

------------------------------------------------------------------------

convert external conditions based on a cartesian coordinate system
and "other" non-dimensionalization to a body normal coordinate system
C based on the boundary layer nondimensionalization
C calculate body normal velocity and local body angle
C---------------------------------------------------------------
C
do 100 i = 1, nei
   if( uei(i).ne.0.0 ) then
      theta = atan(vei(i)/uei(i))
   else
      theta = 3.1415926/2.0
   end if
   uei(i) = abs(uei(i)*cos(theta) + vei(i)*sin(theta))
100 continue
C
C---------------------------------------------------------------
C find stagnation point
C---------------------------------------------------------------
C
isearch = int(0.1*(ile))
ustag = 1.0e6
do 200 i = ile-isearch, ile+isearch
   if( uei(i).le.ustag ) then
      ustag = uei(i)
      istag = i
   end if
200 continue
C
C---------------------------------------------------------------
C calculate surface distance
C---------------------------------------------------------------
C
xs(istag) = 0.0
do 300 i = istag - 1, 1, -1
   xs(i) = xs(i+1) + sqrt((xc(i)-xc(i+1))**2.0 + (yc(i)-yc(i+1))**2.0)
300 continue
do 400 i = istag + 1, nei
   xs(i) = xs(i-1) + sqrt((xc(i)-xc(i-1))**2.0 + (yc(i)-yc(i-1))**2.0)
400 continue
C
C---------------------------------------------------------------
C calculate derivative of external conditions w/r/t surface distance
C--------------------------------------------------------------------------------

C
dueixs(i) = 0.0
dpeixs(i) = 0.0
drhoeixs(i) = 0.0
dueixs(nei) = 0.0
dpeixs(nei) = 0.0
drhoeixs(nei) = 0.0

500 do 500 i = istag, 2, -1
   dueixs(i) = (uei(i-1)-uei(i))/(xs(i-1)-xs(i))
   dpeixs(i) = (pei(i-1)-pei(i))/(xs(i-1)-xs(i))
   drhoeixs(i) = (rhoei(i-1)-rhoei(i))/(xs(i-1)-xs(i))
continue

500 do 600 i = istag, nei-1
   dueixs(i) = (uei(i+1)-uei(i))/(xs(i+1)-xs(i))
   dpeixs(i) = (pei(i+1)-pei(i))/(xs(i+1)-xs(i))
   drhoeixs(i) = (rhoei(i+1)-rhoei(i))/(xs(i+1)-xs(i))
continue

C
C--------------------------------------------------------------------------------
C convert to boundary layer nondimensionalization
C--------------------------------------------------------------------------------

C
do 700 i = 1, nei
   uei(i) = sqrt(pinf)*uei(i)
   pei(i) = pinf*pei(i)
   rhoei(i) = 1.0*rhoei(i)
   dueixs(i) = sqrt(pinf)*dueixs(i)
   dpeixs(i) = pinf*dpeixs(i)
   drhoeixs(i) = 1.0*drhoeixs(i)

700 continue

C
C--------------------------------------------------------------------------------
C write out converted external conditions
C--------------------------------------------------------------------------------

C
write(22,800) nei, i1e, itel, iteu, gamma, minf, aca
write(22,900) ( xc(i), yc(i), xs(i), uei(i), dueixs(i), pei(i),
   1 dpeixs(i), rhoei(i), drhoeixs(i), i = 1, nei )
800 format( 4i4, 3e15.6 )
900 format( 9e15.6 )
return
end

Input - subroutine reads in the seen variables from the data file
sdgminput[1-2].dat

Local variable definition:

subroutine input( theta, initx, totx, delx, transx, maxerror, 
  1         pr, re, seprbi, extrbi, relax, node, inum )

real theta, initx, pr, re, relax
real totx, delx, maxerror, transx, seprbi, extrbi
integer node, inum

read(16,*), node
read(16,*), theta
read(16,*), pr
read(16,*), re
read(16,*), initx
read(16,*), totx
read(16,*), delx
read(16,*), transx
read(16,*), maxerror
read(16,*), extrbi
read(16,*), seprbi
read(16,*), relax

read(23,*), inum

return
end
Calcux - subroutine calculates the x-velocity distribution across
the boundary layer, the initial x-position, maximum and
minimum allowable x-steps

Local variable definition:

subroutine calcux( u, initx, xstep, delx, delxmin, delxmax, totx,
1     xs, xc, transx, ne, istag, side, node )

dimension u(50), xs(50), xc(50)
real u, initx, xstep, delx, delxmin, delxmax, totx
real dum1, delu, xm, xs, xc, transx
integer i, node, ii, ne, ipos1, side, istag

dum1 = node - 1
delu = 1.0 / real(dum1)
u(1) = 0.0
do 100 i = 2, node
   u(i) = u(i-1) + delu
100   continue

xm = totx
if( side.eq.1 ) then
   do 200 ii = istag, ne
      if( xc(ii).ge.xm ) goto 20
200   continue
20   ipos1 = ii
    totx = (((x - xc(ipos1-1)) * (xs(ipos1) - xs(ipos1-1)))
1    /(xc(ipos1) - xc(ipos1-1))) + xs(ipos1-1)
else if( side.eq.2 ) then
   do 300 ii = istag, 1, -1
      if( xc(ii).ge.xm ) goto 30
300   continue
30   ipos1 = ii
    totx = (((x - xc(ipos1+1)) * (xs(ipos1) - xs(ipos1+1)))
1    /(xc(ipos1) - xc(ipos1+1))) + xs(ipos1+1)
end if
xstep = initx
delxmin = delx / 1000.0
delxmax = (totx - initx) / real(iei-1)
transx = (transx/100.0)*(totx-initx)

return
end

Solve - solve a tri-diagonal set of linear algebraic equations
using the LU-decomposition method

Local variable definition:
i - dummy index
u - upper decomposition
l - lower decomposition
d - diagonal decomposition
y - forward reduction variable
stiff - stiffness matrix
load - load vector
field - field variable

subroutine solve(stiff, load, field, node )

dimension stiff(50,50), load(50), field(50)
dimension u(50), l(50), d(50), y(50)
real stiff, load, field
real u, l, d, y
integer node
integer i

perform LU decomposition

d(1) = stiff(1,1)
do 100 i = 1, node-1
   u(i) = stiff(i,i+1)
100   continue
    do 200 i = 2, node
       l(i) = stiff(i,i-1) / d(i-1)
       d(i) = stiff(i,i) - ( l(i)*stiff(i-1,i) )
200   continue
C
C---------------------------------------------------------------
C perform forward reduction
C---------------------------------------------------------------
C
   y(1) = load(1)
    do 300 i = 2, node
       y(i) = load(i) - ( l(i)*y(i-1) )
300   continue
C
C---------------------------------------------------------------
C perform backward substitution
C---------------------------------------------------------------
C
   field(node) = y(node) / d(node)
    do 400 i = node-1, 1, -1
       field(i) = ( y(i) - ( u(i)*field(i+1) ) ) / d(i)
400   continue
C
return
end

C
C Constants - subroutine used to interpolate the external data and
C    calculate coefficients used in the integral equations
C
C
C
C
C Local variable definition:
C
C
C
C
xm - xstep at step (i)
C
C
C
C
xe - cartesian x distance along airfoil (i)
C
C
C
ue - velocity at edge of boundary layer (i)
C uen - velocity at edge of boundary layer (i+1)
C due - derivative of velocity w/r/t xs at edge of boundary layer (i)
C duen - derivative of velocity w/r/t xs at edge of boundary layer (i+1)
C pe - pressure at edge of boundary layer (i)
C pen - pressure at edge of boundary layer (i+1)
C dpe - derivative of pressure w/r/t xs at edge of boundary layer (i)
C dpen - derivative of pressure w/r/t xs at edg of boundary layer (i+1)
C rho - density at edge of boundary layer (i)
C rhoen - density at edge of boundary layer (i+1)
C drhoe - derivative of density w/r/t xs at edge of boundary layer(i)
C drhoen - derivative of density w/r/t xs at edge of boundary layer (i+1)
C

subroutine constants( xstep, delx, xs, xc, yc, uei, dueixs, pei,
  dpeixs, rhoei, drhoeixs, xe, ye, ue, due,
  pe, dpe, rhoe, drhoe, c0, c1, c2, c3, c4,
  twto, sw, gamma, pr, uinf, pinf, nei, istag,
  side )
C
dimension xs(200), xc(200), uei(200), dueixs(200), pei(200)
dimension dpeixs(200), rhoei(200), drhoeixs(200), yc(200)
dimension c0(2), c1(2), c2(2), c3(3), c4(3)
real c0, c1, c2, c3, c4
real delx, xstep, uinf, pinf
real xs, xc, uei, dueixs, pei, dpeixs, rhoei, drhoeixs
real xe, ue, uen, due, duen, pe, pen, dpe, dpen
real rhoe, rhoen, drhoe, drhoen, yc, ye
real xm, xmn, pr, gamma, twto, sw
integer nei, istag, side, i1, i2, ipos1, ipos2
C
C xm = xstep
C xmn = xstep + delx
C
C if( side.eq.1 ) then
C do 100 i1 = istag, nei
C     if( xs(i1).ge.xm ) goto 10
100  continue
10  ipos1 = i1
C do 200 i2 = istag, nei
if( xs(i2) ge xmn ) goto 20

20  continue

i = 12

xe = (((xm - xs(ipos1-1)) * (xc(ipos1) - xc(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + xc(ipos1-1)
ye = (((xm - xs(ipos1-1)) * (yc(ipos1) - yc(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + yc(ipos1-1)
ue = (((xm - xs(ipos1-1)) * (ue(ipos1) - ue(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + ue(ipos1-1)
uen = (((xm - xs(ipos2-1)) * (uei(ipos2) - uei(ipos2-1)))
1 / (xs(ipos2) - xs(ipos2-1))) + uei(ipos2-1)
due = (((xm - xs(ipos1-1)) * (dueixs(ipos1) - dueixs(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + dueixs(ipos1-1)
duen = (((xm - xs(ipos2-1)) * (dueixs(ipos2) - dueixs(ipos2-1)))
1 / (xs(ipos2) - xs(ipos2-1))) + dueixs(ipos2-1)
pe = (((xm - xs(ipos1-1)) * (pei(ipos1) - pei(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + pei(ipos1-1)
pen = (((xm - xs(ipos2-1)) * (pei(ipos2) - pei(ipos2-1)))
1 / (xs(ipos2) - xs(ipos2-1))) + pei(ipos2-1)
dpe = (((xm - xs(ipos1-1)) * (dpeixs(ipos1) - dpeixs(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + dpeixs(ipos1-1)
dpen = (((xm - xs(ipos2-1)) * (dpeixs(ipos2) - dpeixs(ipos2-1)))
1 / (xs(ipos2) - xs(ipos2-1))) + dpeixs(ipos2-1)
rho = (((xm - xs(ipos1-1)) * (rhoi(ipos1) - rhoi(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + rhoi(ipos1-1)
rhoen = (((xm - xs(ipos2-1)) * (rhoi(ipos2) - rhoi(ipos2-1)))
1 / (xs(ipos2) - xs(ipos2-1))) + rhoi(ipos2-1)

drho = (((xm - xs(ipos1-1)) * (drhoixs(ipos1) - drhoixs(ipos1-1)))
1 / (xs(ipos1) - xs(ipos1-1))) + drhoixs(ipos1-1)
drhoen = (((xm - xs(ipos2-1)) * (drhoixs(ipos2) - drhoixs(ipos2-1)))
1 / (xs(ipos2) - xs(ipos2-1))) + drhoixs(ipos2-1)

C

else if( side.eq.2 ) then
    do 300 i1 = istag, 1, -1
        if( xs(i1) ge xmn ) goto 30

300  continue

30   ipos1 = i1
    do 400 i2 = istag, 1, -1
        if( xs(i2) ge xmn ) goto 40

400  continue

40   ipos2 = i2
    xe = (((xm - xs(ipos1+1)) * (xc(ipos1) - xc(ipos1+1)))

C

c0(1) = 1.0
c0(2) = 1.0
c1(1) = (1.0/ue)*due

c1(2) = (1.0/uen)*duen
c2(1) = (c0(1) * ue * pe) / (uinf * pinf)
c2(2) = (c0(2) * uen * pen) / (uinf * pinf)
c3(1) = c2(1)/pr

c3(2) = c2(2)/pr

c4(1) = 2.0*c2(1)*((gamma-1.0)/(gamma+1.0))*(1.0/pr - 1)
c4(2) = 2.0*c2(2)*((gamma-1.0)/(gamma+1.0))*(1.0/pr - 1)

sw = (1.0 - twto)*(1.0/ue**2.0)

return
end

Error - check ratio of b1 to TT and b4 to TS

Local variable definition:

echk1 - maximum ratio of TT/b1 for momentum integral equation
echk2 - maximum ratio of TS/b4 for energy integral equation
tratio1 - current ratio of TT/b1
tratio2 - current ratio of TS/b4

subroutine error( TT, TS, b1, b4, echk, node )

dimension TT(50), TS(50), b1(50), b4(50), ratio1(50), ratio2(50)
real TT, TS, b1, b4, ratio1, ratio2, echk, echk1, echk2
integer i, node

echk1 = 0.0
echk2 = 0.0
echk = 0.0

do 100 i = 1, node
    if( b1(i).ne.0.0 ) ratio1(i) = TT(i)/b1(i)
    if( b4(i).ne.0.0 ) ratio2(i) = TS(i)/b4(i)
    if( abs(ratio1(i)).ge.echk1 ) echk1 = abs(ratio1(i))
    if( abs(ratio2(i)).ge.echk2 ) echk2 = abs(ratio2(i))
    continue
 100  if( echk1.ge.echk2 ) then
        echk = echk1
      else
        echk = echk2
    end if

return
end
C Calcxstep - subroutine to vary the step size
C
C Local variable definition:
C
minerror - minimum error bound
C dmax = potential delta xs if increasing step size
C dmin = potential delta xs if decreasing step size
C
C subroutine calcxstep( delx, echk, maxerror, delxmin, delxmax, elim )
C
real delx, maxerror, delxmin, delxmax, echk
real minerror, dmax, dmin
integer elim
C
minerror = maxerror * 0.5
C dmax = delx * 1.5
C dmin = delx / 2.0
if( echk.lt.minerror.and.dmax.lt.delxmax ) then
   delx = delx * 1.5
   write( *, * ) "Increasing the step size ... ", delx
elem = 0
else if( echk.gt.maxerror.and.dmin.gt.delxmin ) then
   delx = delx / 2.0
   write( *, * ) "Decreasing the step size ... ", delx
elem = 0
else if( echk.gt.maxerror.and.dmin.lt.delxmin ) then
   write( *, * ) "Warning minimum step size ... 
   elem = 1
end if
C
return
end
C
C Output - subroutine to write output to sdgmout[1-2].dat
Local variable definition:

tau - shear stress variable
s - entropy variable
dum1 - dummy variable for interim calculation

subroutine output( TT, TS, b1, b2, b3, b4, echk, xstep, xe, ye, ue,
due, pe, dpe, rhoe, drhoe, re, uinf, pinf, c0,
friction, dstick, transvel, blthick, mothick,
u, pstep, node, stop, ntot, nout, ntol )

dimension TT(50), TS(50), b1(50), b2(50), b3(50), b4(50)
dimension tau(50), s(50), u(50), c0(2)
real TT, TS, b1, b2, b3, b4, echk, u, friction, dstick
real transvel, tau, s, xe, ye, ue, pe, uinf, pinf, c0
real re, due, dpe, rhoe, drhoe, pstep, blthick, mothick
real dum1, xstep
integer node, stop, ntot, nout, ntol
integer i

nout = nout + 1
if( stop.ne.1 ) ntol = ntol + 1
pstep = 0.0

do 100 i = 1, node
   tau(i) = (1.0-u(i))*b2(i)
s(i) = (1.0-u(i))*b3(i)

100 continue

calculate coefficient of friction

based on freestream dynamic pressure
dum1 = (2.0*c0(1)/sqrt(re))*(ue/uinf)**2.0*(pe/pinf)

based on boundary layer edge dynamic pressure
dum1 = (2.0*c0(1)/sqrt(re))*(pe/pinf)*1.0/rhoe

dum1 = (2.0*c0(1)/sqrt(re))*(ue/uinf)**2.0*(pe/pinf)
friction = dum1*tau(i)

C
C-----------------------------------------------
C  write nodal value histories to sdgmout[1-2].dat
C-----------------------------------------------
C
write(18,*) 'n = ', ntot
write(18,*) 'x = ', xe
write(18,*) 'e = ', echk
write(18,*) '    TT        tau      TS
   s''
write(18,200) ( TT(i), tau(i), TS(i), s(i), i = 1, node )
200
format( 4(f15.6) )

C
C-----------------------------------------------
C  write coefficient of friction, displacement thickness, transpiration
C  velocity, boundary layer thickness, and momentum thickness data
C-----------------------------------------------
C
write(19,300) xe, ye, xstep, dthick, transvel, blthick, mothick
300
format(7e15.6)
write(20,400) xe, ye, friction
400
format(3e15.6)

C return
end

C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C
C
C Increment - subroutine to increment the nodal variables, streamwise
C location, counters, and determine if the wall shear
C stress has dropped below a tolerance to indicate
C separation
C
C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C
C
C Local variable definition:
C
C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C C
C
C subroutine increment( b1, b2, b3, b4, TT, TS, xstep, delx, stop,
1  seprb1, extrb1, pstep, node, ntot )
C

dimension b1(50), b2(50), b3(50), b4(50), TT(50), TS(50)
real b1, b2, b3, b4, TT, TS
real xstep, delx, seprb1, extrb1, pstep
integer node, stop, ntot
integer i

C

ntot = ntot + 1
xstep = xstep + delx
pstep = pstep + delx
do 100 i = 1, node
   b1(i) = b1(i) + TT(i)
   b2(i) = 1.0/b1(i)
   b4(i) = b4(i) + TS(i)
   b3(i) = b2(i)*b4(i)
100 continue
if( b1(i).gt.extrb1 ) stop = 1
if( b1(i).gt.seprb1 ) stop = 2

C

return
end

C

Qgaussi - subroutine to perform 10 pt gaussian quadrature numerical
integration for arrays with two indices

C

local variable definition:

C

a - lower limit of integration
b - upper limit of integration
ss - current value of integration
pt - gaussian quadrature point
wt - gaussian weight
xm - half of the sum of the limits of integration
xr - half of the difference of the limits of integration
dx - product of xr and the gaussian point
f = flag to specify which shape function to utilize
g = flag to specify which integral to utilize
i = specific nodal position
j - integer counter

subroutine qgauss1( a, b, ss, u, f, g, i )

dimension u(50)
dimension pt(5), wt(5)
real a, b, ss, u
real pt, wt, xm, xr, dx, func1
integer f, i, g
integer j
data pt/1.1488743389, 0.4333953941, 0.6794096682, 0.8650633666, 0.9739065286/
data wt/0.2955242247, 0.2692667193, 0.2190863626, 0.1494613491, 0.0666713443/

xm = 0.5 * ( b + a )
xr = 0.5 * ( b - a )
ss = 0.0

do 100 j = 1, 5
   dx = xr * pt(j)
   ss = ss + ( wt(j) * ( func1(xm+dx,u,f,g,i)
     + func1(xm-dx,u,f,g,i)) )

100 continue

ss = xr*ss

return
end

Array1 - subroutine to calculate arrays with two indices

Local variable definition:

ss - current value of integration
f = flag to specify which shape function to utilize
g = flag to specify which integral to utilize
i - specific nodal position
subroutine array1(K, g, u, node)

dimension K(50,4), u(50)
real K, u
real ss
integer node
integer f, i, g

do 100 i = 1, node
   if( i.eq.1 ) then
      f = 3
      call qgauss1(u(i), u(i+1), ss, u, f, g, i)
      K(i,3) = ss
      f = 4
      call qgauss1(u(i), u(i+1), ss, u, f, g, i)
      K(i,4) = ss
   else if( i.gt.1.and.i.lt.node ) then
      f = 1
      call qgauss1(u(i-1), u(i), ss, u, f, g, i)
      K(i,1) = ss
      f = 2
      call qgauss1(u(i-1), u(i), ss, u, f, g, i)
      K(i,2) = ss
      f = 3
      call qgauss1(u(i), u(i+1), ss, u, f, g, i)
      K(i,3) = ss
      f = 4
      call qgauss1(u(i), u(i+1), ss, u, f, g, i)
      K(i,4) = ss
   else if( i.eq.node ) then
      f = 1
      call qgauss1(u(i-1), u(i), ss, u, f, g, i)
      K(i,1) = ss
      f = 2
      call qgauss1(u(i-1), u(i), ss, u, f, g, i)
      K(i,2) = ss
   end if

continue

return
funcl - function for arrays with two indices

Local variable definition:

w = current value of u (x-velocity)
f = flag to specify which shape function to utilize
g = flag to specify which integral to utilize
i - specific nodal position
sh1 - shape function N i-1
sh2 - shape function N i-
sh3 - shape function N i+
sh4 - shape function N i+1
dsh1 - derivative of shape function N i-1
dsh2 - derivative of shape function N i-
dsh3 - derivative of shape function N i+
dsh4 - derivative of shape function N i+1

real function funcl( w, u, f, g, i )

dimension u(50)
real w, u
real sh1, sh2, sh3, sh4, dsh1, dsh2, dsh3, dsh4
integer f, i, g

if( f.eq.1 ) then
  sh1 = (w-u(i))/(u(i-1)-u(i))
  dsh1 = (1.0/(u(i-1)-u(i)))
  sh2 = (w-u(i-1))/(u(i)-u(i-1))
  dsh2 = (1.0/(u(i)-u(i-1)))
else if( f.eq.2 ) then
  sh2 = (w-u(i-1))/(u(i)-u(i-1))
  dsh2 = (1.0/(u(i)-u(i-1)))
else if( f.eq.3 ) then
  sh3 = (w-u(i+1))/(u(i)-u(i+1))
dsh3 = (1.0/(u(i)-u(i+1)))
else if( f.eq.4 ) then
    sh3 = (w-u(i+1))/(u(i)-u(i+1))
    dsh3 = (1.0/(u(i)-u(i+1)))
    sh4 = (w-u(i))/(u(i+1)-u(i))
    dsh4 = (1.0/(u(i+1)-u(i)))
end if

C

if( g.eq.1 ) then
if( f.eq.1 ) then
    func1 = w*sh2*sh1
else if( f.eq.2 ) then
    func1 = w*sh2*sh2
else if( f.eq.3 ) then
    func1 = w*sh3*sh3
else if( f.eq.4 ) then
    func1 = w*sh3*sh4
end if

C

else if( g.eq.2 ) then
if( f.eq.1 ) then
    func1 = ((1.0-w)*dsh2-sh2)*((1.0-w)*dsh1-sh1)
else if( f.eq.2 ) then
    func1 = ((1.0-w)*dsh2-sh2)*((1.0-w)*dsh2-sh2)
else if( f.eq.3 ) then
    func1 = ((1.0-w)*dsh3-sh3)*((1.0-w)*dsh3-sh3)
else if( f.eq.4 ) then
    func1 = ((1.0-w)*dsh3-sh3)*((1.0-w)*dsh4-sh4)
end if

C

else if( g.eq.3 ) then
if( f.eq.1 ) then
    func1 = (1.0+w)*((1.0-w)*dsh2-sh2)*sh1
else if( f.eq.2 ) then
    func1 = (1.0+w)*((1.0-w)*dsh2-sh2)*sh2
else if( f.eq.3 ) then
    func1 = (1.0+w)*((1.0-w)*dsh3-sh3)*sh3
else if( f.eq.4 ) then
    func1 = (1.0+w)*((1.0-w)*dsh3-sh3)*sh4
end if

C

else if( g.eq.4 ) then

if( f.eq.1 ) then
    func1 = w*(1.0-w)*sh2*sh1
else if( f.eq.2 ) then
    func1 = w*(1.0-w)*sh2*sh2
else if( f.eq.3 ) then
    func1 = w*(1.0-w)*sh3*sh3
else if( f.eq.4 ) then
    func1 = w*(1.0-w)*sh3*sh4
end if

else if( g.eq.5 ) then
if( f.eq.1 ) then
    func1 = (1.0-w**2.0)*((1.0-w)*dsh2-sh2)*sh1
else if( f.eq.2 ) then
    func1 = (1.0-w**2.0)*((1.0-w)*dsh2-sh2)*sh2
else if( f.eq.3 ) then
    func1 = (1.0-w**2.0)*((1.0-w)*dsh3-sh3)*sh3
else if( f.eq.4 ) then
    func1 = (1.0-w**2.0)*((1.0-w)*dsh3-sh3)*sh4
end if

else if( g.eq.6 ) then
if( f.eq.1 ) then
    func1 = ((1.0-w)**2.0)*sh2*sh1
else if( f.eq.2 ) then
    func1 = ((1.0-w)**2.0)*sh2*sh2
else if( f.eq.3 ) then
    func1 = ((1.0-w)**2.0)*sh3*sh3
else if( f.eq.4 ) then
    func1 = ((1.0-w)**2.0)*sh3*sh4
end if

else if( g.eq.7 ) then
if( f.eq.1 ) then
    func1 = w*(1.0-w)*sh2*((1.0-w)*dsh1-sh1)
else if( f.eq.2 ) then
    func1 = w*(1.0-w)*sh2*((1.0-w)*dsh2-sh2)
else if( f.eq.3 ) then
    func1 = w*(1.0-w)*sh3*((1.0-w)*dsh3-sh3)
else if( f.eq.4 ) then
    func1 = w*(1.0-w)*sh3*((1.0-w)*dsh4-sh4)
end if
end if

return

Qgauss2 -- subroutine to perform 10 pt gaussian quadrature numerical integration for arrays with three indices

Local variable definition:

a - lower limit of integration
b - upper limit of integration
ss - current value of integration
pt - gaussian quadrature point
wt - gaussian weight
xm - half of the sum of the limits of integration
xr - half of the difference of the limits of integration
dx - product of xr and the gaussian point
f = flag to specify which shape function to utilize
g = flag to specify which integral to utilize
i - specific nodal position
j - integer counter

subroutine qgauss2( a, b, ss, u, f, g, i )

dimension u(50)
dimension pt(5), wt(5)
real a, b, ss, u
real pt, wt, xm, xr, dx, func2
integer f, i, g
integer j
data pt/-.1488743389,.4333953941,.6794096568,.8650633666,.9739056285/
data wt/-.2955242247,.2692667193,.2190863625,.1494513491,.0666713443/

xm = 0.5 * ( b + a )

xr = 0.5 * ( b - a )
ss = 0.0

C

do 100 j = 1, 5
  dx = xr * pt(j)
  ss = ss + ( wt(j) * ( func2(xm+dx,u,f,g,i) + func2(xm-dx,u,f,g,i)) )
100  continue
ss = xr*ss

C

return
end

Array2 - subroutine to calculate arrays with three indices

Local variable definition:

ss - current value of integration
f = flag to specify which shape function to utilize
g = flag to specify which integral to utilize
i - specific nodal position

subroutine array2( K, g, u, node )

dimension K(50,50), u(50)
real K, u
real ss
integer node
integer f, i, g

C

do 100 i = 1, node
  if( i.eq.1 ) then
    f = 5
    call qgauss2( u(i), u(i+1), ss, u, f, g, i )
    K(i,5) = ss
    f = 6
  else
call qgauss2( u(i), u(i+1), ss, u, f, g, i )
K(i,6) = ss
f = 7
call qgauss2( u(i), u(i+1), ss, u, f, g, i )
K(i,7) = ss
f = 8
call qgauss2( u(i), u(i+1), ss, u, f, g, i )
K(i,8) = ss
else if( i.gt.1.and.i.lt.node ) then
  f = 1
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,1) = ss
f = 2
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,2) = ss
f = 3
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,3) = ss
f = 4
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,4) = ss
f = 5
call qgauss2( u(i), u(i+1), ss, u, f, g, i )
K(i,5) = ss
f = 6
call qgauss2( u(i), u(i+1), ss, u, f, g, i )
K(i,6) = ss
f = 7
call qgauss2( u(i), u(i+1), ss, u, f, g, i )
K(i,7) = ss
f = 8
call qgauss2( u(i), u(i+1), ss, u, f, g, i )
K(i,8) = ss
else if( i.eq.node ) then
  f = 1
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,1) = ss
f = 2
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,2) = ss
f = 3
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,3) = ss
f = 4
call qgauss2( u(i-1), u(i), ss, u, f, g, i )
K(i,4) = ss

end if

100 continue
C
return
end

Func2 - function for arrays with three indices
C
C
Local variable definition:
C
w = current value of u (x-velocity)
f = flag to specify which shape function to utilize
g = flag to specify which integral to utilize
i - specific nodal position
sh1 - shape function N i-1
sh2 - shape function N i-
sh3 - shape function N i+
sh4 - shape function N i+1
dsh1 - derivative of shape function N i-1
dsh2 - derivative of shape function N i-
dsh3 - derivative of shape function N i+
dsh4 - derivative of shape function N i+1
C
real function func2( w, u, f, g, i )
C
dimension u(50)
real w, u
real sh1, sh2, sh3, sh4, dsh1, dsh2, dsh3, dsh4
integer f, i, g
C
if( f.eq.1 ) then
   sh1 = (w-u(i))/(u(i-1)-u(i))
\[ \text{dsh1} = (1.0/(u(i-1)-u(i))) \]
\[ \text{sh2} = (w-u(i-1))/(u(i)-u(i-1)) \]
\[ \text{dsh2} = (1.0/(u(i)-u(i-1))) \]
\[ \text{else if (f.eq.2) then} \]
\[ \text{sh2} = (w-u(i-1))/(u(i)-u(i-1)) \]
\[ \text{dsh2} = (1.0/(u(i)-u(i-1))) \]
\[ \text{else if (f.eq.3) then} \]
\[ \text{sh2} = (w-u(i+1))/(u(i)-u(i+1)) \]
\[ \text{dsh2} = (1.0/(u(i)-u(i+1))) \]
\[ \text{else if (f.eq.4) then} \]
\[ \text{sh2} = (w-u(i+1))/(u(i)-u(i+1)) \]
\[ \text{dsh2} = (1.0/(u(i)-u(i+1))) \]
\[ \text{end if} \]

\[ \text{if (g.eq.8) then} \]
\[ \text{if (f.eq.1) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh2-sh2)*((1.0-w)*dsh1-sh1)*sh1 \]
\[ \text{else if (f.eq.2) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh2-sh2)*((1.0-w)*dsh2-sh2)*sh1 \]
\[ \text{else if (f.eq.3) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh2-sh2)*((1.0-w)*dsh1-sh1)*sh2 \]
\[ \text{else if (f.eq.4) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh2-sh2)*((1.0-w)*dsh2-sh2)*sh2 \]
\[ \text{else if (f.eq.5) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh3-sh3)*((1.0-w)*dsh3-sh3)*sh3 \]
\[ \text{else if (f.eq.6) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh3-sh3)*((1.0-w)*dsh4-sh4)*sh3 \]
\[ \text{else if (f.eq.7) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh3-sh3)*((1.0-w)*dsh3-sh3)*sh4 \]
\[ \text{else if (f.eq.8) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh3-sh3)*((1.0-w)*dsh4-sh4)*sh4 \]
\[ \text{end if} \]

\[ \text{else if (g.eq.9) then} \]
\[ \text{if (f.eq.1) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh2-sh2)*sh1*(((1.0-w)*dsh1-sh1) \]
\[ \text{else if (f.eq.2) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh2-sh2)*sh2*(((1.0-w)*dsh1-sh1) \]
\[ \text{else if (f.eq.3) then} \]
\[ \text{func2} = (1.0-w)*((1.0-w)*dsh2-sh2)*sh1*(((1.0-w)*dsh2-sh2) \]

C
else if  ( f.eq.4 ) then
  func2 = (1.0-w)*((1.0-w)*dsh2-sh2)*sh2*((1.0-w)*dsh2-sh2)
else if  ( f.eq.5 ) then
  func2 = (1.0-w)*((1.0-w)*dsh3-sh3)*sh3*((1.0-w)*dsh3-sh3)
else if  ( f.eq.6 ) then
  func2 = (1.0-w)*((1.0-w)*dsh3-sh3)*sh4*((1.0-w)*dsh3-sh3)
else if  ( f.eq.7 ) then
  func2 = (1.0-w)*((1.0-w)*dsh3-sh3)*sh3*((1.0-w)*dsh4-sh4)
else if  ( f.eq.8 ) then
  func2 = (1.0-w)*((1.0-w)*dsh3-sh3)*sh4*((1.0-w)*dsh4-sh4)
end if

end if

return

end

Qgauss3  - subroutine to perform 10 pt gaussian quadrature numerical integration for arrays with one index

Local variable definition:

a  - lower limit of integration
b  - upper limit of integration
ss  - current value of integration
pt  - gaussian quadrature point
wt  - gaussian weight
xm  - half of the sum of the limits of integration
xr  - half of the difference of the limits of integration
dx  - product of xr and the gaussian point
f  = flag to specify which shape function to utilize
g  = flag to specify which integral to utilize
i  - specific nodal position
j  - integer counter

subroutine qgauss3( a, b, ss, u, f, g, i )
C
dimension u(50)
dimension pt(5), wt(5)
real a, b, ss, u
real pt, wt, xm, xr, dx, func3
integer f, i, g
integer j
data pt/.1488743389,.4333953941,.6794095682,.8650633666,.9739065285/
data wt/.2955242247,.2692667193,.2190863625,.1494513491,.0666713443/
C
xm = 0.5 * ( b + a )
xr = 0.5 * ( b - a )
ss = 0.0
C
do 100 j = 1, 5
   dx = xr * pt(j)
   ss = ss + ( wt(j) * ( func3(xm+dx,u,f,g,i) + func3(xm-dx,u,f,g,i))
100 continue
   ss = xr*ss
C
return
end

Array3 - subroutine to calculate arrays with one index

Local variable definition:

ss - current value of integration
f = flag to specify which shape function to utilize
g = flag to specify which integral to utilize
i - specific nodal position

subroutine array3( K, g, u, node )
dimension K(50), u(50)
real K, u
real ss
integer node
integer f, i, g

C

  do 100 i = 1, node
    if ( i.eq.1 ) then
      f = 3
      call qgauss3( u(i), u(i+1), ss, u, f, g, i )
      K(i) = ss
    else if ( i.gt.1.and.i.lt.node ) then
      f = 2
      call qgauss3( u(i-1), u(i), ss, u, f, g, i )
      K(i) = ss
      f = 3
      call qgauss3( u(i), u(i+1), ss, u, f, g, i )
      K(i) = K(i) + ss
    else if ( i.eq.node ) then
      f = 2
      call qgauss3( u(i-1), u(i), ss, u, f, g, i )
      K(i) = ss
  end if

100 continue
C

return
end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Func3 - function for arrays with one index
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Local variable definition:
C
C   w = current value of u (x-velocity)
C   f = flag to specify which shape function to utilize
C   g = flag to specify which integral to utilize
C   i - specific nodal position
C   sh2 - shape function N i-
C   sh3 - shape function N i+
C
real function func3( w, u, f, g, i )
C
dimension u(50)
real w, u
real sh2, sh3
integer f, i, g
C
if( f.eq.2 ) then
   sh2 = (w-u(i-1))/(u(i)-u(i-1))
else if( f.eq.3 ) then
   sh3 = (w-u(i+1))/(u(i)-u(i+1))
end if
C
if( g.eq. 1 ) then
if( f.eq.2 ) then
   func3 = sh2
else if( f.eq.3 ) then
   func3 = sh3
end if
C
else if( g.eq.2 ) then
if( f.eq.2 ) then
   func3 = w*sh2
else if( f.eq.3 ) then
   func3 = w*sh3
end if
C
else if( g.eq.3 ) then
if( f.eq.2 ) then
   func3 = sh2
else if( f.eq.3 ) then
   func3 = sh3
end if
C
else if( g.eq.4 ) then
if( f.eq.2 ) then
   func3 = (1.0/(1.0-w))*sh2
else if( f.eq.3 ) then
   func3 = (1.0/(1.0-w))*sh3
end if
end if

return

AssembleKT - subroutine to assemble stiffness matrix KT

Local variable definition:

dum1-dum3 - dummy variables for interim calculations

subroutine assembleKT( KT, K1, K2, K3, c1, c2, b1, b2, b5, db5db1,
                       theta, delx, node )

dimension KT(50,50), K1(50,4), K2(50,4), K3(50,4)
dimension c1(2), c2(2), b1(50), b2(50), b5(50), db5db1(50)
real KT, K1, K2, K3, c1, c2, b1, b2, b5, db5db1, theta
real delx, dum1, dum2, dum3
integer node
integer i

i = 1
dum2 = c1(2)*K3(i,3) - c2(2)*K2(i,3)
  * (b5(i)-(b2(i)**2.0)-b2(i)*db5db1(i))
dum3 = c1(2)*K3(i,4) - c2(2)*K2(i,4)
  * (b5(i+1)-(b2(i+1)**2.0)-b2(i+1)*db5db1(i+1))
KT(i,i) = K1(i,3) - (theta*delx)*dum2
KT(i,i+1) = K1(i,4) - (theta*delx)*dum3

do 100 i = 2, node-1
  dum1 = c1(2)*K3(i,1) - c2(2)*K2(i,1)
        * (b5(i-1)-(b2(i-1)**2.0)-b2(i-1)*db5db1(i-1))
dum2 = c1(2)*K3(i,3) - c2(2)*K2(i,3)
        * (b5(i)-(b2(i)**2.0)-b2(i)*db5db1(i))
dum3 = c1(2)*K3(i,4) - c2(2)*K2(i,4)
1          * (b5(i+1)*(b2(i+1)**2.0)-b2(i+1)*db5db1(i+1))
KT(i,i-1) = K1(i,1) - (theta*delx)*dum1
KT(i,i)  = K1(i,2) + K1(i,3) - (theta*delx)*dum2
KT(i,i+1) = K1(i,4) - (theta*delx)*dum3
100       continue
C
i = node
dum1 = c1(2)*K3(i,1) - c2(2)*K2(i,1)
1          * (b5(i-1)*(b2(i-1)**2.0)-b2(i-1)*db5db1(i-1))
dum2 = c1(2)*K3(i,3) - c2(2)*K2(i,3)
1          * (b5(i)*(b2(i)**2.0)-b2(i)*db5db1(i))
KT(i,i-1) = K1(i,1) - (theta*delx)*dum1
KT(i,i)  = K1(i,2) - (theta*delx)*dum2
C
return
end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  AssembleFT - subroutine to assemble load vector FT
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  Local variable definition:
C
C  con1-con2 - dummy variables for interim calculations
C  dum1-dum6 - dummy variables for interim calculations
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  subroutine assembleFT( FT, K2, K3, c1, c2, b1, b6, theta, delx, node )
C
dimension FT(50), K2(50,4), K3(50,4), c1(2), c2(2), b1(50), b6(50)
real FT, K2, K3, c1, c2, b1, b6, theta, delx
real dum1, dum2, dum3, dum4, dum5, dum6, con1, con2
integer node
integer i

C
con1 = (theta*c1(2))+((1.0-theta)*c1(1))
con2 = (theta*c2(2))+((1.0-theta)*c2(1))
C
i = 1
dum2 = K2(i,3)*b6(i)
dum3 = K2(i,4)*b6(i+1)
dum5 = K3(i,3)*b1(i)
dum6 = K3(i,4)*b1(i+1)

FT(i) = delx*((con2*(dum2+dum3))+(con1*(dum5+dum6)))

C
do 100 i = 2, node-1
    dum1 = K2(i,1)*b6(i-1)
    dum2 = (K2(i,2)+K2(i,3))*b6(i)
    dum3 = K2(i,4)*b6(i+1)
    dum4 = K3(i,1)*b1(i-1)
    dum5 = (K3(i,2)+K3(i,3))*b1(i)
    dum6 = K3(i,4)*b1(i+1)
    FT(i) = delx*((con2*(dum1+dum2+dum3))+(con1*(dum4+dum5+dum6)))

100 continue
C
i = node
    dum1 = K2(i,1)*b6(i-1)
    dum2 = K2(i,2)*b6(i)
    dum4 = K3(i,1)*b1(i-1)
    dum5 = K3(i,2)*b1(i)
    FT(i) = delx*((con2*(dum1+dum2))+(con1*(dum4+dum5)))
C
return
end

dimension b2(50), b6(50), c1(2), c2(2), c3(2)
real KS, K4, K5, K8, K9, b2, b6
real c1, c2, c3, delx, theta
real dum1, dum2, dum3, dum4, dum5, dum6, dum7, dum8
real dum9, dum10, dum11, dum12, dum13, dum14, dum15
real dum16, dum17, dum18, dum19, dum20
integer i, node

C

i = 1
dum2 = K4(i,3)
dum3 = K4(i,4)
dum5 = K5(i,3)
dum6 = K5(i,4)
dum10 = K8(i,5)*b6(i)*b2(i)
dum11 = K8(i,6)*b6(i+1)*b2(i)
dum12 = K8(i,7)*b6(i)*b2(i+1)
dum13 = K8(i,8)*b6(i+1)*b2(i+1)
dum17 = K9(i,5)*b6(i)*b2(i)
dum18 = K9(i,6)*b6(i+1)*b2(i)
dum19 = K9(i,7)*b6(i)*b2(i+1)
dum20 = K9(i,8)*b6(i+1)*b2(i+1)
KS(i,1) = 1.0e8
KS(i,i+1) = K4(i,4) - theta*delx*(-2.0*c1(2)*dum3
1 + c1(2)*dum6 + c2(2)*c2(2)*(dum11+dum12+dum13)
2 - c3(2)*(dum18+dum19+dum20))

C

do 100 i = 2, node-1
dum1 = K4(i,1)
dum2 = (K4(i,2)+K4(i,3))
dum3 = K4(i,4)
dum4 = K5(i,1)
dum5 = (K5(i,2)+K5(i,3))
dum6 = K5(i,4)
dum7 = K8(i,1)*b6(i-1)*b2(i-1)
dum8 = K8(i,2)*b6(i)*b2(i-1)
dum9 = K8(i,3)*b6(i-1)*b2(i)
dum10 = (K8(i,4)+K8(i,5))*b6(i)*b2(i)
dum11 = K8(i,6)*b6(i+1)*b2(i)
dum12 = K8(i,7)*b6(i)*b2(i+1)
dum13 = K8(i,8)*b6(i+1)*b2(i+1)
dum14 = K9(i,1)*b6(i-1)*b2(i-1)
dum15 = K9(i,2)*b6(i)*b2(i-1)
dum16 = K9(i,3)*b6(i-1)*b2(i)
dum17 = (K9(i,4)+K9(i,5))*b6(i)*b2(i)
dum18 = K9(i,6)*b6(i+1)*b2(i)
dum19 = K9(i,7)*b6(i)*b2(i+1)
dum20 = K9(i,8)*b6(i+1)*b2(i+1)
KS(i,i-1) = K4(i,1) - theta*delx*(-2.0*c1(2)*dum1
   + c1(2)*dum4 + c2(2)*(dum7+dum8+dum9)
   - c3(2)*(dum14+dum15+dum16))
KS(i,i) = K4(i,2) + K4(i,3) - theta*delx*(-2.0*c1(2)*dum2
   + c1(2)*dum5 + c2(2)*dum10 - c3(2)*dum17)
KS(i,i+1) = K4(i,4) - theta*delx*(-2.0*c1(2)*dum3
   + c1(2)*dum6 + c2(2)*(dum11+dum12+dum13)
   - c3(2)*(dum18+dum19+dum20))

continue

i = node

i = node

dum1 = K4(i,1)
dum2 = K4(i,2)
dum4 = K5(i,1)
dum5 = K6(i,2)
dum7 = K8(i,1)*b6(i-1)*b2(i-1)
dum8 = K8(i,2)*b6(i)*b2(i-1)
dum9 = K8(i,3)*b6(i-1)*b2(i)
dum10 = K8(i,4)*b6(i)*b2(i)
dum14 = K9(i,1)*b6(i-1)*b2(i-1)
dum15 = K9(i,2)*b6(i)*b2(i-1)
dum16 = K9(i,3)*b6(i-1)*b2(i)
dum17 = K9(i,4)*b6(i)*b2(i)
KS(i,i-1) = K4(i,1) - theta*delx*(-2.0*c1(2)*dum1
   + c1(2)*dum4 + c2(2)*(dum7+dum8+dum9)
   - c3(2)*(dum14+dum15+dum16))
KS(i,i) = K4(i,2) - theta*delx*(-2.0*c1(2)*dum2
   + c1(2)*dum5 + c2(2)*dum10 - c3(2)*dum17)

return
end

C

AssembleFS - subroutine to assemble load vector FS
C
Local variable definition:

con1-con4 - dummy variables for interim calculations
dum1-dum26 - dummy variables for interim calculations

subroutine assembleFS( TT, FS, K4, K5, K6, K7, K8, K9, b3, b4, b6,
                    c1, c2, c3, c4, theta, delx, sw, node )

dimension FS(50), K4(50,4), K5(50,4), K6(50,4), K7(50,4), K8(50,8)
dimension K9(50,8), TT(50), b3(50), b4(50), b6(50), c1(2), c2(2)
dimension c3(2), c4(2)
real FS, K4, K5, K6, K7, K8, K9, TT, b3, b4, b6
real c1, c2, c3, c4, delx, theta, sw
real dum1, dum2, dum3, dum4, dum5, dum6, dum7, dum8
real dum9, dum10, dum11, dum12, dum13, dum14, dum15
real dum16, dum17, dum18, dum19, dum20, dum21, dum22
real dum23, dum24, dum25, dum26, con1, con2, con3, con4
integer i, node

con1 = (theta*c1(2))+((1.0-theta)*c1(1))
con2 = (theta*c2(2))+((1.0-theta)*c2(1))
con3 = (theta*c3(2))+((1.0-theta)*c3(1))
con4 = (theta*c4(2))+((1.0-theta)*c4(1))

i = 1
dum2 = K4(i,3)*b4(i)
dum3 = K4(i,4)*b4(i+1)
dum5 = K5(i,3)*b4(i)
dum6 = K5(i,4)*b4(i+1)
dum10 = K8(i,5)*b6(i)*b3(i)
dum11 = K8(i,6)*b6(i+1)*b3(i)
dum12 = K8(i,7)*b6(i)*b3(i+1)
dum13 = K8(i,8)*b6(i+1)*b3(i+1)
dum17 = K9(i,5)*b6(i)*b3(i)
dum18 = K9(i,6)*b6(i+1)*b3(i)
dum19 = K9(i,7)*b6(i)*b3(i+1)
dum20 = K9(i,8)*b6(i+1)*b3(i+1)
dum22 = K6(i,3)*b6(i)
dum23 = K6(i,4)*b6(i+1)
dum25 = K7(i,3)*b6(i)
dum26 = K7(i,4)*b6(i+1)
FS(i) = 1.0e8*sw*TT(i)
C
   do 100 i = 2, node-1
      dum1 = K4(i,1)*b4(i-1)
      dum2 = (K4(i,2)+K4(i,3))*b4(i)
      dum3 = K4(i,4)*b4(i+1)
      dum4 = K5(i,1)*b4(i-1)
      dum5 = (K5(i,2)+K5(i,3))*b4(i)
      dum6 = K5(i,4)*b4(i+1)
      dum7 = K8(i,1)*b6(i-1)*b3(i-1)
      dum8 = K9(i,2)*b6(i)*b3(i-1)
      dum9 = K8(i,3)*b6(i-1)*b3(i)
      dum10 = (K8(i,4)+K8(i,5))*b6(i)*b3(i)
      dum11 = K8(i,6)*b6(i+1)*b3(i)
      dum12 = K8(i,7)*b6(i)*b3(i+1)
      dum13 = K8(i,8)*b6(i+1)*b3(i+1)
      dum14 = K9(i,1)*b6(i-1)*b3(i-1)
      dum15 = K9(i,2)*b6(i)*b3(i-1)
      dum16 = K9(i,3)*b6(i-1)*b3(i)
      dum17 = (K9(i,4)+K9(i,5))*b6(i)*b3(i)
      dum18 = K9(i,6)*b6(i+1)*b3(i)
      dum19 = K9(i,7)*b6(i)*b3(i+1)
      dum20 = K9(i,8)*b6(i+1)*b3(i+1)
      dum21 = K6(i,1)*b6(i-1)
      dum22 = (K6(i,2)+K6(i,3))*b6(i)
      dum23 = K6(i,4)*b6(i+1)
      dum24 = K7(i,1)*b6(i-1)
      dum25 = (K7(i,2)+K7(i,3))*b6(i)
      dum26 = K7(i,4)*b6(i+1)
      FS(i) = (-2.0*con1*(dum1+dum2+dum3) + con1*(dum4+dum5+dum6)
      + con2*(dum7+dum8+dum9+dum10+dum11+dum12+dum13)
      - con3*(dum14+dum15+dum16+dum17+dum18+dum19+dum20)
      + con4*(dum21+dum22+dum23) + con4*(dum24+dum25+dum26))
   100 continue
C
   i = node
   dum1 = K4(i,1)*b4(i-1)
   dum2 = K4(i,2)*b4(i)
   dum4 = K5(i,1)*b4(i-1)
dum5 = K5(i,2)*b4(i)
dum7 = K8(i,1)*b6(i-1)*b3(i-1)
dum8 = K8(i,2)*b6(i)*b3(i-1)
dum9 = K8(i,3)*b6(i-1)*b3(i)
dum10 = K8(i,4)*b6(i)*b3(i)
dum14 = K9(i,1)*b6(i-1)*b3(i-1)
dum15 = K9(i,2)*b6(i)*b3(i-1)
dum16 = K9(i,3)*b6(i-1)*b3(i)
dum17 = K9(i,4)*b6(i)*b3(i)
dum21 = K6(i,1)*b6(i-1)
dum22 = K6(i,2)*b6(i)
dum24 = K7(i,1)*b6(i-1)
dum25 = K7(i,2)*b6(i)

FS(i) = (-2.0*con1*(dum1+dum2) + con1*(dum4+dum5)
1 + con2*(dum7+dum8+dum9+dum10)
2 - con3*(dum14+dum15+dum16+dum17)
3 + con4*(dum21+dum22) + con4*(dum24+dum25))
4 * delx

return
end

***************
Calcdbdidxs - subroutine to calculate the derivative of b1 w/r/t xs
Local variable definition:

doload - load vector
dbstiff - stiffness matrix
dum1-dum26 - dummy variables for interim calculations

subroutine calcdbdidxs(K1, K2, K3, c1, c2, b1, b6, dbidxs, node )

dimension K1(60,4), K2(60,4), K3(50,4), dbstiff(50,50)
dimension b1(50), b6(50), dbidxs(50), dbload(50), c1(2), c2(2)
real K1, K2, K3, c1, c2, b1, b6, dbidxs, dbload
real dbstiff, dum1, dum2, dum3, dum4, dum5, dum6
real con1, con2
integer node, i

C
\ncon1 = c1(i)
con2 = c2(i)

C
i = 1
dum2 = K2(i,3)\*b6(i)
dum3 = K2(i,4)\*b6(i+1)
dum5 = K3(i,3)\*b1(i)
dum6 = K3(i,4)\*b1(i+1)
dbload(i) = (con2*(dum2+dum3))+(con1*(dum5+dum6))
dbstiff(i,i) = K1(i,3)
dbstiff(i,i+1) = K1(i,4)

C

do 100 i = 2, node-1
   dum1 = K2(i,1)\*b6(i-1)
dum2 = (K2(i,2)+K2(i,3))\*b6(i)
dum3 = K2(i,4)\*b6(i+1)
dum4 = K3(i,1)\*b1(i-1)
dum5 = (K3(i,2)+K3(i,3))\*b1(i)
dum6 = K3(i,4)\*b1(i+1)
dbload(i) = (con2*(dum1+dum2+dum3))+(con1*(dum4+dum5+dum6))
dbstiff(i,i-1) = K1(i,1)
dbstiff(i,i) = K1(i,2) + K1(i,3)
dbstiff(i,i+1) = K1(i,4)
100 continue

C
i = node
   dum1 = K2(i,1)\*b6(i-1)
dum2 = K2(i,2)\*b6(i)
dum4 = K3(i,1)\*b1(i-1)
dum5 = K3(i,2)\*b1(i)
dbload(i) = (con2*(dum1+dum2))+(con1*(dum4+dum5))
dbstiff(i,i-1) = K1(i,1)
dbstiff(i,i) = K1(i,2)

C
call solve( dbstiff, dbload, dbidxs, node )

C
return
end
Calculdb4dxs - subroutine to calculate derivative of b4 w/r/t xs

Local variable definition:

dbload - load vector
dbstiff - stiffness matrix
dum1-dum26 - dummy variables for interim calculations

subroutine calculdb4dxs( K4, K5, K6, K7, K8, K9, b3, b4, b6, c1, c2,
c3, c4, db1dxs, db4dxs, sw, node )

dimension K4(50,4), K5(50,4), K6(50,4), K7(50,4), K8(50,4), K9(50,4),
dimension b3(50), b4(50), b6(50), c1(2), c2(2),
dimension c3(2), c4(2), db1dxs(50), db4dxs(50), dbload(50),
dimension dbstiff(50,50)
real K4, K5, K6, K7, K8, K9, b3, b4, b6
real c1, c2, c3, c4, db1dxs, db4dxs, dbload
real dbstiff, sw
real dum1, dum2, dum3, dum4, dum5, dum6, dum7, dum8
real dum9, dum10, dum11, dum12, dum13, dum14, dum15
real dum16, dum17, dum18, dum19, dum20, dum21, dum22
real dum23, dum24, dum25, dum26
integer i, node

i = 1
dum2 = K4(i,3)*b4(i)
dum3 = K4(i,4)*b4(i+1)
dum5 = K5(i,3)*b4(i)
dum6 = K5(i,4)*b4(i+1)
dum10 = K8(i,5)*b6(i)*b3(i)
dum11 = K8(i,6)*b6(i+1)*b3(i)
dum12 = K8(i,7)*b6(i)*b3(i+1)
dum13 = K8(i,8)*b6(i+1)*b3(i+1)
dum17 = K9(i,5)*b6(i)*b3(i)
dum18 = K9(i,6)*b6(i+1)*b3(i)
dum19 = K9(i,7)*b6(i)*b3(i+1)
dum20 = K9(i,8)*b6(i+1)*b3(i+1)
dum22 = K9(i,3)*b6(i)
dum23 = K9(i,4)*b6(i+1)
dum25 = K7(i,3)*b6(i)
dum26 = K7(i,4)*b6(i+1)
dbload(i) = 1.0e8*sw*db1dxs(i)
dbstiff(i,i) = 1.0e8
dbstiff(i,i+1) = K4(i,4)

C

do 100 i = 2, node-1
    dum1 = K4(i,1)*b4(i-1)
dum2 = (K4(i,2)+K4(i,3))*b4(i)
dum3 = K4(i,4)*b4(i+1)
dum4 = K5(i,1)*b4(i-1)
dum5 = (K5(i,2)+K5(i,3))*b4(i)
dum6 = K5(i,4)*b4(i+1)
dum7 = K8(i,1)*b6(i-1)*b3(i-1)
dum8 = K8(i,2)*b6(i)*b3(i-1)
dum9 = K8(i,3)*b6(i-1)*b3(i)
dum10 = (K8(i,4)+K8(i,5))*b6(i)*b3(i)
dum11 = K8(i,6)*b6(i+1)*b3(i)
dum12 = K8(i,7)*b6(i)*b3(i+1)
dum13 = K8(i,8)*b6(i+1)*b3(i+1)
dum14 = K9(i,1)*b6(i-1)*b3(i-1)
dum15 = K9(i,2)*b6(i)*b3(i-1)
dum16 = K9(i,3)*b6(i-1)*b3(i)
dum17 = (K9(i,4)+K9(i,5))*b6(i)*b3(i)
dum18 = K9(i,6)*b6(i+1)*b3(i)
dum19 = K9(i,7)*b6(i)*b3(i+1)
dum20 = K9(i,8)*b6(i+1)*b3(i+1)
dum21 = K6(i,1)*b6(i-1)
dum22 = (K6(i,2)+K6(i,3))*b6(i)
dum23 = K6(i,4)*b6(i+1)
dum24 = K7(i,1)*b6(i-1)
dum25 = (K7(i,2)+K7(i,3))*b6(i)
dum26 = K7(i,4)*b6(i+1)

dbload(i) = -2.0*c1(1)*(dum1+dum2+dum3) + c1(1)*(dum4+dum5+dum6) + c2(1)*(dum7+dum8+dum9+dum10+dum11+dum12+dum13) - c3(1)*(dum14+dum15+dum16+dum17+dum18+dum19+dum20) + c4(1)*(dum21+dum22+dum23) + c4(1)*(dum24+dum25+dum26)
\[ \text{dbstiff}(i,i-1) = K4(i,1) \]
\[ \text{dbstiff}(i,i) = K4(i,2) + K4(i,3) \]
\[ \text{dbstiff}(i,i+1) = K4(i,4) \]

\text{C}

\begin{verbatim}
100 continue

i = node

dum1 = K4(i,1)*b4(i-1)
dum2 = K4(i,2)*b4(i)
dum4 = K5(i,1)*b4(i-1)
dum5 = K5(i,2)*b4(i)
dum7 = K8(i,1)*b6(i-1)*b3(i-1)
dum8 = K8(i,2)*b6(i)*b3(i-1)
dum9 = K8(i,3)*b6(i-1)*b3(i)
dum10 = K8(i,4)*b6(i)*b3(i)
dum14 = K9(i,1)*b6(i-1)*b3(i-1)
dum15 = K9(i,2)*b6(i)*b3(i-1)
dum16 = K9(i,3)*b6(i-1)*b3(i)
dum17 = K9(i,4)*b6(i)*b3(i)
dum21 = K6(i,1)*b6(i-1)
dum22 = K6(i,2)*b6(i)
dum24 = K7(i,1)*b6(i-1)
dum25 = K7(i,2)*b6(i)

dbload(i) = -2.0*c1(i)*(dum1+dum2) + c1(i)*(dum4+dum5) + c2(i)*(dum8+dum9+dum10) - c3(i)*(dum14+dum15+dum16+dum17) + c4(i)*(dum21+dum22) + c4(i)*(dum24+dum25)

\text{dbstiff}(i,i-1) = K4(i,1)
\text{dbstiff}(i,i) = K4(i,2)
\end{verbatim}

\text{C}

\text{call solve( dbstiff, dbload, dbdxs, node )}

\text{C}

\text{return}
\text{end}

\text{C}
\text{profile - subroutine to calculate a velocity profile at a specific}
\text{C}
\text{xc location}

\text{C}
\text{Local variable definition:}
rho - nodal value of density
s - entropy variable
ydm - dummy variable for interim calculation

subroutine profile( D4, b2, b3, u, uinf, gamma, re, ue, pe, node )
dimension D4(50), b2(50), b3(50), u(50), s(50), rho(50), y(50)
real D4, b2, b3, u, s, rho, y
real gamma, re, uinf, ue, pe, dum1, dum2, ydm
integer node, i

do 100 i = 1, node
   s(i) = (1.0-u(i))*b3(i)
   dum1 = ((2.0*pe*gamma)/(gamma+1.0))
   dum2 = 1.0/(1.0-(s(i))*((gamma-1.0)/(gamma+1.0))
   * (u(i)**2.0)*(ue**2.0))
   rho(i) = dum1*dum2
100 continue

  u(node) = 0.99
  y(1) = 0.0
  call array3( D4, 4, u, node )
ydm = 0.0
  do 200 i = 2, node
     ydum = ydum + D4(i)**(1.0/(b2(i)*rho(i)))
     y(i) = (uinf/(sqrt(re)*ue))*ydum
 200 continue

  u(node) = 1.00

write(*,300) ( y(i), u(i), i = 1, node )
300 format( 2f15.6 )
return
end

Displ - subroutine to calculate the displacement thickness
Local variable definition:

sum1-sum3 - dummy variables for interim calculations
fac1-fac3 - coefficients for displacement thickness calculation

subroutine displ(D1, D2, D3, b1, b4, dstick, uinf, pinf, gamma,
                re, ue, pe, node)

dimension D1(50), D2(50), D3(50), b1(50), b4(50)
real D1, D2, D3, b1, b4
real gamma, re, uinf, pinf, ue, pe
real sum1, sum2, sum3, fac1, fac2, fac3, dstick
integer node, i

sum1 = 0.0
sum2 = 0.0
sum3 = 0.0
do 100 i = 1, node
    sum1 = sum1 + D1(i)*b1(i)
    sum2 = sum2 + D2(i)*b1(i)
    sum3 = sum3 + D3(i)*b4(i)
100       continue
fac1 = ((gamma+1.0)/(2.0*gamma))*((uinf)/(ue*pe*sqrt(re))
fac2 = ((gamma-1.0)/(2.0*gamma))*((uinf*ue)/(pe*sqrt(re))
fac3 = ((gamma+1.0)/(2.0*gamma))*((uinf*ue)/(pe*sqrt(re))
dstick = fac1*sum1 + fac2*sum2 - fac3*sum3

return
end

Momem - subroutine to calculate the momentum thickness

Local variable definition:

sum1 - dummy variables for interim calculations
fac1 - coefficient for momentum thickness calculation

subroutine momen( D2, b1, mothick, uinf, ue, rhoe, re, node )

dimension D2(50), b1(50)
real D2, b1
real uinf, ue, rhoe, re
real sum1, fac1, mothick
integer node, i

sum1 = 0.0
do 100 i = 1, node
    sum1 = sum1 + D2(i)*b1(i)
 100 continue
fac1 = uinf/(sqrt(re)*rhoe*ue)
mothick = fac1*sum1

return
end

Bound - subroutine to calculate the boundary layer thickness

Local variable definition:
rho - nodal value of density
s - entropy variable
ydum - dummy variable for interim calculation

subroutine bound( D4, b2, b3, blthick, u, uinf, gamma, re, ue, pe, 
                  node )

dimension D4(50), b2(50), b3(50), u(50), s(50), rho(50)
real D4, b2, b3, u, s, rho, blthick
real gamma, re, uinf, ue, pe, dum1, dum2, ydum
integer node, i

C
do 100 i = 1, node
   s(i) = (1.0-u(i))*b3(i)
   dum1 = ((2.0*pe*gamma)/(gamma+1.0))
   dum2 = 1.0/(1.0-(s(i)+(gamma-1.0)/(gamma+1.0))
         * (u(i)**2.0)*(ue**2.0))
   rho(i) = dum1*dum2
100 continue
u(node) = 0.99
call array3( D4, 4, u, node )
ydum = 0.0
do 200 i = 2, node
   ydum = ydum + D4(i)*(1.0/(b2(i)*rho(i)))
200 continue
u(node) = 1.00
blthick = (uinfl/(sqrt(re)*ue))*ydum
C
return
end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    Trans - subroutine to calculate the transpiration velocity
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    Local variable definition:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    sum1-sum6 - dummy variables for interim calculations
C    fac1-fac3 - coefficients for transpiration velocity calculation
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    subroutine trans( D1, D2, D3, b1, dbdxs, b4, db4dxs, transvel, uinf, 1
               gamma, re, ue, due, pe, rhoe, drhoe, node )
C
dimension D1(50), D2(50), D3(50), b1(50), dbdxs(50), b4(50)
dimension db4dxs(50)
real D1, D2, D3, b1, dbdxs, b4, db4dxs
real gamma, re, uinf, rhoe, drhoe, ue, due, pe
real sum1, sum2, sum3, fac1, fac2, fac3
real sum4, sum5, sum6, transvel
integer node, i

C

sum1 = 0.0
sum2 = 0.0
sum3 = 0.0
sum4 = 0.0
sum5 = 0.0
sum6 = 0.0
do 100 i = 1, node
   sum1 = sum1 + D1(i)*b1(i)
   sum2 = sum2 + D2(i)*b1(i)
   sum3 = sum3 + D3(i)*b4(i)
   sum4 = sum4 + D1(i)*dbidxs(i)
   sum5 = sum5 + D2(i)*dbidxs(i)
   sum6 = sum6 + D3(i)*db4dxs(i)
100 continue
fac1 = ((gamma+1.0)/(2.0*gamma))*(uinf/(ue*pe*sqrt(re)))
fac2 = ((gamma-1.0)/(2.0*gamma))*((uinf*ue)/(pe*sqrt(re)))
fac3 = ((gamma+1.0)/(2.0*gamma))*((uinf*ue)/(pe*sqrt(re)))
transvel = due*(fac1*sum1 + fac2*sum2 - fac3*sum3)
           + ue*(fac1*sum4 + fac2*sum5 - fac3*sum6)
           + (ue/rhoe)*(drhoe)*(fac1*sum1 + fac2*sum2 - fac3*sum3)
C
return
end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C Similar - implements the fourth order Runge-Kutta method to
C solve Falkner-Skan similarity equation using
C bisection for the shooting method.
C
C f''(alpha) + beta (1-f'(alpha)^2) + (1/(m+1)) f(alpha) f''(alpha)
C = 0
C
C and
C
C g''(alpha) + (pr/(m+1)) f(alpha) g'(alpha)
C - 2 pr beta f'(alpha) g(alpha)
C + (2(gamma-1)/(gamma+1))(1-pr)(f''(alpha)^2
\[ f'(0) = 0 \]
\[ f''(0) = 1 \]
\[ f'''(0) = \text{determined from the shooting method} \]
\[ g(0) = \text{equal to } (1 - twto)(1/u^2) \text{ for heat transfer boundary condition} \]
\[ \text{or determined from the shooting method for adiabatic wall} \]
\[ g(10) = 0 \]
\[ g'(0) = \text{determined from the shooting method for heat transfer} \]
\[ \text{boundary condition or equal to zero for adiabatic wall} \]

The equation can be written in the following form

\[ z(1) = f \]
\[ z(2) = f' \]
\[ z(3) = f'' \]
\[ z(4) = g \]
\[ z(5) = g' \]

which results in

\[ z'(1) = f' = z(2) \]
\[ z'(2) = f'' = z(3) \]
\[ z'(3) = f''' = -(beta*(1-z(2)^2) + (1/(m+1)) z(1) z(3)) \]
\[ z'(4) = g' = z(5) \]
\[ z'(5) = g'' = -(pr/(m+1)) z(1) z(5) + 2 pr beta z(2) z(4) \]
\[ - (2(gamma-1)/(gamma+1))(1-pr)(z(3)^2 + z(2)z(3))' \]

Using the following 4th order Runge-Kutta method:

\[ z(i+1) = z(i) + h \phi(z(i), h) \]

where

\[ \phi = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \]

and

\[ k_1 = f(z) \]
\[ k_2 = f(z + 0.5h \cdot k_1) \]
\[ k_3 = f(z + 0.5h \cdot k_2) \]
\[ k_4 = f(z + h \cdot k_3) \]
Local variable definition:

alpha - independent variable at i step
z - dependent variable at i step
znext - dependent variable at i+1 step
zstart - boundary conditions at alpha = alphastart
zguess - upper and lower bounds of unknown boundary conditions
at alpha = alphastart
zmstart - current guess for unknown momentum boundary condition
at alpha = alphastart
zestart - current guess for unknown energy boundary condition
at alpha = alphastart
h - step width, h=alphanext-alpha
zend - boundary conditions at alpha = alphaend
alphastart - starting value for independent variable
alphaend - ending value for independent variable
phi - runge-kutta interpolating function
beta - parameter describing pressure
m - parameter describing pressure
pr - Prandtl number
test - relative error at alpha = alphaend
ebctype - type of energy boundary condition (0=adiabatic, 1=heat)
eps - error tolerance
i - step location
p - print counter
pstep - print step
j - dummy counter
node - number of nodes for output f', f'', g

subroutine similar( b1, b2, b3, b4, c0, u, uinf, pinf, ue, pe, 
xstep, pr, gamma, twto, node, inum )

dimension b1(50), b2(50), b3(50), b4(50), c0(2), u(50), f2(50), g0(50)
dimension z(5), znext(5), phi(5), zstart(5), zguess(4), zend(5)
real b1, b2, b3, b4, c0, u, uinf, pinf, ue, pe, xstep
real alpha, z, znext, h, alphastart, zstart, alphaend
real beta, m, zguess, zmstart, zestart, eps, test, phi
real pr, gamma, zend, dum1, f2, g0, twto
integer node, i, ectype, inum

C

if( inum.gt.1 ) then
    read(24,*) ( b1(i), b2(i), b3(i), b4(i), i = 1, node )
    return
end if

call siminput( alphastart, zstart, zguess, zend, alphasend,
1        eps, h, beta, m, twto, ue, ectype )
call simomentum( z, znnext, zstart, zguess, zend, phi, alpha,
1        alphastart, alphasend, h, beta, m, zstart, eps,
2        test )
    zstart(3) = zmstart

call simenergy( z, znnext, zstart, zguess, zend, phi, alpha,
1        alphastart, alphasend, h, beta, m, zstart, eps,
2        test, pr, gamma, ectype )

if( ectype.eq.0 ) then
    zstart(4) = zrestart
end if

if( ectype.eq.1 ) then
    zstart(4) = zrestart
end if

call simfinal( z, znnext, zstart, phi, alpha, alphastart,
1        alphasend, h, beta, m, pr, gamma, f2, go, node )

C
dum1 = sqrt((uinuf*pinf)/(c0(1)*abs(ue)*ps*xstep))
do 100 i = 1, node-1
    b1(i) = (1.0-u(i))/(dum1*f2(i))
    b2(i) = 1.0/b1(i)
    b3(i) = go(i)/(1.0-u(i))
    b4(i) = b1(i)*b3(i)
100 continue

b1(node) = b1(node-1) + (b1(node-1)-b1(node-2))
b2(node) = b2(node-1) + (b2(node-1)-b2(node-2))
b3(node) = b3(node-1) + (b3(node-1)-b3(node-2))
b4(node) = b4(node-1) + (b4(node-1)-b4(node-2))
if( inum.le.1 ) then
    write(24,200) ( b1(i), b2(i), b3(i), b4(i), i = 1, node )
200 format( 4e15.6 )
end if

C

return
end
Siminput - subroutine to get input from siminput.dat

Local variable definition:

see global variable definitions

j - dummy counter

subroutine siminput( alphastart, zstart, zguess, zend, alphaend, eps, h, beta, m, twto, ue, ebctype )

dimension zstart(5), zguess(4), zend(5)
real alphastart, zstart, zguess, alphaend, h, beta, m
real eps, zend, twto, ue
integer j, ebctype

read(17,*) alphastart, alphaend, h, eps
read(17,*) ( zguess(j), j = 1, 4 )
read(17,*) ebctype
read(17,*) twto
read(17,*) m

beta = (2.0*m)/(m + 1.0)
zstart(1) = 0.0
zstart(2) = 0.0

for adiabatic wall condition

if( ebctype.eq.0 ) then
    zstart(5) = 0.0
end if

for heat transfer wall condition

if( ebctype.eq.1 ) then
    zstart(4) = (1.0 - twto)*(1.0/ue**2.0)
end if
zend(1) = 1.0
zend(2) = 0.0

C
return
end

C
Simommentum - subroutine to control solution of momentum similarity
C

C
Local variable definition:
C
see global variable definitions
C
j       - dummy counter
C

C

C
 subroutine simomentum( z, znext, zstart, zguess, zend, phi, alpha,
1      alphastart, alphaend, h, beta, m, zmstart, 
2      eps, test )
C

C
dimension z(5), znext(5), phi(5), zstart(5), zguess(4), zend(2)
real alpha, z, znext, h, alphastart, zstart, alphaend
real beta, m, zguess, zmstart, eps, test, zend, phi
integer j

C
continue
alpha = alphastart
do 100 j = 1, 2
    z(j) = zstart(j)
100 continue
zmstart = 0.5*(zguess(1)+zguess(2))
z(3) = zmstart

C
continue
    call rungem( z, znext, h, phi, beta, m )
    alpha = alpha + h
    do 200 j = 1, 3
        z(j) = znext(j)
200 continue
200          continue
C                   
if( alpha.lt.alphaend ) goto 99
  test = abs(zend(1)-z(2))*100.0
  if( test.gt.eps ) then
    if( z(2).gt.zend(1) ) then
      zguess(2) = zmstart
    else if( z(2).lt.zend(1) ) then
      zguess(1) = zmstart
    end if
  goto 9
end if
C                   
return
end

Simenergy - subroutine to control solution of energy similarity
C
Local variable definition:
C
see global variable definitions
C
j      - dummy counter
C

subroutine simenergy(z, znext, zstart, zguess, zend, phi, alpha,
  alphastart, alphaend, h, beta, m, zstart, eps,
  test, pr, gamma, ebtype )

dimension z(5), znext(5), phi(5), zstart(5), zguess(4), zend(2)
real alpha, z, znext, h, alphastart, zstart, alphaend, phi
real beta, m, zguess, zstart, eps, test, zend
real pr, gamma
integer j, ebtype
C
9          continue
alpha = alphastart
do 100 j = 1, 5
z(j) = zstart(j)
100 continue
zstart = 0.5*(zguess(3)+zguess(4))
if( ebctype.eq.0 ) then
  z(4) = zstart
end if
if( ebctype.eq.1 ) then
  z(5) = zstart
end if

99 continue
  call rungek( z, znext, h, phi, beta, m, pr, gamma)
  alpha = alpha + h
  do 200 j = 1, 5
    z(j) = znext(j)
  200 continue
C
  if( alpha.lt.alphaend ) goto 99
  test = abs(zend(2)-z(4))*100.0
  if( test.gte.epsi ) then
    if( z(4).gt.zend(2) ) then
      zguess(4) = zstart
    else if( z(4).lt.zend(2) ) then
      zguess(3) = zstart
    end if
  goto 9
end if

return
end

Simfinal - subroutine to get final momentum and energy similarity
output with captured boundary conditions

Local variable definition:

f1 - tangential velocity at nodal locations
f2 - proportional to shear stress at nodal locations
g0  - enthalpy variable at nodal locations
j  - dummy counter

CC

subroutine simfinal( z, znext, zstart, phi, alpha, alphastart,
  alphaend, h, beta, m, pr, gamma, f2, g0, node )

   dimension z(5), znext(5), phi(5), zstart(5)
   dimension f1(50), f2(50), g0(50)
   real alpha, z, znext, h, alphastart, zstart, alphaend, phi
   real pr, gamma, f1, dum1, f2, g0, beta, m
   integer j, node, iplace

   iplace = 1
   f1(1) = 0.0
   f1(node) = 1.0
   f1(node+1) = 1.0e6
   dum1 = real(f1(node)-f1(1))/(node-1)
   do 100 j = 2, node - 1
     f1(j) = f1(j-1) + dum1
   100 continue

alpha = alphastart
   do 200 j = 1, 5
     z(j) = zstart(j)
   200 continue

   iplace = 1
   f2(iplace) = z(3)
   g0(iplace) = z(4)
   iplace = 2

   call rungee( z, znext, h, phi, beta, m, pr, gamma )
   if( znext(2).ge.f1(iplace) ) then
     f2(iplace) = ((znext(3)-z(3))/(znext(2)-z(2)))
     * (f1(iplace)-z(2)) + z(3)
     g0(iplace) = ((znext(4)-z(4))/(znext(2)-z(2)))
     * (f1(iplace)-z(2)) + z(4)
     iplace = iplace + 1
   end if
   alpha = alpha + h
do 300 j = 1, 5
    z(j) = znext(j)
    continue
    if( alpha.lt.alphaend ) goto 999
end

C
C Rungem - subroutine to calculate the 4th order runge-kutta
C iterates for z'=f(z) where
C
C z(n+1) = z(n) + h phi(z(n),h)
C phi = 1/6( k1 + 2 k2 + 2 k3 + k4 )
C and k's are defined by the local variable definitions
C
C
C
C Local variable definition:
C
C k1  - f(z)
C k2  - f(z + 0.5 h k1)
C k3  - f(z + 0.5 h k2)
C k4  - f(z + h k3)
C j    - dummy counter
C
C
C
C
C
C
C

C subroutine rungem( z, znext, h, phi, beta, m )

C
dimension z(5), znext(5), phi(5), k1(5), k2(5), k3(5), k4(5)
real z, znext, h, phi, beta, m
real k1, k2, k3, k4
real funcm
integer j

C
k1(1) = funcm( 1, z(1), z(2), z(3), beta, m )
k1(2) = funcm( 2, z(1), z(2), z(3), beta, m )
k1(3) = funcm( 3, z(1), z(2), z(3), beta, m )
k2(1) = funcm( 1, z(1) + 0.5*h*k1(1), z(2) + 0.5*h*k1(2),

\[ \begin{align*}
\text{k2(2)} &= \text{funcm}(2, z(1) + 0.5 \cdot h \cdot k1(1), z(2) + 0.5 \cdot h \cdot k1(2),
\quad z(3) + 0.5 \cdot h \cdot k1(3), \beta, m) \\
\text{k2(3)} &= \text{funcm}(3, z(1) + 0.5 \cdot h \cdot k1(1), z(2) + 0.5 \cdot h \cdot k1(2),
\quad z(3) + 0.5 \cdot h \cdot k1(3), \beta, m) \\
\text{k3(1)} &= \text{funcm}(1, z(1) + 0.5 \cdot h \cdot k2(1), z(2) + 0.5 \cdot h \cdot k2(2),
\quad z(3) + 0.5 \cdot h \cdot k2(3), \beta, m) \\
\text{k3(2)} &= \text{funcm}(2, z(1) + 0.5 \cdot h \cdot k2(1), z(2) + 0.5 \cdot h \cdot k2(2),
\quad z(3) + 0.5 \cdot h \cdot k2(3), \beta, m) \\
\text{k3(3)} &= \text{funcm}(3, z(1) + 0.5 \cdot h \cdot k2(1), z(2) + 0.5 \cdot h \cdot k2(2),
\quad z(3) + 0.5 \cdot h \cdot k2(3), \beta, m) \\
\text{k4(1)} &= \text{funcm}(1, z(1) + h \cdot k3(1), z(2) + h \cdot k3(2),
\quad z(3) + h \cdot k3(3), \beta, m) \\
\text{k4(2)} &= \text{funcm}(2, z(1) + h \cdot k3(1), z(2) + h \cdot k3(2),
\quad z(3) + h \cdot k3(3), \beta, m) \\
\text{k4(3)} &= \text{funcm}(3, z(1) + h \cdot k3(1), z(2) + h \cdot k3(2),
\quad z(3) + h \cdot k3(3), \beta, m) \\
\text{C} \\
\text{do 100 j = 1, 3} \\
\quad \text{phi(j)} = (1.0/6.0) \cdot (k1(j) + (2.0 \cdot k2(j))
\quad \quad + (2.0 \cdot k3(j)) + k4(j) ) \\
\quad \text{znext(j)} = z(j) + h \cdot \text{phi(j)} \\
100 & \text{continue} \\
\text{C} \\
\text{return} \\
\text{end} \\
\end{align*} \]
real function funcm( ifnum, a, b, c, beta, m )

real a, b, c, beta, m
integer ifnum

if( ifnum.eq.1 ) then
   funcm = b
else if( ifnum.eq.2 ) then
   funcm = c
else if( ifnum.eq.3 ) then
   funcm = -( beta*(1.0-b**2.0) + (1.0/(m+1.0))*a*c )
end if

return
end

---

Rungee - subroutine to calculate the 4th order Runge-Kutta

iterates for \( z' = f(z) \) where

\[
\begin{align*}
   z(n+1) &= z(n) + h \phi(z(n), h) \\
   \phi &= \frac{1}{6}( k1 + 2 k2 + 2 k3 + k4 )
\end{align*}
\]

and \( k' \)'s are defined by the local variable definitions

Local variable definition:

\[
\begin{align*}
   k1 &= f(z) \\
   k2 &= f(z + 0.5 h k1) \\
   k3 &= f(z + 0.5 h k2) \\
   k4 &= f(z + h k3) \\
   j &= \text{dummy counter}
\end{align*}
\]

---

subroutine rungee( z, znext, h, phi, beta, m, pr, gamma)
dimension z(5), znext(5), phi(5), k1(5), k2(5), k3(5), k4(5)
real z, znext, h, phi, beta, m, pr, gamma
real k1, k2, k3, k4
real func e
integer j

c

k1(1) = func e( 1, z(1), z(2), z(3), z(4), z(5), beta, m, 
1
pr, gamma )

k1(2) = func e( 2, z(1), z(2), z(3), z(4), z(5), beta, m, 
1
pr, gamma )

k1(3) = func e( 3, z(1), z(2), z(3), z(4), z(5), beta, m, 
1
pr, gamma )

k1(4) = func e( 4, z(1), z(2), z(3), z(4), z(5), beta, m, 
1
pr, gamma )

k1(5) = func e( 5, z(1), z(2), z(3), z(4), z(5), beta, m, 
1
pr, gamma )

k2(1) = func e( 1, z(1) + 0.5*h*k1(1), z(2) + 0.5*h*k1(2), 
1
z(3) + 0.5*h*k1(3), z(4) + 0.5*h*k1(4), 
2
z(5) + 0.5*h*k1(5), beta, m, pr, gamma )

k2(2) = func e( 2, z(1) + 0.5*h*k1(1), z(2) + 0.5*h*k1(2), 
1
z(3) + 0.5*h*k1(3), z(4) + 0.5*h*k1(4), 
2
z(5) + 0.5*h*k1(5), beta, m, pr, gamma )

k2(3) = func e( 3, z(1) + 0.5*h*k1(1), z(2) + 0.5*h*k1(2), 
1
z(3) + 0.5*h*k1(3), z(4) + 0.5*h*k1(4), 
2
z(5) + 0.5*h*k1(5), beta, m, pr, gamma )

k2(4) = func e( 4, z(1) + 0.5*h*k1(1), z(2) + 0.5*h*k1(2), 
1
z(3) + 0.5*h*k1(3), z(4) + 0.5*h*k1(4), 
2
z(5) + 0.5*h*k1(5), beta, m, pr, gamma )

k2(5) = func e( 5, z(1) + 0.5*h*k1(1), z(2) + 0.5*h*k1(2), 
1
z(3) + 0.5*h*k1(3), z(4) + 0.5*h*k1(4), 
2
z(5) + 0.5*h*k1(5), beta, m, pr, gamma )

k3(1) = func e( 1, z(1) + 0.5*h*k2(1), z(2) + 0.5*h*k2(2), 
1
z(3) + 0.5*h*k2(3), z(4) + 0.5*h*k2(4), 
2
z(5) + 0.5*h*k2(5), beta, m, pr, gamma )

k3(2) = func e( 2, z(1) + 0.5*h*k2(1), z(2) + 0.5*h*k2(2), 
1
z(3) + 0.5*h*k2(3), z(4) + 0.5*h*k2(4), 
2
z(5) + 0.5*h*k2(5), beta, m, pr, gamma )

k3(3) = func e( 3, z(1) + 0.5*h*k2(1), z(2) + 0.5*h*k2(2), 
1
z(3) + 0.5*h*k2(3), z(4) + 0.5*h*k2(4), 
2
z(5) + 0.5*h*k2(5), beta, m, pr, gamma )
\begin{verbatim}

k3(4) = func4( 4, z(1) + 0.5*h*k2(1), z(2) + 0.5*h*k2(2),
  z(3) + 0.5*h*k2(3), z(4) + 0.5*h*k2(4),
  z(5) + 0.5*h*k2(5), beta, m, pr, gamma )
k3(5) = func4( 5, z(1) + 0.5*h*k2(1), z(2) + 0.5*h*k2(2),
  z(3) + 0.5*h*k2(3), z(4) + 0.5*h*k2(4),
  z(5) + 0.5*h*k2(5), beta, m, pr, gamma )

k4(1) = func4( 1, z(1) + h*k3(1), z(2) + h*k3(2),
  z(3) + h*k3(3), z(4) + h*k3(4),
  z(5) + h*k3(5), beta, m, pr, gamma )
k4(2) = func4( 2, z(1) + h*k3(1), z(2) + h*k3(2),
  z(3) + h*k3(3), z(4) + h*k3(4),
  z(5) + h*k3(5), beta, m, pr, gamma )
k4(3) = func4( 3, z(1) + h*k3(1), z(2) + h*k3(2),
  z(3) + h*k3(3), z(4) + h*k3(4),
  z(5) + h*k3(5), beta, m, pr, gamma )
k4(4) = func4( 4, z(1) + h*k3(1), z(2) + h*k3(2),
  z(3) + h*k3(3), z(4) + h*k3(4),
  z(5) + h*k3(5), beta, m, pr, gamma )
k4(5) = func4( 5, z(1) + h*k3(1), z(2) + h*k3(2),
  z(3) + h*k3(3), z(4) + h*k3(4),
  z(5) + h*k3(5), beta, m, pr, gamma )

do 100 j = 1, 5
  phi(j) = (1.0/6.0)*( k1(j) + (2.0*k2(j))
  + (2.0*k3(j)) + k4(j) )
znext(j) = z(j) + h*phi(j)
100  continue

return
end

C
C Func4 - function to calculate the functional value
C
C f(z(1)) = z(2)
C f(z(2)) = z(3)
C f(z(3)) = -( beta*(1-z(2)^2) + (1/(m+1)) z(1) z(3) )
C f(z(4)) = z(4)
C f(z(5)) = -(pr/(m+1)) z(1) z(5) + 2 pr beta z(2) z(4))
C + (2(gamma-1)/(gamma+1))(1-pr)(z(3)^2 + z(2)z(3))
\end{verbatim}
C

Local variable definition:

func - value of function returned to subroutine

real function funce( ifnum, a, b, c, d, e, beta, m, pr, gamma )

real a, b, c, d, e, beta, m, pr, gamma
integer ifnum

if( ifnum.eq.1 ) then
  funce = b
else if( ifnum.eq.2 ) then
  funce = c
else if( ifnum.eq.3 ) then
  funce = -( beta*(1.0-b**2.0) + (1.0/(m+1.0))*a*c)
else if( ifnum.eq.4 ) then
  funce = e
else if( ifnum.eq.5 ) then
  funce = -(pr/(m+1.0))*a*e + 2.0*pr*beta*b*d
1
  - 2.0*((gamma-1.0)/(gamma+1.0))*(1.0-pr)
2
  * (c**2.0 - b*(( beta*(1.0-b**2.0)
3
  + (1.0/(m+1.0))*a*c))
end if

return
end

C

Cebsmith - subroutine to calculate a simple (zero equation) algebraic eddy-viscosity model for turbulent flow

C

Local variable definition:
eb2 - nodal value of shear stress variable
 eb3 - nodal value of entropy variable
 eb5 - nodal value of eddy-viscosity  
 eb6 - product of nodal value of shear stress and eddy-viscosity
 edb5db1 - derivative of b5 w/r/t b1
 rho - nodal value of density
 tau - shear stress variable
 s - entropy variable
 y - normal distance from airfoil surface
 utau - friction velocity
 yplus - used in van-driest damping
 pplus - used in van-driest damping for parametric correction
 aplus - used in van-driest damping
 l - mixing length
 mutmuin - ratio of eddy-viscosity and viscosity in inner region
 mutmuout - ratio of eddy-viscosity and viscosity in outer region
 blthick - boundary-layer thickness

CCC

C subroutine cebsmith( D4, eb2, eb3, eb5, eb6, edb5db1, c0, u, ue, pe,
 1      dpe, uinf, pinf, dsthick, xstep, transx, gamma,
 2      re, node )
C
C dimension D4(50), rho(50), tau(50), s(50), eb2(50), eb3(50), c0(2)
C dimension u(50), y(50), yplus(50), l(50), mutmuin(50), mutmuout(50)
C dimension eb5(50), eb6(50), edb5db1(50)
C real D4, rho, tau, s, eb2, eb3, c0, u, eb5, eb6, edb5db1
C real pplus, aplus, gamma, pe, ue, dpe, uinf, pinf, re
C real xstep, transx, dum1, dum2, dum3, blthick
C real ydum, y, yplus, l, mutmuin, mutmuout, dsthick
C real utau, testsqrt
C integer i, j, node
C
C if( xstep.ge.transx ) then
C
C---------------------------------------------------------------
C calculate the inner region
C---------------------------------------------------------------
C
C do 100 i = 1, node
 tau(i) = (1.0-u(i))*eb2(i)
\[ s(i) = (1.0 - u(i)) \cdot eb3(i) \]
\[ dum1 = \frac{((2.0 \cdot pe \cdot gamma) / (gamma + 1.0))}{(1.0 - (s(i) \cdot ((gamma - 1.0) / (gamma + 1.0)))} \]
\[ dum2 = \frac{1.0}{(1.0 - (s(i) \cdot ((gamma - 1.0) / (gamma + 1.0)))} \]
\[ \ast (u(i) \cdot u(i) \cdot 2.0) \cdot (ue \cdot ue \cdot 2.0) \]
\[ rho(i) = dum1 \cdot dum2 \]

100 continue
dum3 = sqrt(pin/(c0(1) \cdot sqrt(re) \cdot pe \cdot (rho(1) \cdot rho(1) \cdot 2.0) \cdot (tau(1) \cdot tau(1) \cdot 3.0)))
pplus = uinf \cdot dum3 \cdot dpe
testsqrt = 1.0 + 10.0 \cdot pplus
aplus = 28.0 / sqrt(abs(1.0 + 10.0 \cdot pplus))
upla = sqrt((c0(1) \cdot pe \cdot tau(1)) / (sqrt(re) \cdot rho(1) \cdot pinf)) \cdot ue
u(node) = 0.09
y(1) = 0.0
yplus(1) = 0.0
l(1) = 0.0
mutmumin(1) = 0.0
do 200 i = 2, node
    ydum = 0.0
    call array3(D4, 4, u, i)
do 300 j = 1, i
        ydum = ydum + D4(j) \cdot (1.0 / (eb2(j) \cdot rho(j)))
    continue
    y(i) = (uinf / (sqrt(re) \cdot ue)) \cdot ydum
    yplus(i) = (re \cdot (rho(1) \cdot rho(1) \cdot 2.0) \cdot pinf \cdot y(i) \cdot u(0) \cdot u(0)) / (uinf \cdot c0(1) \cdot pe)
    l(i) = 0.41 \cdot y(i) \cdot (1.0 - exp(-yplus(i) / aplus))
    mutmumin(i) = (re \cdot (rho(1) \cdot rho(1) \cdot 2.0) \cdot pinf \cdot (rho(1) \cdot rho(1) \cdot 3.0))
        \ast (1(i) \cdot 1(i) \cdot 2.0) \cdot tau(i)) / (c0(1) \cdot uinf \cdot 2.0) \cdot pe)
200 continue
u(node) = 1.00
blthick = y(node)
C
C calculate the outer region
C
C
C do 400 i = 1, node
    mutmou(i) = ((0.0158 \cdot re \cdot pinf \cdot ue) / (c0(1) \cdot pe \cdot uinf))
        \ast (rho(i) \cdot rho(i) \cdot 2.0) \cdot dsthick
2
    / (1.0 + 5.5 \cdot ((y(i) / blthick) \cdot blthick))
400 continue
C
C
C  determine the smaller values of mutmu in inner and outer regions
C----------------------------------------------------------------------------------
C
C  do 500 i = 1, node
   if( mutmuin(i).ge.mutmuout(i).or.i.eq.node ) then
      eb5(i) = 1.0 + mutmuout(i)
      eb6(i) = eb5(i)*eb2(i)
   else
      eb5(i) = 1.0 + mutmuin(i)
      eb6(i) = eb5(i)*eb2(i)
   end if
500  continue
C
C----------------------------------------------------------------------------------
C  assign eddy-viscosity equal to 1.0 if transition has not occurred
C----------------------------------------------------------------------------------
C
C  else
C  do 600 i = 1, node
   eb5(i) = 1.0
   eb6(i) = eb5(i)*eb2(i)
600  continue
end if
C
C----------------------------------------------------------------------------------
C  calculate the derivative of b5 w/r/t b1
C----------------------------------------------------------------------------------
C
C  do 700 i = 1, node-1
   edb5db1(i) = (eb5(i+1)-eb5(i))*(eb2(i+1)-eb2(i))
700  continue
   edb5db1(node) = (eb5(i)-eb5(i-1))*(eb2(i)-eb2(i-1))
C
return
end

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  Baldlomx - subroutine to calculate the eddy-viscosity based on the
C  baldwin-lomax zero equation model
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Local variable definition:

- `eb2` - nodal value of shear stress variable
- `eb3` - nodal value of entropy variable
- `eb5` - nodal value of eddy-viscosity
- `eb6` - product of nodal value of shear stress and eddy-viscosity
- `edb5db1` - derivative of b5 w/x+t b1
- `rho` - nodal value of density
- `tau` - shear stress variable
- `s` - entropy variable
- `y` - normal distance from airfoil surface
- `uta` - friction velocity
- `yplus` - used in van-driest damping
- `pplus` - used in van-driest damping for parametric correction
- `aplus` - used in van-driest damping
- `l` - mixing length
- `mutmuin` - ratio of eddy-viscosity and viscosity in inner region
- `mutmuout` - ratio of eddy-viscosity and viscosity in outer region
- `ffmax` - baldwin-lomax function in outer region
- `fmax` - maximum value of `ffmax`
- `betatau` - baldwin-lomax parametric correction factor
- `ckleb` - baldwin-lomax parameter
- `ccp` - baldwin-lomax parameter

Subroutine `baldlomx`:

```c
subroutine baldlomx( D4, eb2, eb3, eb5, eb6, edb5db1, c0, u, ue, due,
                     pe, dpe, uinf, pinf, dstick, xstep, transx,
                     gamma, re, node )
```

```c
dimension D4(50), rho(50), tau(50), s(50), eb2(50), eb3(50), c0(2)
dimension u(50), y(50), yplus(50), l(50), mutmuin(50), mutmuout(50)
dimension eb5(50), eb6(50), edb5db1(50), ffmax(50)
real D4, rho, tau, s, eb2, eb3, c0, u, eb5, eb6, edb5db1
real yplus, aplus, gamma, pe, ue, dpe, uinf, pinf, re
real xstep, transx, dum1, dum2, dum3
real ydum, y, yplus, l, mutmuin, mutmuout, dstick
real fmax, ymax, ffmax, betatau, ckleb, ccp, due
real utau
integer i, j, node
```
if (xstep.ge.transx) then

C
C---------------------------------------------------------------
C calculate the inner region
C---------------------------------------------------------------
C
do 100 i = 1, node
    tau(i) = (1.0-u(i))*eb2(i)
    s(i) = (1.0-u(i))*eb3(i)
    dum1 = ((2.0*pe*gamma)/(gamma+1.0))
    dum2 = 1.0/(1.0-(s(i)+((gamma-1.0)/(gamma+1.0))
          * (u(i)**2.0))*(ue**2.0))
    rho(i) = dum1/dum2
100  continue
    dum3 = sqrt(pinf/(c0(1)*sqrt(re)*pe*(rho(i)**3.0)*(tau(i)**3.0))
    pplus = uinf*dum3*dpe
    aplus = 26.0/sqrt(abs(1.0 + 10.0*pplus))
    utau = sqrt((c0(1)*pe*tau(i))/(sqrt(re)*rho(i)*pinf))*ue
    u(node) = 0.99
    y(1) = 0.0
    yplus(1) = 0.0
    l(i) = 0.0
    mutmuin(i) = 0.0
    do 200 i = 2, node
        ydum = 0.0
        call array3(D4, 4, u, i )
        do 300 j = 1, i
            ydum = ydum + D4(i)*(1.0/(eb2(j)*rho(j)))
        300  continue
        y(i) = (uinf/(sqrt(re)*ue))*ydum
        yplus(i) = (re*(rho(i)**2.0)*pinf*y(i)*utau)/(uinf*c0(1)*pe)
        l(i) = 0.41*y(i)*(1.0-exp(-yplus(i)/aplus))
        mutmuin(i) = ((re**3.0/2.0.0)*ue**2.0.0)*pinf*(rho(i)**3.0)
                     * (l(i)**2.0)*tau(i))/(c0(1)*(uinf**2.0)*pe)
200  continue
    u(node) = 1.00

C
C---------------------------------------------------------------
C calculate the outer region
C---------------------------------------------------------------
C
fmax = 0.0
\[
\text{ymax} = 0.0 \\
\text{do } 310 \text{ i = 1, node} \\
\quad \text{ffmax}(i) = \sqrt{\text{re}}(\text{ue} + 2.0/\text{uinf})y(i)\rho(i)\tau(i) \\
\qquad \times (1.0 - \exp(-y\text{plus}(i)/\text{aplus})) \\
\quad \text{if} (\text{ffmax}(i) \geq \text{fmax}) \text{ then} \\
\qquad \text{fmax} = \text{ffmax}(i) \\
\qquad \text{ymax} = y(i) \\
\quad \text{end if} \\
310 \text{ continue} \\
\text{betatau} = \text{y} = \text{due}/\tau \text{a} \\
\text{ckleb} = (2.0/3.0) - (0.01312/(0.1724 + \text{betatau})) \\
\text{ccp} = (3.0 - 4.0*\text{ckleb})/(2.0*\text{ckleb}*(2.0 - 3.0*\text{ckleb} + \text{ckleb}^2*3.0)) \\
\text{do } 400 \text{ i = 1, node} \\
\quad \text{mutmuout}(i) = (((0.0168*\text{ccp} + \text{re}^2*/\text{pmax})*\text{ymax})/(\text{cO}(i)*\text{p}^2*\text{uinf})) \\
\qquad \times (\text{rho}(i) * 2.0)^2 \text{ymax} \\
\qquad / (1.0 + 5.5^((\text{ckleb}^2*y(i)/\text{ymax})^2*6.0)) \\
400 \text{ continue} \\
\text{C} \\
\text{C---------------------------------------------------------------------------} \\
\text{C determine the smaller values of mutmu in inner and outer regions} \\
\text{C---------------------------------------------------------------------------} \\
\text{C} \\
\text{do } 500 \text{ i = 1, node} \\
\quad \text{if} (\text{mutmuin}(i) \geq \text{mutmuout}(i), \text{or}.i . \text{eq}. \text{node}) \text{ then} \\
\qquad \text{eb5}(i) = 1.0 + \text{mutmuout}(i) \\
\qquad \text{eb6}(i) = \text{eb5}(i) * \text{eb2}(i) \\
\quad \text{else} \\
\qquad \text{eb5}(i) = 1.0 + \text{mutmuin}(i) \\
\qquad \text{eb6}(i) = \text{eb5}(i) * \text{eb2}(i) \\
\quad \text{end if} \\
500 \text{ continue} \\
\text{C} \\
\text{C---------------------------------------------------------------------------} \\
\text{C assign eddy-viscosity equal to 1.0 if transition has not occurred} \\
\text{C---------------------------------------------------------------------------} \\
\text{C} \\
\quad \text{else} \\
\quad \text{do } 600 \text{ i = 1, node} \\
\qquad \text{eb5}(i) = 1.0 \\
\qquad \text{eb6}(i) = \text{eb5}(i) * \text{eb2}(i) \\
600 \text{ continue} \\
\text{end if}
C  calculate the derivative of b5 w/r/t b1
C  
C     do 700 i = 1, node-1
        edb5db1(i) = (eb5(i+1)-eb5(i))*(eb2(i+1)-eb2(i))
700    continue
        edb5db1(node) = (eb5(i)-eb5(i-1))*(eb2(i)-eb2(i-1))
C  
return
end

C  Liftdrag - subroutine to calculate lift, total drag, and viscous
C  drag on the airfoil
C
C  Local variable definition:
C
C     xcv - cartesian x-coordinate for visc
C     ycv - cartesian y-coordinate for visc
C     cfv - coefficient of friction
C     xci - cartesian x-coordinate for invisc
C     yci - cartesian y-coordinate
C     cpi - coefficient of pressure
C     sum1-sum9 - dummy variables for interim calculations
C     cnt - total normal force coefficient
C     cat - total axial force coefficient
C     clt - total lift force coefficient
C     cdt - total drag force coefficient
C     cnp - normal force coefficient due to pressure
C     cap - axial force coefficient due to pressure
C     cvn - normal force coefficient due to viscous
C     cav - axial force coefficient due to viscous
C     clp - lift force coefficient due to pressure
C     cdp - drag force coefficient due to pressure
C     clv - lift force coefficient due to viscous
C     cdv - drag force coefficient due to viscous
C     nev - number of viscous data points
nei - number of inviscid data points

subroutine liftdrag( aoa )

dimension xcv(400), ycv(400), cfv(400)
dimension xci(200), yci(200), cpi(200)
real xcv, ycv, xci, yci, cfv, cpi, aoa
real sum1, sum2, sum3, sum4
real sum5, sum6, sum7, sum8
real clt, cdv, clv, cnv, cav
real cdp, cpl, cap, cat, cnt, cnp
integer i, nev, nei, ile, ite, itel, jle, jteu, jtel, isearch

open( unit=15, file='liftdrag.dat', status='unknown' )
open( unit=16, file='integral.dat', status='old' )
open( unit=17, file='fcoef.dat', status='old' )
open( unit=18, file='external.dat', status='old' )
open( unit=19, file='pcoef.dat', status='old' )

--- read input

read(16,*) nev, jle, jtel, jteu
read(17,*) ( xcv(i), ycv(i), cfv(i), i = 1, nev )
read(18,*) nei, ile, itel, ite
read(19,*) ( xci(i), yci(i), cpi(i), i = 1, nei )

--- find stagnation point

isearch = int(0.1*(jle))
do 10 i = jle-isearch, jle+isearch
   if( ycv(i).ge.0.0 ) goto 20
10 continue
20 jle = i

--- calculate the normal force coefficient due to pressure on upper
C surface of airfoil
C-----------------------------------------------
C
sum1 = 0.0
do 100 i = ile, ite+1
   sum1 = sum1 + 0.5*(cpi(i)+cpi(i+1))*(xci(i+1)-xci(i))
100 continue
C
C calculate the normal force coefficient due to pressure on lower C surface of airfoil
C-----------------------------------------------
C
sum2 = 0.0
do 200 i = ile, ite+1, -1
   sum2 = sum2 + 0.5*(cpi(i)+cpi(i-1))*(xci(i-1)-xci(i))
200 continue
C
C calculate the normal force coefficient due to friction on upper C surface of airfoil
C-----------------------------------------------
C
sum3 = 0.0
do 300 i = jle, jte+1
   sum3 = sum3 + 0.5*(cfv(i)+cfv(i+1))*(ycv(i+1)-ycv(i))
300 continue
C
C calculate the normal force coefficient due to friction on lower C surface of airfoil
C-----------------------------------------------
C
sum4 = 0.0
do 400 i = jle-1, jte+1, -1
   sum4 = sum4 + 0.5*(cfv(i)+cfv(i-1))*(ycv(i-1)-ycv(i))
400 continue
C
C calculate the axial force coefficient due to pressure on upper C surface of airfoil
C-----------------------------------------------
C
    sum5 = 0.0
    do 500 i = ile, iteu-1
        sum5 = sum5 + 0.5*(cpi(i)+cpi(i+1))*(yci(i+1)-yci(i))
    500     continue
C
C calculate the axial force coefficient due to pressure on lower
C surface of airfoil
C
C    sum6 = 0.0
    do 600 i = ile, iter+1, -1
        sum6 = sum6 + 0.5*(cpi(i)+cpi(i-1))*(yci(i-1)-yci(i))
    600     continue
C
C calculate the axial force coefficient due to friction on upper
C surface of airfoil
C
C    sum7 = 0.0
    do 700 i = jle, jteu-1
        sum7 = sum7 + 0.5*(cfv(i)+cfv(i+1))*(xcv(i+1)-xcv(i))
    700     continue
C
C calculate the axial force coefficient due to friction on lower
C surface of airfoil
C
C    sum8 = 0.0
    do 800 i = jle-1, jteu+1, -1
        sum8 = sum8 + 0.5*(cfv(i)+cfv(i-1))*(xcv(i-1)-xcv(i))
    800     continue
C
C calculate force coefficients
C
C    cnt = -sum1 + sum2 + sum3 + sum4
    cat = sum5 - sum6 + sum7 + sum8
cnp = -sum1 + sum2
cap = sum5 - sum6
cnv = sum3 + sum4
cav = sum7 + sum8
clt = cnt*cost(aoa*3.1415926/180.0) - cat*sin(aoa*3.1415926/180.0)
cdt = cnt*sin(aoa*3.1415926/180.0) + cat*cos(aoa*3.1415926/180.0)
clp = cnp*cos(aoa*3.1415926/180.0) - cap*sin(aoa*3.1415926/180.0)
cdp = cnp*sin(aoa*3.1415926/180.0) + cap*cos(aoa*3.1415926/180.0)
clv = cnv*cos(aoa*3.1415926/180.0) - cav*sin(aoa*3.1415926/180.0)
cdv = cnv*sin(aoa*3.1415926/180.0) + cav*cos(aoa*3.1415926/180.0)

C
C-------------------------------------------------------------------------------------
C write output
C-------------------------------------------------------------------------------------
C
write(*,*) "Coeff Pressure Viscous Total"
write(*,*) "Normal", cnp, cnv, cnt
write(*,*) "Axial ", cap, cav, cat
write(*,*) "Lift ", clp, clv, clt
write(*,*) "Drag ", cdp, cdv, cdt
write(15,900) clt, cdt, cdp, cdv
900 format( 4e15.6 )
C
close(15)
close(16)
close(17)
close(18)
close(19)

C
return
end

CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC
C
C Bl2ext - subroutine to extrapolate the displacement thickness,
C transpiration velocity, boundary layer thickness, and
C momentum thickness for airfoil past separation,
C interpolate the viscous data to inviscid grid, and
C convert to "other" nondimensionalizaton
C
CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC...CCC
Local variable definition:

xsv - surface coordinate for visc
xcv - cartesian x-coordinate for visc
ycv - cartesian y-coordinate for visc
dsv - displacement thickness based on visc grid
vtv - transpiration velocity based on visc grid
xsi - surface coordinate for invisc
xci - cartesian x-coordinate for invisc
yci - cartesian y-coordinate for invisc
dsi - displacement thickness based on invisc grid
bli - boundary layer thickness based on inviscid grid
moi - momentum thickness based on inviscid grid
vti - transpiration velocity based on inviscid grid
slopes - dsi slope of linear extrapolation
slopebl - bli slope of linear extrapolation
slopemo - moi slope of linear extrapolation
dsint - dsi intercept for linear extrapolation
blint - bli intercept for linear extrapolation
moiint - moi intercept for linear extrapolation
xcr - critical x where extrapolation begins
icr - invisc nodal location of xcr
jcr - visc nodal location of xcr
nei - number of invisc grid points
nev - number of visc grid points
ile - inviscid location of leading edge
ijteu - inviscid location of trailing edge on upper surface
ijtel - inviscid location of trailing edge on lower surface
jle - viscous location of first step from stagnation
ijteu - viscous location of the start of extrapolation on us
ijtel - viscous location of the start of extrapolation on ls
imax - inviscid nodal location halfway between trailing edge
and far end of inviscid grid

subroutine bl2ext( pinf )

dimension xcv(400), ycv(400), xsv(400), dsv(400), vtv(400)
dimension mov(400), blv(400), moi(200), bli(200)
dimension xci(200), yci(200), xsi(200), dsi(200), vti(200)
dimension uei(200), duei(200), rhoei(200), drhoei(200)
real xcv, ycv, dsv, vtv, xsv
real xci, uei, duei, rhoei, drhoei, xsi
real yci, dsi, vti, bli, moi, blv, mov
real slopeds, dsint, slopem0, moint, slope1, blint, xcr
real dum, pinf
integer i, j, nei, nev, icr, jcr
integer ile,itel, jteu, jle, jtel, jteu, imax

C
open( unit=15, file='external.dat', status='old' )
open( unit=16, file='integral.dat', status='old' )
open( unit=17, file='bl2ext.dat', status='unknown' )

C
read input

C
read(15,*) nei, ile, itel, jteu, dum, dum, dum
read(16,*) ( xci(i), yci(i), xsi(i), uei(i), duei(i), dum, dum,
1     rhoei(i), drhoei(i), i = i, nei )
read(16,*) nev, jle, jtel, jteu
read(16,*) ( xcv(i), ycv(i), xsv(i), dsv(i), vtv(i), blv(i),
1     mov(i), i = i, nev )

C
C upper surface

C obtain extrapolation point

C
jcr = jteu
xcr = xcv(jteu)
do 100 i = ile, nei
   if( xci(i).ge.xcr ) goto 10
100 continue
10 icr = i
xcr = xci(icr)

C
C interpolate displacement thickness, transpiration velocity,
C boundary layer thickness, and momentum thickness to xci = xcr

C
do 200 i = ile, icr
    do 300 j = jle, jcr
        if ( xsv(j).ge.xsi(i) ) goto 30
    continue
300     continue
30      continue
    dsi(i) = (((xsi(i) - xsv(j-1))*(dsv(j) - dsv(j-1)))
       / (xsv(j) - xsv(j-1))) + dsv(j-1)
    vti(i) = (((xsi(i) - xsv(j-1))*(vtv(j) - vtv(j-1)))
       / (xsv(j) - xsv(j-1))) + vtv(j-1)
    bli(i) = (((xsi(i) - xsv(j-1))*(blv(j) - blv(j-1)))
       / (xsv(j) - xsv(j-1))) + blv(j-1)
    moi(i) = (((xsi(i) - xsv(j-1))*(mov(j) - mov(j-1)))
       / (xsv(j) - xsv(j-1))) + mov(j-1)
200     continue
C
C---------------------------------------------
C    find the slope and intercept of the extrapolation for
C    displacement thickness, boundary layer thickness, and
C    momentum thickness
C---------------------------------------------
C
    if( xcr.lt.xci(iteu) ) then
        slopeds = (dsi(icr)-dsi(icr-1))/(xci(icr)-xci(icr-1))
        dsiint = dsi(icr) - slopeds*xcr
        slopebl = (bli(icr)-bli(icr-1))/(xci(icr)-xci(icr-1))
        blint = bli(icr) - slopebl*xcr
        slopemo = (moi(icr)-moi(icr-1))/(xci(icr)-xci(icr-1))
        moint = moi(icr) - slopemo*xcr
    C
C---------------------------------------------
C    extrapolate displacement thickness, boundary layer thickness,
C    and momentum thickness (linearly) from xcr to xci(iteu)
C---------------------------------------------
C
    do 400 i = icr, iteu
        dsi(i) = dsiint + slopeds*xci(i)
        bli(i) = blint + slopebl*xci(i)
        moi(i) = moint + slopemo*xci(i)
400     continue
C
C---------------------------------------------
blend displacement thickness, boundary layer thickness, and
momentum thickness (exponentially) from xci(iteu) to xci(imax)

imax = iteu + int((nei-iteu)/2)
do 500 i = iteu+1, imax
   dsi(i) = dsi(iteu)*exp(-(xci(i)-xci(iteu))*xci(imax))
   bli(i) = bli(iteu)*exp(-(xci(i)-xci(iteu))*xci(imax))
   moi(i) = moi(iteu)*exp(-(xci(i)-xci(iteu))*xci(imax))
500   continue
C
calculate new transpiration velocity based on extrapolated
and blended displacement thickness

do 600 i = icr, imax
   vti(i) = uei(i)*((dsi(i)-dsi(i-1))/(xsi(i)-xsi(i-1)))
       + duei(i)*dsi(i)
       + (drhoei(i)*uei(i)/rhoei(i))*dsi(i)
600   continue
C
lower surface
obtain extrapolation point

jcr = jtel
xcr = xcv(jtel)
do 150 i = ile, 1, -1
   if( xci(i).ge.xcr ) goto 15
15   icr = i
    xcr = xci(icr)
C
interpolate displacement thickness, transpiration velocity,
boundary layer thickness, and momentum thickness to xci = xcr

do 250 i = ile, icr, -1
do 350 j = jle, jcr, -1
    if (xsv(j).ge.xsi(i)) goto 35
350      continue
35      continue

    dsi(i) = (((xsi(i) - xsv(j+1))*(dsv(j) - dsv(j+1)))
        / (xsv(j) - xsv(j+1)) + dsv(j+1)
    vti(i) = (((xsi(i) - xsv(j+1))*(vtv(j) - vtv(j+1)))
        / (xsv(j) - xsv(j+1)) + vtv(j+1)
    bli(i) = (((xsi(i) - xsv(j+1))*(blv(j) - blv(j+1)))
        / (xsv(j) - xsv(j+1)) + blv(j+1)
    moi(i) = (((xsi(i) - xsv(j+1))*(mov(j) - mov(j+1)))
        / (xsv(j) - xsv(j+1)) + mov(j+1)

250      continue

C
C-----------------------------------------------
C      find the slope and intercept of the extrapolation for
C      displacement thickness, boundary layer thickness, and
C      momentum thickness
C-----------------------------------------------
C
    if (xcr.lt.xci(itel)) then
      slopeds = (dsi(icr)-dsi(icr+1))/(xci(icr)-xci(icr+1))
      dsint = dsi(icr) - slopeds*xcr
      slopebl = (bli(icr)-bli(icr+1))/(xci(icr)-xci(icr+1))
      blint = bli(icr) - slopebl*xcr
      slopemo = (moi(icr)-moi(icr+1))/(xci(icr)-xci(icr+1))
      moint = moi(icr) - slopemo*xcr
C
C-----------------------------------------------
C      extrapolate displacement thickness, boundary layer thickness,
C      and momentum thickness (linearly) from xcr to xci(itel)
C-----------------------------------------------
C
    do 450 i = icr, itel, -1
      dsi(i) = dsint + slopeds*xci(i)
      bli(i) = blint + slopebl*xci(i)
      moi(i) = moint + slopemo*xci(i)
450      continue
end if

C
C-----------------------------------------------
C      blend displacement thickness, boundary layer thickness, and
C momentum thickness (exponentially) from xci(itel) to xci(imax)

```
C

imax = itel - int((itel-1)/2)
do 650 i = itel-1, imax, -1
    dsi(i) = dsi(itel)*exp(-(xci(i)-xci(iteu))*xci(imax))
    bli(i) = bli(itel)*exp(-(xci(i)-xci(iteu))*xci(imax))
    moi(i) = moi(itel)*exp(-(xci(i)-xci(iteu))*xci(imax))
 650 continue
C
```

C calculate new transpiration velocity based on extrapolated and blended displacement thickness

```
C

do 650 i = icr, imax, -1
    vti(i) = uei(i)*((dsi(i)-dsi(i+1))/(xsi(i)-xsi(i+1)))
    1 + duei(i)*dsi(i)
    2 + (drhoei(i)*uei(i)/rhoi(i))*dsi(i)
 650 continue
C
```

C convert transpiration velocity from boundary layer nondimensionalization to "other" nondimensionalization

```
C

do 700 i = 1, nei
    vti(i) = vti(i)/sqrt(pinf)
 700 continue
C
```

C write output

```
C

do 750 i = 1, nei
    write(17,800) xci(i), dsi(i), vti(i), bli(i), moi(i)
 750 continue
800 format( 5e15.6 )
C
```

close(15)
close(16)
close(17)
return
end

Stagconv - subroutine to reorder the output files

Local variable definition:

xcv - cartesian x based on viscous grid
ycv - cartesian y based on viscous grid
xsv - surface x based on viscous grid
dsv - displacement thickness based on viscous grid
vtv - transpiration velocity based on viscous grid
blv - boundary layer thickness based on viscous grid
mov - momentum thickness based on viscous grid
cfv - coefficient of friction based on viscous grid
nout1 - total number of data points on upper
nout2 - total number of data points on lower

subroutine stagconv

dimension xcv(400), ycv(400), xsv(400), dsv(400), vtv(400)
dimension blv(400), mov(400), cfv(400)
real xcv, ycv, xsv, dsv, vtv, blv, mov, cfv
integer nout1, ntol1, nout2, ntol2, i

open( unit=15, file='points1.dat', status='old' )
open( unit=16, file='points2.dat', status='old' )
open( unit=17, file='integral1.dat', status='old' )
open( unit=18, file='integral2.dat', status='old' )
open( unit=19, file='fcoeff1.dat', status='old' )
open( unit=20, file='fcoeff2.dat', status='old' )
open( unit=21, file='integral.dat', status='unknown' )
open( unit=22, file='fcoeff.dat', status='unknown' )
C       read input
C---------------------------------------------------------------
C
read(15,*) nout1, ntol1
read(16,*) nout2, ntol2
read(18,*) ( xcv(i), ycv(i), xsv(i), dsv(i), vtv(i), blv(i),
             mov(i), i = nout2, 1, -1 )
read(17,*) ( xcv(i), ycv(i), xsv(i), dsv(i), vtv(i), blv(i),
             mov(i), i = nout2, nout1+ntol2-1 )
read(20,*) ( xcv(i), ycv(i), cfv(i), i = nout2, 1, -1 )
read(19,*) ( xcv(i), ycv(i), cfv(i), i = nout2, nout1+ntol2-1 )
C
C---------------------------------------------------------------
C       write output
C---------------------------------------------------------------
C
dsv(nout2) = (dsv(nout2+1)+dsv(nout2-1))*0.5
vtv(nout2) = (vtv(nout2+1)+vtv(nout2-1))*0.5
write(21,100) nout1+ntol2-1, nout2, nout2-ntol2+1, nout2+ntol1-1
write(21,200) ( xcv(i), ycv(i), xsv(i), dsv(i), vtv(i), blv(i),
            mov(i), i = 1, nout1+ntol2-1 )
100  format( 4i4  )
200  format( 7e15.6 )
300  format( 3e15.6 )
C
close(15)
close(16)
close(17)
close(18)
close(19)
close(20)
close(21)
close(22)
C
    return
end

C       subroutine to relax the displacement thickness,
C       transpiration velocity, boundary layer thickness,
and momentum thickness

Local variable definition:

xci - cartesian x from inviscid grid
dsi - displacement thickness interpolated to inviscid grid
disold - displacement thickness interpolated to inviscid grid from
  previous vii iteration
vti - transpiration velocity interpolated to inviscid grid
vtiold - transpiration velocity interpolated to inviscid grid from
  previous vii iteration
bli - boundary layer thickness interpolated to inviscid grid
bliold - boundary layer thickness interpolated to inviscid grid from
  previous vii iteration
moi - momentum thickness interpolated to inviscid grid
moiold - momentum thickness interpolated to inviscid grid from
  previous vii iteration
disrel - relaxed displacement thickness
vtirel - relaxed transpiration velocity
blirel - relaxed boundary layer thickness
moirel - relaxed momentum thickness

subroutine relaxv( relax, nei, inum )

dimension dsi(200), disold(200), vti(200), vtiold(200)
dimension disrel(200), vtirel(200), xci(200)
dimension bli(200), bliold(200), moi(200), moiold(200)
dimension blirel(200), moirel(200)
real dsi, disiold, vti, vtiold, bli, bliold, moi, moiold, xci
real disirel, vtirel, blirel, moirel, dum, relax
integer i, nei, inum

open( unit=16,file='bl2ext.dat',status='old' )
if ( inum.gt.1 ) then
  open( unit=17,file='bl2extold.dat',status='old' )
end if
C read input
C--------------------------------------------------
C
read(16,*) ( xci(i), dsi(i), vti(i), bli(i), moi(i), i = 1, nei )
if( inum.gt.1 ) then
read(17,*) ( dum, dsiold(i), vtiold(i), bliold(i), moioold(i),
1 i = 1, nei )
end if
rewind(16)
C--------------------------------------------------
C perform relaxation
C---------------------------------------------
C
do 100 i = 1, nei
   dsirel(i) = relax*dsi(i) + (1.0-relax)*dsiold(i)
   vtirel(i) = relax*vti(i) + (1.0-relax)*vtiold(i)
   blirel(i) = relax*bli(i) + (1.0-relax)*bliold(i)
   moirel(i) = relax*moi(i) + (1.0-relax)*moioold(i)
100 continue
C---------------------------------------------
C write output
C--------------------------------------------------
C
write(16,200) ( xci(i), dsirel(i), vtirel(i), blirel(i),
1 moirel(i), i = 1, nei )
200 format( 5e15.6 )
C close(16)
close(17)
C
return
end
Bibliography


