A COUPLED WAKE-INTEGRAL/VORTICITY CONFINEMENT TECHNIQUE
FOR THE PREDICTION OF DRAG FORCE

A Thesis

Presented to

The Graduate Faculty of The University of Akron

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Troy Snyder

December, 2012
A COUPLED WAKE-INTEGRAL/VORTICITY CONFINEMENT TECHNIQUE
FOR THE PREDICTION OF DRAG FORCE

Troy Snyder

Thesis

Approved:

Advisor
Dr. Alex Povitsky

Dean of the College
Dr. George K. Haritos

Faculty Reader
Dr. Abilash Chandy

Dean of the Graduate School
Dr. George R. Newkome

Faculty Reader
Dr. Scott Sawyer

Date

Department Chair
Dr. Celal Batur

ii
ABSTRACT

This work couples the two enabling technologies of wake-integral drag prediction and Vorticity Confinement (VC) for the improved prediction of drag from an Euler CFD simulation. Induced drag computations of a thin wing are shown to be more accurate than the more common method of surface pressure integration when compared to Prandtl lifting-line theory. Furthermore, the Vorticity Confinement method is shown to improve trailing vortex preservation and counteract the shift from induced to entropy drag as the distance with which the vortex convects downstream of the wing increases.

The desired application of this work is drag prediction, most notably induced drag, for aerodynamic design. High-fidelity Euler CFD simulations are desirable as they provide the most complete inviscid flowfield solution. However, accurate induced drag prediction via the surface integration of pressure is generally intractable barring a sufficiently refined surface grid and resultant increase in computational load. Furthermore, the alternative wake-integral technique for drag prediction suffers from numerical dissipation. VC is shown to control the numerical dissipation with very modest computational overhead.
VC is implemented in both a two-dimensional finite-volume Euler code written by the author as well as the commercial cfd code ANSYS FLUENT. The two-dimensional research code is used to test specific formulations of the VC body force terms and illustrate the computational efficiency of the method compared to a 'brute force' reduction in spatial step size. For a three-dimensional wing simulation, ANSYS FLUENT is employed with the VC body force terms added to the solver with user-defined functions (UDFs).
ACKNOWLEDGEMENTS

I would like to acknowledge financial and technical support from the AFRL/DAGSI Ohio Student-Faculty Research Fellowship Topic # RB09-8 sponsored by Drs. Jose Camberos and Raymond Kolonay. I would like to thank my advisor Dr. Alex Povitsky for technical guidance. Also, I would like to thank Ernest Thompson for his help during summers spent at the AFRL in Dayton.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Wake-Integral Method Development and Application</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Numerical Diffusion</td>
<td>8</td>
</tr>
<tr>
<td>1.4 Counteracting Numerical Diffusion</td>
<td>11</td>
</tr>
<tr>
<td>1.5 Vorticity Confinement Development and Application</td>
<td>13</td>
</tr>
<tr>
<td>1.6 Research Goals</td>
<td>17</td>
</tr>
<tr>
<td>II. METHODOLOGY</td>
<td>19</td>
</tr>
<tr>
<td>2.1 Derivation of Wake-Integral Drag Formulas</td>
<td>19</td>
</tr>
<tr>
<td>2.2 The Vorticity Confinement Method</td>
<td>30</td>
</tr>
<tr>
<td>III. CFD CODE DESCRIPTIONS</td>
<td>39</td>
</tr>
<tr>
<td>3.1 2DFlow–General Description and Implementation of VC</td>
<td>39</td>
</tr>
<tr>
<td>3.2 ANSYS FLUENT</td>
<td>42</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Advecting Vortex Physical Parameters</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>Advecting Vortex Computational Parameters</td>
<td>52</td>
</tr>
<tr>
<td>6.1</td>
<td>Rectangular Planform Wing Physical Parameters</td>
<td>75</td>
</tr>
<tr>
<td>6.2</td>
<td>Rectangular Planform Wing Computational Models [1]</td>
<td>75</td>
</tr>
<tr>
<td>6.3</td>
<td>Rectangular planform wing root mean square deviation (RMSD) or standard error of wake-integral induced drag coefficient</td>
<td>80</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Control volume surrounding aircraft and wake-survey plane</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Sod Shock Tube Results, Normalized Plots of Density, Pressure, and Velocity, $T = 0.00636$ s.</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Velocity vectors: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with no confinement</td>
<td>60</td>
</tr>
<tr>
<td>5.2</td>
<td>Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with no confinement ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)</td>
<td>61</td>
</tr>
<tr>
<td>5.3</td>
<td>Velocity vectors: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with $\epsilon = 25$</td>
<td>62</td>
</tr>
<tr>
<td>5.4</td>
<td>Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with $\epsilon = 25$ ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)</td>
<td>63</td>
</tr>
<tr>
<td>5.5</td>
<td>Velocity vectors: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with $\epsilon = 100$</td>
<td>64</td>
</tr>
<tr>
<td>5.6</td>
<td>Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with $\epsilon = 100$ ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)</td>
<td>65</td>
</tr>
<tr>
<td>5.7</td>
<td>Profiles of velocity and vorticity through vortex center: $0\Delta t$, $500\Delta t$ with no VC, $500\Delta t$ with $\epsilon = 25$, and $500\Delta t$ with $\epsilon = 100$</td>
<td>66</td>
</tr>
<tr>
<td>5.8</td>
<td>Temporal evolution of two-dimensional advecting vortex circulation (no VC, $\epsilon = 25$, and $\epsilon = 100$)</td>
<td>67</td>
</tr>
<tr>
<td>5.9</td>
<td>Velocity vectors: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$, compressible confinement with $\epsilon = 100$</td>
<td>68</td>
</tr>
<tr>
<td>5.10</td>
<td>Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$, compressible confinement with $\epsilon = 100$ ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)</td>
<td>69</td>
</tr>
<tr>
<td>5.11</td>
<td>Comparison of circulation and computation time, 2-D advecting vortex</td>
<td>70</td>
</tr>
</tbody>
</table>
6.1 Rectangular planform wing and computational plenum .......................... 83

6.2 Computational Grid 1: (a) wing surface, (b) wake region .......................... 83

6.3 Computational Grid 2: (a) wing surface, (b) wake region .......................... 84

6.4 Rectangular planform wing trailing vortex at varying Trefftz plane locations, Grid 1, isolines of vorticity (|ω| varying from 0 to 300, Δ|ω| = 15): (a)-(c) No Confinement, (d)-(f) Scaled Confinement $c_{vc} = 0.075$ ........................ 85

6.5 Rectangular planform wing trailing vortex at varying Trefftz plane locations, Grid 2, isolines of vorticity (|ω| varying from 0 to 300, Δ|ω| = 15): (a)-(c) No Confinement, (d)-(f) Scaled Confinement $c_{vc} = 0.075$ ........................ 86

6.6 Rectangular planform wing wake-integral induced drag coefficient at various Trefftz plane locations in chords downstream of trailing edge, linear curve fitting: (a) First-Order, (b) Second-Order, lifting-line $C_{Di} = 0.0056$ ........................ 87

6.7 Rectangular planform wing wake-integral entropy drag coefficient at various Trefftz plane locations in chords downstream of trailing edge, linear curve fitting: (a) Grid 1, (b) Grid 2, lifting-line $C_{Di} = 0.0056$ ........................ 87
CHAPTER I

INTRODUCTION

1.1 Background

During the flight of an aircraft, the two primary components of aerodynamic force are lift and drag. It is often the goal of aerodynamicists to characterize these forces and their changes as a function of design parameters in order to optimize a final aerodynamic configuration. With the ever increasing expense of experimental investigation, many researchers have put more reliance on computational techniques for lift and drag prediction. As a result, computational methods have begun to displace wind tunnel testing in many aspects of an aircraft design cycle to reduce costs and to improve optimization procedures [2]. While experiments are still necessary for accurate drag predictions of complex flows, there are many situations during aircraft design when simple physical models solved numerically can provide the requisite design information.

For preliminary design, it is often enough to determine the surface pressure distribution of an aircraft in order to determine wing loadings and structural requirements for a particular design. As a result, aerodynamic designers often employ numerical methods for the solution of inviscid flows such as: vortex lattice methods,
panel methods, full-potential CFD codes, and Euler CFD codes. Vortex lattice methods, panel methods, and full-potential CFD codes are much lower computational cost than Euler CFD codes. However, these methods all require a priori specification of the wake whereas the freely deforming wake shape is captured in the Euler solution [3]. Thus, the Euler CFD codes provide the most complete description of an inviscid flow [3] and promotes their usage in the present work.

In addition to providing surface pressures, Euler CFD codes also facilitate the prediction of lift-induced drag. The lift-induced or induced drag is an inviscid phenomena that arises due to the pressure difference between upper and lower surfaces of a wing. This pressure difference causes air flow around the wing tip and the formation of vortices at the trailing edge of the wing. The vortices then roll-up into wingtip vortices which induce a downward velocity component behind the trailing edge of the wing referred to as downwash. The downwash velocity reduces the effective angle of attack and tilts the lift vector such that a portion of the lift is projected in the direction freestream flow. This is know as the lift-induced drag [4].

At subsonic speeds, the lift-induced drag generally constitutes approximately 40% of the total drag for transport aircraft at cruise conditions. During takeoff and landing the induced drag can increase to 80-90% of the total drag. Additionally, the takeoff portion of the flight, in which the dominant drag is lift-induced, can have significant influence on overall aircraft design. "Conditions associated with engine-out climb shortly after takeoff are often critical constraints in the aircraft design and changes in aircraft performance at these conditions influence the overall design
and so have an indirect, but powerful, effect on the aircraft cruise performance [5].”

While induced drag can significantly constrain overall aircraft design, it is also readily reduced by design changes which alter the surface pressure distribution of the aircraft. However, before design changes can be implemented, some method must be employed to extract the drag from numerical simulation.

Typically, the computation of lift and drag force from an Euler CFD simulation involves integrating pressure over the surface of the aerodynamic body, referred to as the near-field method. This method generally provides accurate lift predictions, however, accurate drag predictions for subsonic and transonic flow regimes can be difficult to obtain barring a sufficiently refined grid at the near-surface region of the aircraft. The refined surface grid is necessary as the curved aerodynamic surfaces must be approximated with flat facets [6]. For example, wings designed for low-speed flight are generally characterized by a curved leading edge which is approximated by flat facets defining the faces of control volumes of the computational mesh. It is not until the facets become sufficiently small, i.e. the mesh sufficiently refined, that the curved surface can be closely approximated. In the near-field technique, the pressure drag integral is computed by the dot product of pressures and normal vectors at the wing surface. As the normal vectors are defined by the flat facets approximating the curved surface of the wing, the direction of projected pressures and thus the pressure drag integral becomes highly sensitive the the mesh density along the curved surface of the wing. Additionally, for lower-speed subsonic and transonic flows, the near-field
technique can suffer a loss of accuracy due to the addition and subtraction of surface pressures very close in magnitude.

As an alternative to the near-field technique, the wake-integral technique is proposed for drag computation from CFD simulation. As the name would suggest, the wake-integral technique computes drag via integration of flow variables over an arbitrary cross-flow plane in the wake of the aircraft also known as a Trefftz plane [7]. This method of drag computation directly overcomes the major shortcomings of the near-field method as the integration is performed within the flowfield and not at the interface of the flowfield and solid body surfaces. As a result, the wake-integral method does not suffer from the inaccuracies incurred by the misdirection of projected pressures or the additive cancellation of pressures very close in magnitude. Additionally, the wake-integral method has the benefit of providing additional insight into the physical sources of drag [8].

Thus far, the discussion has been implicitly limited to subsonic, inviscid flows for which the only drag present is induced drag. However, more complex flows may also include profile drag arising from viscous effects and wave drag arising from the formation of shock waves. The wake-integral method has the advantage in that the drag can be decomposed into these physically relevant components as disparate integrals. This is in contrast to the near-field method which computes drag based on the integration of the mechanical forces of pressure and shear stress. The phenomenological breakdown of drag achieved with the wake-integral method can then be used to better guide design decisions [8].
1.2 Wake-Integral Method Development and Application

Before discussing the development of wake-integral and the more general class of far-field drag prediction methods, the terminology associated with such techniques requires some clarification. The term 'far-field' applies to any method which involves integration within the flowfield whereas the term 'near-field' applies to the more commonly employed method of integration at the interface of solid surfaces and the surrounding fluid. The wake-integral method is a subset of the far-field methods which involves integration over a single cross-flow plane within the wake of an aircraft commonly known as a Trefftz plane [7]. Thus the method is also known as Trefftz-plane analysis. Additional far-field methods may employ surface integration over one or more surfaces within the flowfield or they may even involve volume integration over specific regions of the flowfield.

In the present work, the wake-integral method is applied in the extraction of drag from an Euler CFD solution. However, wake-integral methods were initially developed and applied by experimentalists. Typically, wind-tunnel experiments employ strain-gauge balances to measure lift and drag forces. While lift is relatively easy to measure, drag is often an order of magnitude smaller and can be difficult to predict accurately with strain-gauge balances. As a result, some early experimenters developed a method for determining drag by measuring the momentum deficit parallel to the freestream in the wake of an experimental model [9]. Such a method involved performing a wake-survey of flow variables to compute the momentum deficit and
resultant drag. However, the early formulation developed by [9] did not account for the induced or vortex drag component and was modified by [10] to account for both profile and induced components. Additional improvements to the wake-integral method have been applied in experiments by the authors in [11] and [12].

Eventually, the wake-integral method developed by experimentalists was adopted by some CFD researchers. This was motivated partly by the previously successful application of the method in experiments but also by the inaccuracy of the near-field surface integration of pressures, the computational equivalent of force measurement in a wind tunnel [8]. Wake-integral prediction of drag in CFD has proven to be reasonably accurate and has been used to predict induced, profile, and wave drag components. Euler simulations of subsonic (elliptic wing) and transonic (ONERA M6 wing) flows conducted by [13] showed that the wake-integral method outperformed the near-field method in predicting induced drag with the value closer to theoretical lifting-line. In general, the near-field integration of surface pressure over-predicted theoretical drag predictions whereas the wake-integral method computed drag values slightly less than the analytical. Similar results were also obtained by the authors in [6, 14, 15, 16, 17] for the prediction of induced drag of finite wings using both Euler and Navier-Stokes solvers.

The authors in [14] highlighted the ability of the wake-integral method to separate drag into physically relevant induced, wave, and profile components. Additionally, they applied an entropy correction to account for the spurious entropy generated by numerical dissipation of the flow solver (numerical dissipation and spu-
rious entropy drag will be discussed shortly). Similarly, the authors in [15] discussed the shift from the induced drag integral to the entropy drag integral as the Trefftz plane moved farther downstream of the aircraft. The same authors also experimented with various cutoffs to reduce the size of the plane over which integration must be performed in the wake as an effort to reduce computational overhead and prevent contamination by the computational boundaries. The authors in [16] employed cutoffs as well but as a means to separate the shock wake and vortical wake and thereby separate wave and profile drag components.

More recently, the authors in [18] compared drag predictions via wake-integration and volume-integration, referred to as the mid-field method. By making use of the divergence theorem to transform between volume and surface integration, the authors in [18] show the equivalence of near/mid/far-field methods analytically and compare the numerically predicted drag. This mid-field method of decomposing drag carries into the work conducted by the same authors in [19]. In that case, the presence of spurious entropy drag predicted by flowfield volume integration is employed as an indicator for mesh refinement. Rather than use local indicators such as flow gradients, the authors argue that global indicators such as drag may be more useful in adaptive mesh refinement schemes.

As the majority of commercial CFD codes employ first- or second-order discretization for convective fluxes, numerical dissipation can smear the wake behind an aircraft and degrade the accuracy of wake-integral drag predictions.
In order to counteract the numerical diffusion of the vortical wake, a method known as "Vorticity Confinement (VC)" [20] is proposed. The essence of the VC method is to provide additional momentum forcing to counteract numerical diffusion inherent to the computational scheme in vortical regions of the flow-field. The original VC formulation, referred to as VC1, takes the form of an additional source term added to the momentum equations with the degree of confinement controlled by a heuristic constant or ‘confinement parameter’.

1.3 Numerical Diffusion

While the wake-integral method provides a number of advantages over the near-field method, a significant obstacle in its application to high-fidelity CFD simulations such as Euler and Navier-Stokes solutions is smearing of the vortical wake due to numerical diffusion. Whether implicit in the discretization or explicitly added for stability, numerical algorithms for the solution of Euler or Navier-Stokes introduce some artificial smoothing of the solution. This is problematic for the prediction of drag via wake-integration as the vortical wake may be artificially smeared and its strength reduced by the time it reaches the Trefftz plane downstream. The numerical diffusion downstream of the aircraft generally reduces the induced drag predicted via wake-integration and generates what is referred to as spurious entropy drag. There are several options to remedy this situation and counteract numerical diffusion, but let us first illustrate how numerical diffusion arises in the numerical solution by discretizing a simple PDE as an example.
In CFD, the solution of specific partial differential equations is desired. However, the numerical solutions are always somewhat erroneous. The accuracy of the numerical solutions is often characterized in terms of truncation error but may alternatively be described by the solution of a modified differential equation. For example, the finite-difference solution of the one-dimensional linear wave-equation (wave speed \( a \)) with a first-order forward difference in time and first-order rearward difference in space yields Equation 1.1 [21].

\[
\frac{u_{i}^{t+\Delta t} - u_{i}^{t}}{\Delta t} + a \frac{u_{i}^{t} - u_{i-\Delta x}^{t}}{\Delta x} = 0
\]  

(1.1)

The finite-difference equation does not solve the linear wave equation but rather a modified equation given by Equation 1.2 [21].

\[
\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = \frac{a \Delta x}{2} (1 - \nu) \frac{\partial^2 u}{\partial^2 x} + \frac{a(\Delta x)^2}{6} (3\nu - 2\nu^2 - 1) \frac{\partial^3 u}{\partial^3 x} + \text{H.O.T.}
\]  

(1.2)

Notice that the modified differential equation provides some insight into the behavior of the numerical scheme. On the right hand side of Equation 1.2 is a term that looks like physical diffusion in the Navier-Stokes equations, \( \frac{\partial^2 u}{\partial^2 x} \), with an artificial viscosity \( \frac{a \Delta x}{2} (1 - \nu) \). However, this term is of purely numerical origin and causes non-physical smearing of flow gradients in a CFD solution. This term introduces artificial viscosity into the numerical solution and is referred to as numerical dissipation [22]. The odd ordered derivative on the right hand side of Equation 1.2 is termed numerical
dispersion. Rather than cause solution smearing and nonphysical amplitude decay, this term distorts the propagation of different wave speeds. The combined effects of numerical dissipation and numerical dispersion is referred to as numerical diffusion. Thus, this simple example illustrates how numerical diffusion is introduced into the solution implicitly during the discretization of the governing PDE. Although numerical diffusion can reduce solution accuracy, there is a benefit of improved stability [21]. The artificial diffusion terms act as physical diffusion terms and help to smooth out any errors that may cause divergence of the solution.

Often authors will use the terms numerical diffusion and numerical dissipation interchangeably. In general, such synonymous use occurs when numerical dispersion effects are not the primary concern of the particular author's investigation. Such authors are generally concerned with solution smearing and both of the terms diffusion and dissipation describe this effect. As a result, in the present work, the terms numerical diffusion and dissipation may be used synonymously to describe artificial smoothing and dispersion effects are neglected.

As with the simple example of the discretization of the linear wave-equation, the majority of commercial CFD codes also employ first- or second-order upwinding of convective flux terms. Upwind discretization is generally preferred over central-differencing due to improved stability and robustness. Furthermore, central schemes generally require explicitly added artificial dissipation to become stable [22]. Also, unlike central schemes, upwind schemes attempt to account for the physical properties of the flow and one-way propagation of waves. For convection dominant flows, the
spatial domain may behave as a ‘one-way coordinate’ [23] and upwinding attempts to motivate the discretization based on flow physics. For practical applications, the first-order upwind scheme is overly dissipative due to the leading second-order truncation error and is often supplanted by the second-order upwind scheme. Second-order accuracy substantially reduces numerical dissipation but is still not adequate for tracking of vortices over large distances [24] such as the trailing vortex system behind a wing.

1.4 Counteracting Numerical Diffusion

As mentioned previously, there are several possible ways to combat numerical diffusion of the vortical wake of an aircraft in order to preserve the accuracy of wake-integral drag prediction. The most straightforward way is the reduce the magnitude of the truncation error associated with the discretization. This can be achieved by increasing the grid density in the wake region of the aircraft and/or using a higher-order discretization scheme. While effective, increasing the grid density or employing higher-order spatial discretization can incur a significant increase in computational overhead. The high-fidelity Euler CFD solution may then become intractable for aerodynamic design and may be supplanted by lower-fidelity vortex lattice or panel methods. The added computational cost of employing an Euler simulation may outweigh the less complete flowfield description of lower-fidelity models.

Another possible solution to overcome numerical diffusion effects is to place the downstream Trefftz plane very close to the aircraft. By doing so, the wake-integral
drag predictions are computed before numerical diffusion effects become significant. However, it is not clear how close the Trefftz plane should be positioned to the aircraft. If positioned too close, the trailing vortices may still be developing and gradients along the freestream direction may be significant. The computation of drag then requires many additional higher-order terms as an underlying assumption in the derivation of the wake-integral drag formulas is that the Trefftz plane is far enough downstream such that the flow is no longer varying in the freestream direction. Additionally, for complex aerodynamic configurations with several lifting surfaces, the question of where to position the Trefftz plane is further complicated. Complex geometries may produce several vortical wakes and the distance between the Trefftz plane and the location of initially shed vorticity may not be readily defined.

Also, another method to account for numerical diffusion effects in wake-integral drag prediction is applicable only for subsonic Euler flows. For such flows, the only physically relevant drag component is induced drag. Therefore, any entropy drag generated is almost entirely spurious. The spurious entropy drag arises due to numerical smoothing in regions of high flow gradients and inadequate grid resolution, such as near the leading of a wing and within vortical wake. It is then possible to obtain a more accurate prediction of the induced drag of the aircraft by neglecting the spurious entropy generated at the leading edge of the wing and lumping any spurious entropy drag generated downstream of the origin of the tip vortex with the induced drag prediction. This method is commonly referred to as the entropy correction technique [3, 8]. But "for transonic Euler calculations with shocks, and for Navier-
Stokes calculations with entropy generation in the boundary layer, it is much harder to distinguish between physically correct entropy generation and spurious numerical generation, so it may not be possible to apply such a correction" [8].

An alternative method to overcome the numerical dissipation of trailing vortices proposed in the present work is a method referred to as Vorticity Confinement (VC) [20]. Unlike more traditional techniques, the VC method attempts capture flow features via modeling rather than increasing resolution. Both grid refinement and employing higher-order discretization attempt to capture flow features such as trailing vortices by better resolving the governing PDE in the vicinity of the vortex. The VC method, however, assumes that if the vortex is small relative to the other scales of the flow, it is enough to model to the strength and position of the vortex. It is not necessary to accurately resolve the internal structure of the vortex. The VC method is accomplished by the addition of body force terms that provide rotational momentum forcing to the vortex to counteract the numerical dissipation of the CFD solver.

1.5 Vorticity Confinement Development and Application

Development and early implementation of the vorticity confinement methodology was conducted by John Steinhoff and associates at The University of Tennessee Space Institute at Tullahoma Tennessee. In [20, 25], the authors provide a detailed description of the initial formulation (known as VC1) for the vorticity confinement terms as well as application to simple two and three-dimensional analytical and numerical test
cases. For the continuum case, it is shown that while a convecting axisymmetric vortex will continually spread with time in the presence of diffusion, the addition of a confinement term allows the solution to relax to a steady solution of finite length scale determined by the confinement parameter [20]. Early numerical applications of vorticity confinement simulated three-dimensional flows such as the convection of a concentrated vortex in helicopter rotor flow [20] and the evolution of coaxial/coplanar vortex rings [25]. In general, the results of early numerical experiments illustrated the effectiveness of the vorticity confinement method in preserving concentrated vorticity even in the presence of diffusive numerical solvers. Additionally, in [20] several possibilities are presented for the specification of the auxiliary scalar field which serves as a basis for defining the centroid of vortical regions.

Dimensional analysis conducted in [24], shows the confinement parameter has units of a velocity. As a result, its specification is dependent on local flow conditions in addition to numerical inaccuracies of the specific flow solver under consideration. In general, the confinement parameter must be "tuned" on a case by case basis in order to achieve physically relevant solutions. Many researchers have investigated methods to make the specification of the confinement parameter more tractable and essentially automatic. The authors in [26] developed an expression for the confinement parameter based on the assumption that it must cancel the curl of the leading truncation error in regions of the flow-field with significant vorticity. The method requires an explicit estimate of numerical diffusion which the authors assumed to be the leading artificial viscosity terms added to make the solver stable. In a similar fashion, the authors
in [27] developed a expression for the confinement parameter based on numerical diffusion cancellation. This time, however, an estimate is made for the numerical viscosity by considering the difference between central and upwind discretizations of convective terms.

Rather than define the confinement parameter based on truncation error cancellation, the authors in [24] derived a general vorticity confinement term based on dimensional analysis. The original confinement parameter which carries units of a velocity is replaced with a function of local element size, vorticity, and the gradient of the absolute value of vorticity. The vorticity confinement formulation in [24] attempts to eliminate the grid dependency of the confinement parameter as with the truncation error cancellation methods, however, the method still requires the specification of a heuristic constant. This confinement parameter scaling has been successfully applied to numerical simulations of engineering significance such as the inviscid flow over a finite NACA0012 wing and viscous flow over a delta wing. These two test cases were also investigated in [28] with a constant confinement parameter and unstructured grids. The authors in [28] achieved mixed results tuning the confinement parameter to the particular grid with over-confinement (confinement parameter set to too large a value) producing non-physical results such as the generation of spurious vortical structures. For viscous flows, VC is often applied judiciously to the flow-field so as not to confine the boundary layer region [24, 28] where the grid resolution is generally adequate.
A fundamental issue many engineers have concerning the vorticity confinement method is that the original formulation, VC1, does not explicitly conserve momentum. As a result, Steinhoff [29] developed an alternative vorticity confinement formulation referred to as VC2. VC2 is an intrinsically discrete formulation which explicitly conserves momentum and has been analyzed in detail [29, 30]. Although the VC1 formulation does not explicitly conserve momentum, Steinhoff notes that for a wide range of flows with thin, concentrated regions of convecting vorticity, exact momentum conservation may not be as important as other features and should not be considered as an absolute requirement [31]. A similar argument is also made in [22] that the figure of merit of a computational scheme is not solely the conservative property and the importance of maintaining the conservative property is highly problem dependent.

In the present study, the second vorticity confinement formulation, VC2, is not investigated. The VC1 formulation has been successfully applied to a wide range of flows [20, 24, 25, 26, 27, 28, 31, 32] with no deleterious effects arising from its non-conservative nature. Additionally, the VC1 formulation has been more extensively investigated and applied to non-uniform, unstructured grids. As a result, the present work exclusively considers the VC1 formulation.

The vorticity confinement method has also been extended to compressible flows by accounting for work done by the vorticity confinement body force in the energy conservation equation [33, 34]. Additionally, some authors have investigated the addition of a density confinement term to the mass conservation equation in order
to alleviate the artificial dissipation of density [26]. The vast majority of applications of the compressible vorticity confinement method, however, only account for the work done by the vorticity confinement body force.

1.6 Research Goals

The main goal of this thesis is to show the efficiency of the wake-integral technique coupled with Vorticity Confinement for the prediction of drag from an Euler CFD simulation. Additionally, we would like to illustrate a lesser dependency on Trefftz plane location with the use of VC compared to the standard wake-integral technique. The wake-integral technique with VC could supplant the more common method of near-field integration as means of drag computation within the design cycle of an aircraft. It has already been demonstrated that drag obtained by the wake-integral technique is generally much more accurate than surface pressure integration for Euler flows [14]. However, the wake-integral technique suffers reduced accuracy due to the numerical dissipation of trailing vortices. While an entropy correction method has already been used to correct for the numerical diffusion of trailing vortices in an invisicid flow [3], the extension of this method to flows with physical sources of entropy may not be possible [8]. Thus, the novel aspect of this work is the use of Vorticity Confinement as a means to counteract numerical and preserve trailing vortices. While the VC method has previously been used to capture trailing vortices [24], it is has not be used in the context of improving wake-integral drag predictions.
This work begins with a detailed derivation and physical interpretation of the wake-integral drag predictions in CHAPTER 2. Also in CHAPTER 2 is a description of the Vorticity Confinement method and the form of the VC body force terms added to the governing conservation equations. CHAPTER 3 describes the computational flow solvers used in the numerical test cases and how VC body force terms are added to each. CHAPTER 4 includes a numerical validation test case of the Euler solver 2DFlow. In CHAPTER 5, numerical results of the advection of a two-dimensional vortex and the resultant impact of Vorticity Confinement are presented. The research code 2DFlow is used for this purpose as well as experimenting with alternative formulations of the VC body force terms. CHAPTER 6 presents computational results for subsonic Euler flow over a rectangular planform wing. The wake-integral technique is employed for drag prediction with VC used to preserve the trailing vortices of the wing. Induced drag predictions computed by the wake-integral method, near-field integration of surface pressure, and theoretical lifting-line theory are compared. CHAPTER 7 summarizes conclusions from the numerical test cases regarding the wake-integral technique for drag prediction and Vorticity Confinement method for vortex preservation.
2.1 Derivation of Wake-Integral Drag Formulas

In this section, the wake-integral drag formulations are derived. The equivalence of near-field and far-field surface integration techniques is first presented. Then, the more general far-field integral formulation is reduced to a wake-integral formulation following specific assumptions. A small perturbation technique is used to derive disparate wake-integral drag formulations which provide insight into the physical sources of drag.

2.1.1 Equivalence of Near-Field and Far-Field Methods

Let us first consider the equivalence of near-field and far-field surface integration techniques for a combined aerodynamic force, \( \vec{F}_a \), which includes both lift and drag forces \[35\]. Referring to Figure 2.1, the near-field drag integral is given by Equation 2.1 with the integration performed over the surface of the aircraft.

\[
\vec{F}_a = \int_{S_b} (-p\vec{n} + \vec{r} \cdot \vec{n}) \, dS_b \quad (2.1)
\]
In Equation 2.1, $\vec{n}$ is the outward normal vector of the aircraft and $S_b$ denotes the surface of the aircraft. The aerodynamic force in this near-field formulation is composed of pressure and viscous stresses and is the typical method for lift and drag computation from CFD simulation. The use of sting-balances essentially employ this same method for aerodynamic force computation. Although experiments generally do not directly apply Equation 2.1, the pressure and shear stress distributions on the aircraft give rise to an aerodynamic force which is detected by a strain gauges in the sting-balance.

Again referring to Figure 2.1, the far-field aerodynamic force can be computed by applying the integral form of momentum equation over a control volume surrounding the aircraft. The far-field formulation is given by Equation 2.2 with the integration performed over all the surfaces of the control volume.

$$\vec{F}_a = \int_S (-p\vec{n} - \rho(\vec{u} \cdot \vec{n})\vec{u} + \vec{\tau} \cdot \vec{n}) \, dS$$  \hspace{1cm} (2.2)
In Equation 2.2, $\vec{n}$ is the outward normal vector of each of the control surfaces comprising the control volume surrounding the aircraft. Equation 2.2 is equally applicable to any arbitrarily shaped control volume surrounding the aircraft. Equation 2.2 is the same as Equation 2.1 with an additional force contribution due to the momentum flux through the surfaces of the control volume, $\rho(\vec{u} \cdot \vec{n})\vec{u}$.

As no assumptions were made in their derivation, Equations 2.1 and 2.2 are equivalent forms for the computation of aerodynamic force. The near-field and far-field methods are simply the application of the integral form of the momentum equation applied to the body of the aircraft and a control volume surrounding the aircraft respectively. Substitution of Equation 2.1 into Equation 2.2 for the force, $\vec{F}_a$, results in Equation 2.3.

$$\int_{S_b} (-p\vec{n} + \vec{\tau} \cdot \vec{n}) \, dS_b = \int_{S} (-p\vec{n} - \rho(\vec{u} \cdot \vec{n})\vec{u} + \vec{\tau} \cdot \vec{n}) \, dS \quad (2.3)$$

2.1.2 Wake-Integral from Far-Field Integral

The far-field integral given by Equation 2.2 includes all of the surfaces of the control volume surrounding the aircraft. Applying several assumptions, the integration can be reduced to a single surface in the wake of the aircraft. The first assumption is that the rectangular control volume depicted by 2.1 is aligned with a Cartesian coordinate system. The coordinate system is defined by a x-coordinate aligned with the freestream flow direction, a y-coordinate aligned with the spanwise direction of the aircraft, and the z-coordinate aligned with the vertical direction. The second assumption is that
the upstream orthogonal plane of the control volume is far enough upstream such
the flow conditions at each location on the plane are equal to the freestream flow
conditions \((U_\infty, P_\infty, \rho_\infty)\). The third assumption is that the four planes of the control
volume parallel to the freestream flow direction are positioned infinitely far from the
aircraft such that there is no momentum flux through these surfaces and the pressure
is equal to the freestream value. The final assumption is that the surfaces of the
control volume are far enough from the aircraft such that the viscous stresses vanish.
Applying these assumptions the drag can be expressed by Equation 2.4.

\[
D = -\int_{\text{Wake}} (p - P_\infty) + (\rho u^2 - \rho_\infty U_\infty^2) \, dy \, dz \quad (2.4)
\]

Equation 2.4 can be cast in another form by making use of the conservation of mass,
Equation 2.5.

\[
\int_{\text{Upstream}} (\rho U_\infty) \, dy \, dz = \int_{\text{Wake}} (\rho u) \, dy \, dz \quad (2.5)
\]

Substituting Equation 2.5 into Equation 2.4 results in Equation 2.6.

\[
D = -\int_{\text{Wake}} (p - P_\infty) + \rho u (u - U_\infty) \, dy \, dz \quad (2.6)
\]

Equation 2.6 can be applied directly to CFD results by numerical approximation of
the integral. The formulation is particularly convenient in experimental applications
as the integrand only contains the deviation of flow quantities from their freestream
values \([35]\) and the flow quantities of pressure and velocity are readily measured.

22
However, the wake-integral formula given by Equation 2.6 does not provide any more physical insight to the sources of drag than the more commonly applied near-field wake-integral formulation presented in Equation 2.1. In order to further decompose the wake-integral drag of Equation 2.6 a perturbation technique [36] method is employed in the next section.

2.1.3 Perturbation Technique

Following the method outlined in [36], the downstream states in the wake are assumed to differ from the freestream values by small perturbations, Equation 2.7.

\begin{equation}
\begin{aligned}
\rho & = \rho_\infty + \Delta \rho \\
p & = P_\infty + \Delta p \\
\end{aligned}
\end{equation}

The \( \rho u \) term in Equation 2.6 is expanded with the introduction of the perturbation terms.

\begin{equation}
\rho u = \rho_\infty U_\infty + \rho_\infty \Delta u + U_\infty \Delta \rho + \Delta \rho \Delta u
\end{equation}

Additionally, making use of the definition of the speed of sound, the density perturbation can be related to the pressure perturbation.

\begin{equation}
\Delta \rho = \frac{\Delta p}{a_\infty^2}
\end{equation}
The pressure perturbation can be related to the velocity perturbation via the one-dimensional Euler Equation 2.10.

\[
\frac{u \, du}{dx} = -\frac{1}{\rho} \frac{dp}{dx} \tag{2.10}
\]

Approximating the differential elements with perturbations and substituting the perturbed quantities for \( u \) and \( \rho \) results in Equation 2.11.

\[
\Delta p = -\rho_\infty u_\infty \Delta u - \rho_\infty (\Delta u)^2 - u_\infty \Delta \rho \Delta u - \Delta \rho (\Delta u)^2 \approx 0 \tag{2.11}
\]

The quadratic perturbation terms in Equation 2.11 are neglected. Combining Equations 2.9 and 2.11 results in Equation 2.12.

\[
\Delta \rho = -\frac{\rho_\infty U_\infty \Delta u}{a^2_\infty} \tag{2.12}
\]

Substituting 2.8 into 2.6 and using 2.12 to eliminate the density perturbation terms results in 2.13.

\[
D = \int_{Wake} \rho_\infty U_\infty (U_\infty - u) + (P_\infty - p) - \rho_\infty [1 - M^2_\infty] (\Delta u)^2 + \frac{\rho_\infty u_\infty}{a^2_\infty} (\Delta u)^3 \, dy \, dz \tag{2.13}
\]
In order to later introduce entropy into the drag integral, enthalpy terms, \( h \) and \( h_\infty \), are added and subtracted to Equation 2.13 producing Equation 2.14.

\[
D = \int_{Wake} \rho_\infty (h - h_\infty) - \rho_\infty (h - h_\infty) \\
+ \rho_\infty U_\infty (U_\infty - u) + (P_\infty - p) \\
- \rho_\infty [1 - M_\infty^2] (\Delta u)^2 \\
+ \frac{\rho_\infty u_\infty}{a_\infty^2} (\Delta u)^3 \, dy \, dz 
\]

(2.14)

If the flow is considered to be steady and adiabatic, total enthalpy along equilibrium sections of a streamline remains constant [37]. The conservation of total enthalpy can be expressed by Equation 2.15.

\[
h + \frac{u^2 + v^2 + w^2}{2} = h_\infty + \frac{U_\infty^2}{2} 
\]

(2.15)

Rearranging Equation 2.15 results in Equation 2.16.

\[
h - h_\infty = \frac{1}{2} (U_\infty + u)(U_\infty - u) - \frac{1}{2} (v^2 + w^2) 
\]

(2.16)

Introducing the velocity perturbation results in Equation 2.17.

\[
h - h_\infty = U_\infty (U_\infty - u) - \frac{1}{2} (v^2 + w^2) - \frac{1}{2} (\Delta u)^2 
\]

(2.17)
Substituting Equation 2.17 into Equation 2.14 results in Equation 2.18.

\[
D = \int_{Wake} \rho_\infty (h - h_\infty) - (p - P_\infty)
+ \frac{\rho_\infty}{2} (v^2 + w^2)
- \frac{\rho_\infty}{2} [1 - 2M_\infty^2](\Delta u)^2
+ \frac{\rho_\infty u_\infty}{a_\infty^2} (\Delta u)^3 \, dy \, dz
\]  

(2.18)

For a small, reversible change in state, entropy may be expressed in terms of state variables by Equation 2.19 [37].

\[
\rho dh - dp = \rho T ds
\]  

(2.19)

If the entropy equation is written in terms of deviations from the freestream flow condition, the result is Equation 2.20.

\[
\rho_\infty (h - h_\infty) - (p - P_\infty) = \rho T ds
\]  

(2.20)
Substituting Equation 2.20 into Equation 2.17 results in Equation 2.21.

\[
D = \int_{Wake} \rho_\infty T_\infty (s - s_\infty) \, dy \, dz \\
+ \int_{Wake} \frac{\rho_\infty}{2} (v^2 + w^2) \, dy \, dz \\
- \int_{Wake} \frac{\rho_\infty}{2} [1 - 2M_\infty^2] (\Delta u)^2 \, dy \, dz \\
+ \int_{Wake} \frac{\rho_\infty u_\infty}{a_\infty^2} (\Delta u)^3 \, dy \, dz
\]

\begin{align*}
\text{(2.21)}
\end{align*}

Utilizing the equation of state for a perfect gas, \( P_\infty = \rho_\infty RT_\infty \), Equation 2.21 becomes Equation 2.22.

\[
D = \int_{Wake} \frac{P_\infty}{R} (s - s_\infty) \, dy \, dz \\
+ \int_{Wake} \frac{\rho_\infty}{2} (v^2 + w^2) \, dy \, dz \\
- \int_{Wake} \frac{\rho_\infty}{2} [1 - 2M_\infty^2] (\Delta u)^2 \, dy \, dz \\
+ \int_{Wake} \frac{\rho_\infty u_\infty}{a_\infty^2} (\Delta u)^3 \, dy \, dz
\]

\begin{align*}
\text{(2.22)}
\end{align*}
Neglecting the higher-order integrals (3 and 4), the first and second integrals of Equation 2.22 are commonly referred to as profile and induced drag respectively.

\[ D_p = \int_{Wake} \frac{P_\infty}{R} (s - s_\infty) \, dy \, dz \]  (2.23)

\[ D_i = \int_{Wake} \frac{\rho_\infty}{2} (v^2 + w^2) \, dy \, dz \]  (2.24)

### 2.1.4 Physical Interpretation of Wake-Integral Drag Formulas

A simple physical interpretation of the drag integrals, Equations 2.23 and 2.24, can be formulated in the context of work and energy principles. The aircraft is doing work on the surrounding air at the rate \( DU_\infty \), where \( D \) is the total drag. A portion of the work is recoverable as kinetic energy transferred to the wake crossflow. This is represented by the induced drag integral, Equation 2.23 [35]. The integrand of the induced drag integral is the crossflow kinetic energy, \( \frac{v^2 + w^2}{2} \), multiplied by the density in the freestream, \( \rho_\infty \). In other words, the aircraft performs reversible and recoverable work on the surrounding air at the rate \( D_i U_\infty \). Likewise, the rate of irreversible and unrecoverable work performed on the surrounding air is \( D_p U_\infty \). The profile drag, \( D_p \), arises from non-isentropic processes such as boundary-layer flow and flow through shock waves [37]. Thus, the profile drag is composed of both viscous drag \( D_v \) and wave drag, \( D_w \). Furthermore, in CFD simulations, numerical dissipation results in a non-physical component of the profile drag referred to as spurious drag, \( D_s \) [18]. It should be noted that the wake-integral formulations derived in the previous section provide the decomposition of drag into induced and profile components. Additional
decomposition of the profile drag into viscous, wave, and spurious components requires additional consideration an is not discussed presently. The decomposition of total drag into induced, viscous, wave, and spurious components is presented in 2.25.

\[
D = D_i + D_p
\]

\[
= D_i + D_v + D_w + D_s
\]

(2.25)

It is this ability to split drag contributions into induced and profile components that provides the wake-integral method a distinct advantage over the more traditional surface-integral method. The wake-integral method provides a physical decomposition of drag whereas the surface-integral method provides a mechanical decomposition of drag. Physical decomposition of drag refers to the direct computation of induced, wave, and/or viscous drag components whereas the mechanical decomposition of drag refers to the computation of drag through the integration of the mechanical forces arising from pressure and shear stress. As a result, the wake-integral method provides more physical insight into drag sources and can more effectively steer design changes [35].

\[
D = D_{\text{pressure}} + D_{\text{viscous}} = D_{\text{induced}} + D_{\text{profile}}
\]

(2.26)

For example, consider an aerodynamic configuration with a high level of induced drag. The wake-integral method directly provides a quantification of this
drag component via the integration of cross-flow kinetic energy in Equation 2.24. The surface-integral method, however, does not provide direct computation of the induced drag. The induced drag component is lumped within the pressure integral and cannot be separated from other contributions to the pressure integral such as wave drag. As a result, the wake-integral method’s ability to directly compute a high level of induced drag could be used in drag reduction through modification of the spanwise lift distribution of the aircraft [35].

2.2 The Vorticity Confinement Method

In this section, the Vorticity Confinement (VC) modification of the governing conservative equations is presented.

2.2.1 Basic Description

The basic goal of the VC method is to capture small-scale vortical structures and efficiently model them on an Eulerian computational grid [31]. The term small-scale refers to scales smaller than the main scales of the flow or vortices spanning approximately 1-3 grid cells. Rather than resolve these small-scale vortices, they are modeled through the interaction of viscous effects and nonlinear momentum forcing terms added to the conservative equations. In the case of inviscid CFD computations, non-physical diffusion implicit to the solver or added explicitly for stability could serve as the aforementioned viscous effects. The overall effect is that concentrated vortices which would normally diffuse rapidly can convect endlessly in the flowfield without
any spreading [31]. Furthermore, the vortices can be preserved on coarse grids with low-order discretization schemes.

The VC method is somewhat related to other methods involving thin structures such as shock and contact discontinuity capturing. As a result, it is often helpful to present the analogous features of these methods as they are familiar to CFD researchers [31].

Generally, shock capturing methods use a moderately sized inviscid grid in the region of the shock in order to capture the 'essential physics' of the shock. The 'essential physics' refer to features of the shock which impact the flowfield external to the shock. Additionally, the shock capturing methods preserve the conservative laws when integrated through the shock. The shock capturing schemes essentially 'model' rather than 'resolve' the internal structure of the shock. 'Resolution' of the shock would require a highly refined grid and more computationally expensive viscous computation. 'Modeling' of the internal structure is generally adequate provided that the shock remains thin and provides a significant reduction in computational time compared to 'resolution' of the shock [31].

In the VC method, this notion of shock 'modeling' vs. shock 'resolution' is readily extended to concentrated vortices. The vortices are modeled and thus captured over a small number of grid cells under the assumption that the internal structure of the vortices is not important. Given that the vortical structures are thin, it is only necessary that the 'essential physics' are captured. For a concentrated vortex, the 'essential physics' which must be preserved are the circulation of the
vortex and the location of the vortex centroid such that the induced flow external to
the vortex is accurately prescribed [31].

However, there is added difficulty in modeling vortices and other contact-like discontinuities as compared to shocks. This difference has both a physical and mathematical interpretation. Physically, shocks are naturally compressive and an equilibrium is rapidly achieved between compression and numerical spreading resulting from artificial diffusion [38]. Contact discontinuities are neither compressive or expansive and are highly sensitive to numerical diffusion. Numerical spreading causes the contact discontinuities to behave as if they were expansions with the contacts becoming wider and wider as the solution proceeds [38]. Mathematically, the difference between shock and contact discontinuities is described by the characteristics near the discontinuity. For shocks, the characteristics slope inward toward the shock and tend to steepen during a computation [31]. For contact discontinuities, numerical diffusion causes the characteristics to slope away from the discontinuity much like an expansion [38]. As a result, the VC method adds an additional "steepening" or "confinement" term to correct the artificial sloping of contact discontinuity characteristics away from the discontinuity [31].

2.2.2 Modification of the Governing Equations

The VC method is realized by the addition of a body force term to the momentum transport equation. For incompressible flows, this is the only modification that is necessary. For compressible flows, the work done by the VC body force is added to
the energy transport equation [39]. The integral form of the two-dimensional Euler
equations with the addition of the VC body force term is given by Equation 2.27 [34].

\[
\frac{d}{dt} \int_V \vec{W} \, dV + \int_S \vec{F} \cdot \vec{n} \, dS + \int_S \vec{G} \cdot \vec{n} \, dS = \int_V \vec{S} \, dV \tag{2.27}
\]

\[
\vec{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho e \end{pmatrix}, \quad \vec{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uh \end{pmatrix}, \quad \vec{G} = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vh \end{pmatrix}, \quad \vec{S} = \begin{pmatrix} 0 \\ \rho \vec{f}_b \cdot i \\ \rho \vec{f}_b \cdot j \\ \rho \vec{f}_b \cdot \vec{u} \end{pmatrix}
\]

The formulation for the body force term, denoted as VC1, is given by Equation 2.28.

\[
\vec{f}_b = -\epsilon \hat{n} \times \vec{\omega}
\]

\[
\hat{n} = \frac{\nabla |\vec{\omega}|}{|\nabla |\vec{\omega}||}
\]

The VC body force term, $\vec{f}_b$, is composed of the cross product of the normalized vorticity gradient vector, $\hat{n}$, and the vorticity vector, $\vec{\omega}$. The strength of the confinement is controlled by the heuristic 'confinement parameter', $\epsilon$ [34].

There are several important features to note concerning the VC formulation presented in Equation 2.28. First, the VC source term is intrinsically multidimensional and rotationally invariant [31]. As a result, arbitrary vortical structures are readily supported within the VC framework. Also, it is clear that the VC body force
term vanishes away from flow regions with significant vorticity and the conservative equations revert to their unmodified form. Thus, the effect of the VC body force term is limited to regions of the flow with large gradients and therefore significant numerical diffusion. Another comment concerns the behavior of the VC body force term in regions of the flow with significant levels of vorticity.

In the formulation presented in Equation 2.28, the thin vortical structures are essentially 'flagged' through the specification of an auxiliary scalar field and a normal direction is defined which points along the gradient of this scalar field. In this case, the auxiliary scalar field is $|\vec{\omega}|$ and the vector $\hat{n}$ points from the region of lower vorticity magnitude to the region of higher vorticity magnitude or toward the vortex center. As the VC source term involves the vector cross product of $\hat{n}$ and the local value of vorticity, the VC body force term provides tangential momentum forcing to counteract numerical diffusion. This form of the VC body force term also illustrates how the VC method bridges the gap between Lagrangian methods such as point vortex methods with prescribed vortices and typical Eulerian methods which quickly diffuse concentrated vortical structures.

The VC method "allows concentrated vortical regions to be convected with essentially no numerical diffusion even on relatively coarse grids, as in Lagrangian marker-based methods" [20]. However, the VC method does not require the a priori specification of markers to define vortical regions and thus can accommodate arbitrary vorticity distributions with no difficulty. Rather than utilize explicit markers, the VC method uses an auxiliary scalar field, as mentioned above, which is readily computed
on the Eulerian grid in order to flag the vortical structures. Furthermore, the VC method also parallels Lagrangian marker-based methods in the assumption that the internal structure of the vortical regions does not need to be resolved in detail [20].

Alternatively, the VC body force term may be considered as a convection term that convects vorticity back toward the vortex center. Numerical diffusion causes vorticity to diffuse away from the vortex center and the VC body force term provides a countering convective force. Eventually the convection-diffusion will become nearly balanced resulting in a confined vortical structure that can convect through the flowfield without spreading. Considering a local coordinate system convecting along with the vortex, the numerical diffusion and vorticity convection realized by the VC body force term reach a steady-state solution [20]. The authors in [20] showed analytically that a convecting axisymmetric vortex can achieve a steady-state form in the presence of diffusion given the addition of the VC body force term. Furthermore, the length scale of the steady-state form is characterized by the ratio of viscosity, $\mu$, and the confinement parameter, $\epsilon$.

2.2.3 Dimensional Analysis of Confinement Parameter

A fundamental issue in the implementation of the VC method is the specification of the heuristic confinement parameter, $\epsilon$. In early applications, the VC method was used on uniform Cartesian grids with the confinement parameter set to a uniform value [20, 25]. However, in extending the VC method to non-uniform, unstructured grids, some measure must be taken to scale the confinement parameter. One ap-
proach for scaling the confinement parameter is based on dimensional analysis of the
conservative momentum equation and supplants the specification of \( \epsilon \) with a true
constant, \( c_{vc} \) [24].

Considering the integral form of the momentum conservation equation in the
\( x \)-direction for an inviscid fluid,

\[
\frac{d}{dt} \int_V (\rho u) \, dV + \int_S (\rho u^2 + p) \cdot \vec{n} \, dS + \int_S (\rho uv) \cdot \vec{n} \, dS = -\int_V \left( \rho \epsilon \frac{\nabla |\vec{\omega}|}{|\nabla |\vec{\omega}|} \times \vec{\omega} \right) \cdot i \, dV
\]

Substituting the units kilograms, meters, and seconds into Equation 2.29, it is clear
that the confinement parameter, \( \epsilon \), must have units of a velocity in order for the VC
body force term to be consistent with the other terms in the equation.

\[
\frac{d}{dt} \int_V (\text{kg m}^{-2} s^{-1}) \, dV + \int_S (\text{kg m}^{-1} s^{-2} + \text{kg m}^{-1} s^{-2}) \cdot \vec{n} \, dS + \cdots = -\int_V \left( \epsilon \text{kg m}^{-3} s^{-1} \right) \cdot i \, dV
\]

\[
m^3 s^{-1} (\text{kg m}^{-2} s^{-1}) + m^2 (\text{kg m}^{-1} s^{-2} + \text{kg m}^{-1} s^{-2}) + m^2 (\text{kg m}^{-1} s^{-2}) = \cdots
\]

\[
\cdots = -m^3 \epsilon \left( \text{kg m}^{-3} s^{-1} \right)
\]

\[
\epsilon \propto O \left( m s^{-1} \right)
\]

36
The authors in [24] chose to replace the confinement parameter, $\epsilon$, with a true constant, $c_{vc}$ multiplied by a length scale squared, $h^2$, and the gradient of vorticity magnitude, $\nabla |\vec{\omega}|$.

$$\epsilon \propto c_{vc} h^2 \nabla |\vec{\omega}|$$

(2.33)

Unlike $\epsilon$, $c_{vc}$ does not carry any units. For isotropic grids, the choice of characteristic length scale, $h$, is rather arbitrary producing similar results for the average of edge-lengths surrounding a point or volume to surface ratio of control volumes surrounding a point [24]. For highly stretched grids, the authors in [24] suggest using a characteristic length taken along the same direction as the gradient of vorticity magnitude. In the present work, the computational grids are primarily isotropic and the particular choice of characteristic length scale should be of little consequence.

It is also worth noting that the scaled confinement parameter formulation ensures the consistency of the VC source term. As the mesh gets progressively finer, $h \to 0$, the VC body force term vanishes. This behavior is desireable if the VC body force term is considered as a numerical diffusion cancellation term. As the mesh is refined, the truncation error and numerical diffusion introduced by the discretization diminish. Likewise, the magnitude of the scaled confinement parameter and thus the correction introduced by the VC body force term is reduced. As a matter of fact, for a first-order discretization scheme, the leading truncation error is $O(h^2)$ and responds to changes in grid size as the scaled confinement parameter formulation in Equation 2.33.
Thus, if the mesh becomes sufficiently refined or if the flowfield lacks sufficient vorticity, the governing conservative equations revert to their original form without any VC modification.

Although the formulation presented in Equation 2.33 does provide some means of scaling the original confinement parameter, $\epsilon$, it is still necessary to specify a heuristic constant, $c_{vc}$. However, once $c_{vc}$ is specified, the additional weighting of the scaled confinement formulation in terms of $h^2$ and $\nabla |\vec{\omega}|$ should alleviate some of the trouble in specifying the confinement parameter accompanying changes made to the grid or flow conditions. This conclusion is partially confirmed by the computational results of [24].

The authors in [24] utilized the scaled confinement formulation to improve vortex preservation in two computational cases with different grids and physics. In both cases, the VC method is used to improve vortex capture over the standard second-order flow solver and $c_{vc}$ is kept at a constant value of 0.1. The first case involves the preservation of trailing vortices shed by a rectangular NACA0012 wing in an inviscid flowfield. The second case involves capturing the detached vortex resulting from the viscous computation of a delta wing at high angle of attack. Despite having different grids and flow physics (inviscid vs. viscous), the same confinement coefficient $c_{vc}$ could be used between the two cases producing physically relevant solutions and improved vortex preservation over the standard second-order solver.
CHAPTER III

CFD CODE DESCRIPTIONS

The code 2DFlow has been developed by the author to check the efficiency and feasibility of VC for a 2-D convective vortex in simple geometric set-up while the commercially available CFD code ANSYS FLUENT is used to investigate feasibility of the VC method for 3-D wings.

3.1 2DFlow–General Description and Implementation of VC

2DFlow is a two-dimensional compressible Euler flow solver written in Fortran 95 during the course of the current research. The stuctured finite volume code solves for conservative variables, \( \vec{W} \) (see Equation 2.27) with first-order explicit time stepping presented in Equation 3.1 [40].

\[
\vec{W}_{i+1} = \vec{W}_i + \frac{\Delta t}{\Omega} \vec{R}
\]

(3.1)

The unsteady Euler equations are hyperbolic and admit discontinuous solutions such as shocks. As such, the Steger-Warming flux-vector splitting scheme is employed. This scheme accounts for the eigenvalue structure of the convective flux
jacobian by splitting flux contributions into positive and negative components. The overall theme of Steger-Warming as well as other flux-vector splitting schemes is the proper specification of control volume boundary fluxes [22]. Once split-fluxes are determined at cell faces, a first-order upwind scheme is employed to compute the residual. Consider the two-dimensional Euler equations in conservative differential form given by Equation 3.2.

\[
\frac{\partial \vec{W}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial x} = 0 \tag{3.2}
\]

\[
\vec{W} = \begin{pmatrix} 
\rho \\
\rho u \\
\rho v \\
\rho e 
\end{pmatrix} \quad \vec{F} = \begin{pmatrix} 
\rho u \\
\rho u^2 + p \\
\rho u v \\
\rho u h 
\end{pmatrix} \quad \vec{G} = \begin{pmatrix} 
\rho v \\
\rho u v \\
\rho v^2 + p \\
\rho v h 
\end{pmatrix}
\]

The residual can be expressed as in Equation 3.3.

\[
\vec{R} = -\frac{\partial \vec{F}}{\partial x} - \frac{\partial \vec{G}}{\partial x} \tag{3.3}
\]
The fluxes are split into positive and negative components based on the eigenvector structure of the flux jacobians $[A]$ and $[B]$ defined in Equation 3.4. The flux jacobians are split into positive and negative components and the split fluxes are defined as in Equation 3.5 [22].

$$
\vec{F} = [A] \vec{W}
$$

$$
\vec{G} = [B] \vec{W}
$$

$$
\vec{F} = \vec{F}^+ + \vec{F}^- = [A^+] \vec{W} + [A^-] \vec{W}
$$

$$
\vec{G} = \vec{G}^+ + \vec{G}^- = [B^+] \vec{W} + [B^-] \vec{W}
$$

The residual is written in split-flux form as Equation 3.6 [22].

$$
\vec{R} = -\frac{\partial \vec{F}^+}{\partial x} - \frac{\partial \vec{F}^-}{\partial x} - \frac{\partial \vec{G}^+}{\partial x} - \frac{\partial \vec{G}^-}{\partial x}
$$

(3.6)

Applying first-order upwinding to the split-fluxes, the residual becomes Equation 3.7. The fluxes are upwind based on whether they are positive or negative flux contributions. Positive flux contributions are backward differenced and negative flux contributions are forward differenced [22].

$$
\vec{R} = -\frac{1}{\Delta x} \left( \left[ \vec{F}_{i+1}^- - \vec{F}_{i-1}^+ \right] + \left[ \vec{F}_{i+1}^- - \vec{F}_{i}^- \right] \right) - \frac{1}{\Delta y} \left( \left[ \vec{G}_{j+1}^+ - \vec{G}_{j-1}^+ \right] + \left[ \vec{G}_{j+1}^- - \vec{G}_{j}^- \right] \right)
$$

(3.7)

In treating the boundaries, the use of dummy cells as detailed in [40] is adopted. Dummy cells are simply additional layers of computational cells added to
the outside of the physical domain of interest which ease the computation of fluxes and gradients along the physical boundaries. Normally, the computational stencil of the boundary cells would extend beyond the physical domain and the boundaries would require special treatment. Using dummy cells makes it possible to lump the computation of physical boundary cells within the same loop as the computation of the rest of the interior cells. The values within the dummy cells are continually updated as the solution advances by extrapolating from the interior [40].

In 2DFlow the computation and addition of the VC source terms is achieved by the subroutine CONFINEMENT.F. Following the computation of the conservative variables at the next time step, the confinement subroutine is called to compute the VC source terms and add them to the conservative variables. In formulating the VC source terms, \( \vec{\omega} \) and \( \nabla | \vec{\omega} | \) are computed with central-differences as in Equations 3.8 and 3.9 respectively.

\[
\vec{\omega} = \left( \frac{v_{i+1,j} - v_{i-1,j}}{2\Delta x} - \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} \right) \vec{k} \tag{3.8}
\]

\[
\nabla | \vec{\omega} | = \left( \frac{| \vec{\omega} |_{i+1,j} - | \vec{\omega} |_{i-1,j}}{2\Delta x} \right) \vec{i} + \left( \frac{| \vec{\omega} |_{i,j+1} - | \vec{\omega} |_{i,j-1}}{2\Delta y} \right) \vec{j} \tag{3.9}
\]

3.2 ANSYS FLUENT

ANSYS FLUENT is a general-purpose commercial flow solver that provides modeling capabilities for a wide range of fluid flow problems. FLUENT can perform incompressible/compressible, inviscid/laminar/turbulent, and steady-state/transient anal-
yses. In addition, FLUENT supports multiphase flows, sliding meshes, Lagrangian particle tracking, and boasts a host of additional features [1].

For all fluid flow problems FLUENT solves conservative equations for mass and momentum. For compressible flows, an additional energy equation is solved with the ideal gas equation used to couple pressure and density variations [1].

3.2.1 General Description

At this stage, a number of ANSYS FLUENT solver parameters will be discussed. The chosen parameters apply specifically to modeling the external flow over a 3-D wing at Mach 0.3 and an angle of attack of 4°.

Although the freestream Mach number is low enough such that variations in density are small, the fluid is modeled as compressible. Additionally, viscous effects are neglected. The resulting conservation equations for mass, momentum, and energy (Euler equations) solved by FLUENT are given by Equations 3.10, 3.11, and 3.12 respectively [1].

\[
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = S_m \quad (3.10)
\]

\[
\frac{\partial}{\partial t} (\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \vec{v}) = -\nabla p + \rho \vec{g} + \vec{F} \quad (3.11)
\]

\[
\frac{\partial}{\partial t} (\rho E) + \nabla \cdot (\vec{v} (\rho E + p)) = -\nabla \cdot \left( \sum_{j} h_j J_j \right) + S_h \quad (3.12)
\]

The flow is also modeled as steady-state and the pressure-based solver is used. The pressure-based solver belongs to the more general class of solvers which ensure mass conservation of the velocity field by solving a pressure correction equation. More
specifically, the SIMPLE algorithm is used to enforce mass conservation and to obtain
the pressure field. Given that the flow is considered steady-state, the transient terms
in the Euler equations are dropped and the system is linearized and solved using
Gauss-Seidel in conjunction with algebraic multigrid (AMG) [1].

In order to compute the convective terms in the governing equations, it is
necessary to interpolate values at the cell-centers to the cell faces. This is achieved
with an upwind discretization scheme. Specifically, the second-order upwind scheme
is used for density, momentum, and energy terms. When the second-order upwind
scheme is used, the face value of an arbitrary scalar quantity, \( \phi_f \) may be expressed
by Equation 3.13 [1].

\[
\phi_f = \phi + \nabla \phi \cdot \vec{r} \tag{3.13}
\]

In Equation 3.13 \( \phi \) is the cell-centered value of an arbitrary scalar quantity and \( \nabla \phi \)
is its gradient in the upstream cell. \( \vec{r} \) is the displacement vector directed from the
upstream cell centroid to the face centroid. The gradient \( \nabla \phi \) is computed using a least
squares cell-based approach. In the least squares cell-based approach, the solution
is assumed to vary linearly between cell-centers. The change in cell values between
adjacent cells \( c_0 \) and \( c_i \) along the vector \( \delta r_i \) which points the cell-centroid of cell \( c_0 \)
to cell \( c_1 \) may be expressed as in Equation 3.14 [1].

\[
(\nabla \phi) \cdot \Delta r_i = (\phi_{ci} - \phi_{c0}) \tag{3.14}
\]
Writing similar equations for the remaining cells surrounding cell \( c_0 \) results in the system expressed in Equation 3.15. This system is then solved as a least-squares minimization problem for \((\nabla \phi)_{c_0}\) [1].

\[
[J] (\nabla \phi)_{c_0} = \Delta \phi
\] (3.15)

3.2.2 Implementation of VC

The VC body force term is added to the core ANSYS FLUENT solver through the use of user-defined functions (UDF). UDF’s are functions "that can be dynamically loaded with the ANSYS FLUENT solver to enhance the standard features of the code" [41]. They are commonly used to define boundary conditions, material properties, source terms, as well as initialize a solution, and enhance post-processing [41]. The UDF’s are basically subroutines written in the C programming language that are linked and executed with the ANSYS FLUENT solver. Additionally, the UDF’s can make use of predefined macros and functions supplied by ANSYS FLUENT to ease specific operations such as calculating gradients and accessing solver data [41].

In order to compute the VC source terms, it is necessary to form terms such as \(\nabla |\vec{\omega}|\). However, \(\vec{\omega}\) is not a variable that ANSYS FLUENT normally carries during the solution and is generally only available once the numerical simulation has been completed. As a result, terms such as \(|\vec{\omega}|\) are specified as user-defined-scalars (UDS) and gradient operations are performed on the user-defined-scalars making use of the ANSYS FLUENT macro C_UDSI_RG(c,t,VORTICITY) where \(c\) is the cell, \(t\) is
is the cell thread (grouping of cells), and VORTICITY is the name of a previously
defined UDS. RG denotes that this macro is a reconstruction gradient which employs
limiters. The standard gradient macro does not employ limiters an may introduce new
maxima/minima. As discussed above, gradients are computed using a least-squares
cell-based approach [41].

The specification of the user-defined-scalars is nested within the DEFINE_ADJUST
macro which is a general-purpose macro used to adjust ANSYS FLUENT variables
that are not passed as arguments. The source terms are specified using the DE-
FINE_SOURCE macro [41].
CHAPTER IV

NUMERICAL TEST CASES

4.1 Sod Shock Tube Problem

The Sod Shock Tube Problem [42] has been used as a one-dimensional test of the accuracy of the compressible CFD solver 2DFlow. The problem consists of a shock tube with a diaphragm which separates two regions which have different densities and pressures. Both fluids on either side of the diaphragm are initially at rest. The diaphragm is then broken which leads to the development of three different characteristic waves. A shock wave propagates into the low pressure side of the tube and an expansion or rarefaction wave propagates into the low pressure side. A contact surface marks the boundary between the fluids which were initially on either side of the diaphragm [37]. Unlike a shock wave, which exhibits discontinuities in all quantities ($\rho, u, e, p$), a contact discontinuity is characterized by jumps in $\rho$ and $e$. 
The initial conditions of the shock tube problem for the left and right states are given by Equation 4.1 in normalized quantities [42]. The diaphragm is initially located in the center of the domain which corresponds to a value of 0.5 along the x-axis in Figure 4.1.

\[
\begin{align*}
\left\{ \begin{array}{c}
\rho \\
p \\
u
\end{array} \right\}_{\text{left}} &= \left\{ \begin{array}{c}
1.0 \\
1.0 \\
0.0
\end{array} \right\}, & \left\{ \begin{array}{c}
\rho \\
p \\
u
\end{array} \right\}_{\text{right}} &= \left\{ \begin{array}{c}
0.125 \\
0.1 \\
0.0
\end{array} \right\}
\end{align*}
\] (4.1)

The shock tube was simulated with 2DFlow with a spatial step \( h = 0.01 \) and CFL = 0.8. Plots of normalized density, pressure, and velocity along the horizontal centerline of the domain are depicted in 4.1. The results in Figure 4.1 illustrate 2DFlow’s ability to simulate the complex nonlinear wave interactions which develop in a shock tube and correctly satisfy the Rankine-Hugoniot relations [37]. The code accurately predicts the locations of head and tail of the rarefaction wave, the contact discontinuity, and the shock wave. The code is first-order accurate in both space and time which can introduce significant numerical dissipation into the solution. This is apparent in the rounding of sharp discontinuities at the head and tail of the rarefaction wave as well as the contact discontinuity. However, the shock wave discontinuity appears to be less smeared compared to the other discontinuities.
Figure 4.1: Sod Shock Tube Results, Normalized Plots of Density, Pressure, and Velocity, $T = 0.00636 \text{s}$. 
CHAPTER V

TWO-DIMENSIONAL VORTICITY CONFINEMENT

The simple test case of an 2-D advecting vortex has been used to assess the accuracy of 2DFlow and allows for visualization of numerical diffusion errors. The advecting vortex is particularly appealing as an analytical solution is available. Additionally, the vortex advection problem serves as a good test case to illustrate the effectiveness of the Vorticity Confinement (VC) method in preserving a two-dimensional, isolated vortex. Furthermore, the 2-D advecting vortex is commonly used by Vorticity Confinement researchers for parameter tuning and developing alternative formulations [20, 26, 27, 30, 43]

5.1 Problem Description

The 2-D advecting vortex is prescribed by perturbing an initially uniform flowfield. The uniform freestream conditions are: \( u_\infty \) and \( v_\infty \). Perturbations (Equation 5.1) are superimposed on the freestream velocity to create the vortex: \( u = u_\infty + \delta u \), \( v = v_\infty + \delta v \) [44].

\[
\begin{align*}
\delta u &= -(y - y_{ctr}) \frac{\beta}{2\pi} \exp\left(\frac{1 - r^2}{2}\right) \\
\delta v &= (x - x_{ctr}) \frac{\beta}{2\pi} \exp\left(\frac{1 - r^2}{2}\right)
\end{align*}
\] (5.1)
In Equation 5.1, $\beta$ represents the strength of the vortex and $r = \sqrt{(x - x_{ctr})^2 + (y - y_{ctr})^2}$ is the radius relative to the vortex center $(x_{ctr}, y_{ctr})$. A nice feature of this vortex is the perturbed quantities superimposed on the flowfield are smooth. Additionally, the flow does not exhibit any shocks or contact discontinuities. Thus, the primary goal of the advecting vortex test case is to illustrate the numerical diffusion errors of the computational solver and to serve as a test bed for the VC method. This is in contrast to the shock tube problem in which the specific goal was to test the solver’s ability to capture flow discontinuities. The analytical solution for the 2-D advecting vortex is simply the initial solution shifted by the distance covered by the advection velocity, $(u, v) = (u_\infty t, v_\infty t)$.

The relevant physical parameters for the specification of the vortex are summarized in Table 5.1. The working fluid is considered to be air at standard conditions. For $\beta = 50$, the core radius of the vortex, $R_c$, is approximately 1m with a peak velocity perturbation of approximately 7.96$\text{m s}^{-1}$. The physical domain is rectangular extending $50R_c$ in the x-direction and $20R_c$ in the y-direction with the initial vortex center located at $(10R_c, 10R_c)$. For the conditions specified in Table 5.1, the local Mach number remains approximately 0.3 and compressibility effects are relatively small.

In addition to the physical parameters described above, there are a number of numerical parameters that must also be specified. The relevant computational parameters are summarized in Table 5.2.
### Physical Parameter Value

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_\infty$</td>
<td>100 m s$^{-1}$</td>
</tr>
<tr>
<td>$v_\infty$</td>
<td>0 m s$^{-1}$</td>
</tr>
<tr>
<td>$p_\infty$</td>
<td>101325 Pa</td>
</tr>
<tr>
<td>$\rho_\infty$</td>
<td>1.225 kg m$^{-3}$</td>
</tr>
<tr>
<td>$R$</td>
<td>287 J kg$^{-2}$ K$^{-1}$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>1.4</td>
</tr>
<tr>
<td>$\beta$</td>
<td>50</td>
</tr>
<tr>
<td>$R_c$</td>
<td>1 m</td>
</tr>
<tr>
<td>$x_{\text{cntr}}$</td>
<td>10$R_c$</td>
</tr>
<tr>
<td>$y_{\text{cntr}}$</td>
<td>10$R_c$</td>
</tr>
</tbody>
</table>

Table 5.1: Advecting Vortex Physical Parameters

### Computational Parameter Value (units)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta x$</td>
<td>0.1$R_c$</td>
</tr>
<tr>
<td>$\Delta y$</td>
<td>0.1$R_c$</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.8</td>
</tr>
<tr>
<td>$</td>
<td>\omega</td>
</tr>
</tbody>
</table>

Table 5.2: Advecting Vortex Computational Parameters

In Table 5.2, the $\sigma$ is the CFL number used in the determination of the maximum time step. For the two-dimensional structured Euler code 2-DFlow, the time step is determined from Equation 5.2 [40].

$$\Delta t_I = \sigma \frac{\Omega_I}{(\Lambda_c^I + \Lambda_c^J)}$$  \hspace{1cm} (5.2)

In Equation 5.2, $\Delta t_I$ is the time step for cell $I$, $\Omega_I$ is the cell volume (cell area times unit depth for 2-D), and $\Lambda_c^I$ and $\Lambda_c^J$ are the spectral radii of the convective flux.
Jacobians in the $I$ and $J$ directions respectively. The spectral radii are determined by Equation 5.3 [40].

\[
\Lambda_c^I = \left( |\vec{u} \cdot \vec{n}^I| + c \right) \Delta S^I \\
\Lambda_c^J = \left( |\vec{u} \cdot \vec{n}^J| + c \right) \Delta S^J
\]

(5.3)

The normal vectors, $\vec{n}^I$ and $\vec{n}^J$, and face areas, $\Delta S^I$ and $\Delta S^J$, in Equation 5.3 are approximated by averaging the corresponding values from the two opposite sides of the control volume in the respective direction. The global time step is determined from the minimum time step for all control volumes, $\Delta t = \min I \Delta t^I$ [40].

An important practical consideration regarding the implementation of Vorticity Confinement is the use of limiters. Limiters are required to ensure that the VC source terms are added only to regions of the flowfield of interest. Without limiters, small spurious vortices can be strengthened by VC and degrade the solution. A limiter based on $|\nabla|\omega||$ was successfully employed by the authors in [34]. In the present work, a limiter based on $|\omega|$ is used. Basically, confinement is limited to regions of the flowfield in which the vorticity magnitude is at least some percentage of the maximum vorticity magnitude. The baseline value used for the vorticity limiter is 0.1 or 10%. Thus, only regions of the flowfield with $|\omega| > 0.1 |\omega_{max}|$ are confined.
5.2 Computational Results with No Confinement

Computational results for the advecting vortex with no confinement can be seen in Figures 5.1 and 5.2. Theoretically, the vortex should convect unaltered through the domain. However, it is clear in Figure 5.1b that after $250\Delta t$ the vortex is visibly weaker and by $500\Delta t$ (Figure 5.1c) has nearly disappeared. As the numerical flow solver 2DFlow is first-order in both space and time, it introduces significant numerical diffusion into the solution and results in smearing and weakening of the vortex. The effect of numerical diffusion is also supported by the reduced concentration of vorticity isolines near the vortex core in Figures 5.2b and 5.2c at $250\Delta t$ and $500\Delta t$ respectively.

5.3 Computational Results with Constant Confinement Parameter

The following are computational results for the vortex advection test case with Vorticity Confinement (VC) and constant confinement parameter, $\epsilon$. The confinement source terms are added exclusively to the governing momentum conservation equations with a constant confinement parameter, $\epsilon = 25$. Also, the vorticity limiter in Table 5.2 is applied. Both vorticity and vorticity gradients necessary to form the VC source terms are computed by central-differences. The VC method is implemented by a separate "confinement" subroutine called each time step to re-compute the VC source terms and provide the additive momentum/velocity correction.
As with the no confinement case above, results are presented in the form of plots of the velocity vector field and isolines of vorticity. The vector field depicted in Figure 5.3c shows improved vortex preservation over the no confinement case depicted in Figure 5.1c. The improved vortex preservation through the addition of VC source terms is reaffirmed by the higher concentration of vorticity isolines near the vortex core in Figure 5.4c.

While the temporal preservation of the vortex is improved with VC and $\epsilon = 25$, the choice of $\epsilon$ was rather arbitrary and is most likely not optimal for the particular grid and flow conditions under consideration. In fact, choosing $\epsilon = 100$ further improved vortex capture. The goal here is simply to illustrate the quanlitative impact of the addition of vorticity confinement source terms on the numerical solution.

The results with $\epsilon = 100$ are depicted in Figures 5.5 and 5.6 in the form of velocity vector field plots and isolines of vorticity respectively. After $500\Delta t$, the vortex is both strengthened and confined over a smaller region compared to the no confinement case, compare Figures 5.2c and 5.6c.

Results for the advecting vortex test case with constant confinement are summarized in Figure 5.7 in profiles of velocity and vorticity through the vortex center. In Figure 5.7a, the initial velocity profile is significantly smoothed by the flow solver after $500\Delta t$. The peak velocity change is approximately 77% less than the initial peak velocity. However, with confinement, the peak velocity change for $\epsilon = 25$ and $\epsilon = 100$ is approximately $-62\%$ and $-54\%$ respectively. In addition to better preserving the peak velocity of the vortex, VC also has a gradient steepening effect in both the
velocity and vorticity profiles. In Figure 5.7a, confinement increases the velocity gradient at the vortex center with a larger confinement parameter producing a steeper gradient. This is also true of the vorticity profile in Figure 5.7b. Accompanying the steepening of the vorticity gradient is also a reduction in the 'spread' of the vortex. For $\epsilon = 100$, the peak vorticity at the vortex center is actually increased beyond that of the initial vortex. However, the vortex is confined to a smaller number of cells and the large values of vorticity are spread over less area. Profiles of velocity and vorticity provide substantial insight into the effect of VC within the vortex. However, in regard to the computation of drag and characterizing the overall strength of a vortex, circulation is of greater interest. Circulation can be computed by the line integral of velocity or the surface integral of vorticity, see Equation 5.4 [4].

$$\Gamma = \oint \vec{u} \cdot d\ell = \int \vec{\omega} \cdot dS \quad (5.4)$$

For a two-dimensional flow, the discrete equation for vorticity can be written as in Equation 5.5.

$$\Gamma = \sum \omega_i A_i \quad (5.5)$$

In Equation 5.5, $\omega_i$ is the cell-center value of vorticity and $A_i$ is the area of the cell. Considering that the initial vortex has a core radius of approximately 1 m, the summation in Equation 5.5 is limited to this region. It is only this core region of the vortex that has significant vorticity and limiting the region of integration reduces additional computational overhead. The temporal evolution of the circulation is summarized
in Figure 5.8. After $500\Delta t$, the computed vortex circulation is significantly reduced in the no confinement case. The percent change in circulation from the initial value of approximately $50m^2 s^{-1}$ is approximately $-79\%$ with no confinement (at $500\Delta t$). When the VC source terms are added with constant confinement, the percent change in circulation from the initial vortex is approximately $-61\%$ for both $\epsilon = 25$ and $\epsilon = 100$. If the induced drag is considered to be directly proportional to the circulation [4] and the two-dimensional advecting vortex is considered to represent the trailing vortex of a wing, Figure 5.8 suggests that the addition of VC source terms may improve the induced drag prediction by $18\%$.

5.4 Computational Results with Compressibility Effects

Thus far, the implementations of VC have been incompressible. The work done by the VC body force terms has been assumed small and neglected from the energy equation. In this section, this assumption is examined in more detail.

Results for the advecting vortex with compressible confinement and $\epsilon = 100$ are depicted in Figures 5.9 and 5.10. From these results, it does not appear that accounting for the work done by the VC body force in the energy equation produces significant changes within the vortex. The velocity vectors (Figure 5.9c) and isolines of vorticity (Figure 5.10c) after $500\Delta t$ are indistinguishable from the incompressible VC case with $\epsilon = 100$ depicted in Figures 5.5c and 5.6c respectively.

For the low-speed subsonic flow of the advecting vortex, it is not necessary to add the work done by VC source terms to the energy equation. The local flow
Mach number remains approximately 0.3, a value at which compressible effects may be considered negligible.

5.5 Computational Results – Coarse Grid with VC vs. Refined Grid

A significant advantage of the VC method is that vortices can be captured on relatively coarse grids using low-order discretization schemes that would normally be overly diffusive. As a result, it seems appropriate to compare results on a coarse grid with VC and results on a more refined grid without VC. Such a comparison is made in the present section.

Computational results for circulation and computer runtime are summarized for three different cases in Figure 5.11. Circulation is represented by bars and is scaled along the primary vertical axis on the left. Computer runtime in seconds is represented by points connected by lines and is scaled along the secondary vertical axis on the right. Case 1 is the 'coarse' grid \((h = 0.1R_c)\) with no VC. Case 2 is the 'coarse' grid with VC and constant \(\epsilon = 100\). Case 3 is a 'refined' grid \((h = 0.05R_c)\) with no VC. The values for circulation are taken at 500\(\Delta t\) for the coarse grid and 1000\(\Delta t\) for the refined grid. Different \(\Delta t\) are compared in order to approximately match the physical time. Furthermore, in order to match the total physical time, the computer runtime is 3000\(\Delta t\) for the coarse grid and 6000\(\Delta t\) for the refined grid. Computer runtimes correspond to a linux workstation with dual 2.4Ghz Intel Xenon Processors and 4Gb of memory.
In Figure 5.11 going from Case 1 to Case 2 there is a 79% increase in the computed circulation and a 11% increase in computational overhead. Going from Case 1 to Case 3 there is a 103% increase in the computed circulation and 389% increase in computational overhead. Although the computed circulation on the refined grid is slightly improved over the coarse grid with VC, the computational overhead is increased by nearly a factor of four. These results illustrate the potential of the VC method to significantly improve vortex preservation at a fraction of the computational cost of techniques which employ more conventional 'brute force' methods like grid refinement.
Figure 5.1: Velocity vectors: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with no confinement.
Figure 5.2: Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with no confinement ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)
Figure 5.3: Velocity vectors: (a) $0 \Delta t$, (b) $250 \Delta t$, and (c) $500 \Delta t$ with $\epsilon = 25$
Figure 5.4: Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with $\epsilon = 25$ ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)
Figure 5.5: Velocity vectors: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with $\epsilon = 100$
Figure 5.6: Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$ with $\epsilon = 100$ ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)
Figure 5.7: Profiles of velocity and vorticity through vortex center: $0\Delta t$, $500\Delta t$ with no VC, $500\Delta t$ with $\epsilon = 25$, and $500\Delta t$ with $\epsilon = 100$
Figure 5.8: Temporal evolution of two-dimensional advecting vortex circulation (no VC, $\epsilon = 25$, and $\epsilon = 100$)
Figure 5.9: Velocity vectors: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$, compressible confinement with $\epsilon = 100$
Figure 5.10: Isolines of vorticity: (a) $0\Delta t$, (b) $250\Delta t$, and (c) $500\Delta t$, compressible confinement with $\epsilon = 100$ ($\omega$ varying from $-10$ to $50$, $\Delta \omega = 2.5$)
Figure 5.11: Comparison of circulation and computation time, 2-D advecting vortex
In this chapter, the 3-D Vorticity Confinement method is applied to the subsonic flow over a rectangular planform wing with NACA 0012 profile. VC is implemented to counteract numerical dissipation of trailing vortices and improve prediction of induced drag via wake-integration.

6.1 Theoretical Induced Drag

The rectangular planform wing with NACA 0012 profile is a good test case for the drag prediction from CFD as the induced drag can be approximated by Prandtl’s classical lifting-line theory. In lifting-line theory, the finite wing is assumed to be thin and is modeled as an infinite number horseshoe vortices whose bound vortices coincide along a single line called the lifting line. Under these assumptions and making use of the Biot-Savart Law to express the induced velocity of a vortex, Prandtl developed a theoretical prediction for induced drag coefficient given by Equation 6.1 [4].

\[ C_{D,i} = \frac{C_L^2}{\pi e AR} \]  

(6.1)
In Equation 6.1, $C_{D,i}$ in the induced drag coefficient, $C_L$ is the lift coefficient, $e$ is the span efficiency factor that accounts for non-elliptic circulation distribution, and $AR$ is the aspect ratio. Thin airfoil theory predicts that the lift coefficient is linearly proportional to the angle of attack $\alpha$ and may be expressed as $c_L = 2\pi \alpha$ where $2\pi \text{rad}^{-1}$ is a measure of the lift slope. However, for a finite wing the effective angle of attack is less than geometric angle of attack and the lift slope is reduced. Thus the lift coefficient of a finite wing may be expressed as Equation 6.2 [4].

$$C_L = \left( \frac{a_0}{1 + (a_0/\pi AR) (1 + \tau)} \right) \alpha$$  \hspace{1cm} (6.2)

In Equation 6.2, $a_0$ is the airfoil lift slope and $\tau$ is a factor that accounts for the finite nature of the wing. The aspect ratio is a function of the geometry of the wing and is given by Equation 6.3 for a rectangular planform wing [4].

$$AR = \frac{b^2}{S} = \frac{b^2}{bc} = \frac{b}{c}$$  \hspace{1cm} (6.3)

In Equation 6.3 $b$ is the span, $S$ is the planform area, and $c$ is the chord length.

For the rectangular planform wing considered presently, $AR \approx 6.67$, $e \approx 0.94$, $\tau = 0.065$ [4], and $\alpha = 4^\circ$, and the theoretical lift coefficient and theoretical induced drag coefficient are approximately 0.3325 and 0.0056 respectively.
6.2 Computational Geometry and Mesh

The rectangular planform wing has a chord length $c = 1\text{m}$ and aspect ratio $AR = 6.67$ and no aerodynamic twist. Additionally, the definition of the NACA 0012 profile is slightly altered to ensure that the wing has a sharp trailing edge to ease grid generation. The three-dimensional computational model consists of rectangular planform wing positioned within a rectangular plenum with walls located 10$c$ from the wing in all directions, see Figure 6.1. A more computationally efficient half-span model was initially used, however, it was found during the course of research that the VC source terms interfered with ANSYS FLUENT’s symmetric boundary condition. Rather than remedy the problem by eliminating VC near the boundary through the use of cuttoffs, the symmetric boundary was removed by employing a full-span model.

Two different computational grids are used with the rectangular wing model. Both grids are composed exclusively of tetrahedral cells but with varying degrees of grid refinement. The first grid, denoted Grid 1, has a grid spacing of 0.06$c$ on the wing surface and along the approximate location of the trailing vortices in the wake. The increased density along the paths of the trailing vortices is achieved by extending line sources from the trailing edge of the wing downstream along the freestream flow direction. Additionally, the a priori estimate for the grid size is based on the maximum thickness of airfoil, which is 0.12$c$. Assuming that the trailing vortices span at least the maximum thickness of the wing, a grid spacing of 0.06$c$ will ensure that there are at least two grid cells across the vortex. In order to have any hope of capturing
the trailing vortices with VC, there must be at least a few grid cells spanning the vortex. The total cell count for Grid 1 is 300,942 and the wing surface grid is visible in Figure 6.2a and the increased grid density in the wake is visible in Figure 6.2b.

The second grid, Grid 2, is similar to Grid 1 but with increased grid refinement near the leading and trailing edges of the wing. The increased grid refinement is necessary to suppress spurious entropy generation. In a CFD simulation, spurious entropy generation generally occurs in regions of significant flow gradients such as near the leading edge of a wing and within shed vortical structures. In the present work, a goal of the Vorticity Confinement method is to control the generation of spurious entropy in the wake. As a result, it is desirable to reduce any spurious entropy generation occurring upstream of the wake as this will increase the sensitivity of wake-integral entropy drag predictions to changes incurred by Vorticity Confinement. The total cell count for Grid 2 is 781,078 and the wing surface grid and grid density in the wake are visible in Figures 6.3a and 6.3b respectively.

6.3 Physical and Computational Parameters

The pertinent physical parameters and computational models for the specification of the rectangular wing simulation are summarized in Tables 6.1 and 6.2 respectively. ANSYS FLUENT is the numerical solver used and additional details regarding the computational models are discussed in section 3.2.
Physical Parameter Value

<table>
<thead>
<tr>
<th>Physical Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_p$</td>
<td>1006.43 J kg$^{-1}$ K$^{-1}$</td>
</tr>
<tr>
<td>$MW$</td>
<td>28.966 kg kgmol$^{-1}$</td>
</tr>
<tr>
<td>$M_\infty$</td>
<td>0.3</td>
</tr>
<tr>
<td>$p_\infty$</td>
<td>101325 Pa</td>
</tr>
<tr>
<td>$T_\infty$</td>
<td>300 K</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>4$^\circ$</td>
</tr>
</tbody>
</table>

Table 6.1: Rectangular Planform Wing Physical Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solver</td>
<td>Pressure-Based</td>
</tr>
<tr>
<td>Solver Algorithm</td>
<td>Gauss-Seidel with AMG</td>
</tr>
<tr>
<td>Pressure-Velocity Coupling</td>
<td>SIMPLE</td>
</tr>
<tr>
<td>Pressure Interpolation</td>
<td>Standard</td>
</tr>
<tr>
<td>Energy Equation</td>
<td>On, Compressible</td>
</tr>
<tr>
<td>Viscous Model</td>
<td>Inviscid</td>
</tr>
<tr>
<td>Temporal Discretization</td>
<td>Steady-State</td>
</tr>
<tr>
<td>Spatial Discretization</td>
<td>Second-Order Upwind</td>
</tr>
<tr>
<td>Gradients</td>
<td>Least Squares Cell-Based</td>
</tr>
</tbody>
</table>

Table 6.2: Rectangular Planform Wing Computational Models [1]

6.4 Computational Results

In this section, computational results of the rectangular planform wing simulations are presented. Both no confinement and scaled confinement cases are compared directly. Comparison focuses on the trailing vortices and the resulting impact of trailing vortex preservation on wake-integral drag predictions. As the grid is unstructured, the scaled confinement parameter formulation presented in section 2.2.3 is employed to better capture changes in computational cell size. Additionally, the VC body force terms
are not activated until the trailing vortices have convected $0.25c$ downstream of the trailing edge in order to reduce interference with vortex roll-up. For this subsonic flow, the VC source terms neglected from the energy equation.

6.4.1 Visualizing Trailing Vortex

The impact of Vorticity Confinement is readily observed by examining the vorticity distribution within the wing’s trailing vortex. Figure 6.4 depicts vorticity isolines within the trailing vortex at varying Trefftz plane locations downstream of the wing. When the wing is simulated without confinement, the trailing vortex quickly diffuses as it is convected downstream as indicated by the reduced concentration of vorticity isolines in Figures 6.4a - 6.4c. In contrast, the addition of VC body force terms produces noticeable improvement in the vortex preservation as the trailing vortex convects downstream, Figures 6.4d - 6.4f. In addition to maintaining vortex strength, VC also serves to concentrate the vortex over just a few grid cells. This result is indicative that VC provides a framework for vortex preservation based on implicit modeling rather than resolution.

Without confinement, the vortex preservation is dependent upon the accurate resolution of the governing PDE’s in the vicinity of the vortex. Considering just truncation error, this accurate resolution occurs as $h \to 0$ or the computational grid is sufficiently refined. In the present case (Figures 6.4a - 6.4c), the grid spacing across the trailing vortex is not sufficient enough to resolve the vortex accurately and significant numerical dissipation is introduced into the solution. The numerical
dissipation has a cumulative effect and the solution smearing increases as the trailing vortex convects downstream.

The VC method, however, serves to implicitly model the vortex rather than resolve it. Implicit modeling suggests that VC preserves the vortex strength and location without accurate resolution of the governing PDE’s in the immediate vicinity of the vortex. Thus, within the vortex, the actually resolved PDE’s may deviate substantially from the original governing PDE’s due to the addition of the VC body force terms. The resulting internal structure of the vortex may then be physically irrelevant. However, given that the scale of the vortex is much smaller than the other length scales of the flow, the internal structure of the vortex should not have a significant influence on the remaining flowfield and is considered unimportant. Rather only the strength and position of the vortex are considered important and are modeled [25]. In the present case (Figures 6.4d - 6.4f), it is apparent that VC is modeling rather than resolving the internal structure of the vortex. The modeled trailing vortex almost behaves as a concentrated point vortex commonly employed in Lagrangian methods. Thus, the VC method provides a means to achieve Lagrangian type results in an Eulerian framework [25].

Isolines of vorticity within the trailing vortex are presented for the more refined grid, Grid 2, in Figure 6.5. As the increased grid refinement is limited to the leading edge of the wing, the behavior of the trailing vortex is similar to Grid 1 in Figure 6.4 and the same comments made above apply.
6.4.2 Wake-Integral Drag

It is clear that Vorticity Confinement has a marked effect on the preservation of the wing’s trailing vortices, but the quantitative impact on wake-integral drag predictions is of more interest. In Figure 6.6 the wake-integral predictions of induced drag are plotted at various Trefftz planes positioned downstream of the wing. The first-order computation of induced drag, Figure 6.6a, involves the integral of cross-flow kinetic energy given by Equation 2.24 whereas the second-order computation, Figure 6.6b, includes the additional $(\Delta u)^2$ contribution given in Equation 2.22.

For both Grid 1 and the more refined Grid 2 and no confinement, the wake-integral induced drag coefficient decays as the Trefftz plane location moves downstream (square symbols in Figures 6.6a and 6.6b). As the trailing vortices move downstream they are smeared by numerical dissipation and result in decreasing induced drag predictions. However, VC preservation of the trailing vortices increases the induced drag predictions far downstream of the wing and improves the overall precision of the drag computations. Thus, VC lessens the wake-integral induced drag prediction dependency on Trefftz plane location. In Figure 6.6a, the linear trendline becomes nearly horizontal following the application Vorticity Confinement.

In Figure 6.6b, however, the linear trendlines indicate increasing values $C_{Di}$ following the application of VC as the Trefftz plane position moves farther downstream. This behavior may indicate that the confinement parameter is not optimal and is actually strengthening the trailing vortices as they are convected downstream. The specification of the scaled confinement parameter, $c_{vc}$, is achieved through trial and
error. Barring a sufficient number of iterations, \( c_{vc} \), may result in over- or under-confinement of the trailing vortices. However, the isolines of vorticity presented in Figures 6.4 and 6.5 do not explicitly indicate over-confinement of the trailing vortices. Additionally, this behavior may indicate that the VC source terms have been activated too close to the trailing edge of the wing or that additional higher-order terms of \( \Delta u \) should be included in the computation of induced drag when the Trefftz plane is located near the trailing edge. In any case, the use of the second-order correction is only necessary immediately downstream of the wing and the present application of VC is to facilitate placement of the Trefftz plane far downstream. Therefore, the impact of the second-order correction on the linear trendline is neglected and the remaining conclusions made about the \( C_{Di} \) following the application of VC are limited to the first-order computation in Figure 6.6a.

The improved precision of the induced drag predictions with the addition of VC body force terms can be characterized by the root mean square deviation (RMSD) or standard error. If the lifting-line induced drag coefficient \( C_{Di} = 0.0056 \) is considered the 'standard' value, the formula for RMSD [45] is given by Equation 6.4 and the resulting values for the first-order wake-integral drag predictions are summarized in Table 6.3.

\[
\text{RMSD} = \sqrt{\frac{\sum_{j} (C_{Di,wake-integral} - C_{Di,lifting-line})^2}{n}}
\]  

(6.4)
In Equation 6.4, the summation is performed over all the Trefftz planes in the wake.

In the present case there are 8 Trefftz planes located from 1 to 8 chord lengths downstream of the wing’s trailing edge.

<table>
<thead>
<tr>
<th>Case Description</th>
<th>RMSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid 1, No VC</td>
<td>0.00127</td>
</tr>
<tr>
<td>Grid 1, Scaled VC, $c_{vc} = 0.075$</td>
<td>0.00075</td>
</tr>
<tr>
<td>Grid 2, No VC</td>
<td>0.00100</td>
</tr>
<tr>
<td>Grid 2, Scaled VC, $c_{vc} = 0.075$</td>
<td>0.00039</td>
</tr>
</tbody>
</table>

Table 6.3: Rectangular planform wing root mean square deviation (RMSD) or standard error of wake-integral induced drag coefficient

In Table 6.3, following the application of VC, the standard error reduces by a approximately a factor of two for both Grid 1 and Grid 2. The improved precision of wake-integral induced drag predictions with Vorticity Confinement is a direct consequence of the improved preservation of the wing’s trailing vortices.

In addition to improving the precision of wake-integral induced drag predictions, VC also improves the accuracy. Even without confinement, the wake-integral drag coefficients are generally more accurate than the more common near-field integration of surface pressure. For Grid 1, the near-field predicted drag coefficient $C_D = 0.0199$ over-predicts the lifting-line induced drag coefficient by 143 drag counts (1 drag count = 0.0001). Increasing the grid density near the leading edge of the wing provides some improvement in the near-field computed drag coefficient as for Grid 2, the near-field $C_D = 0.0103$, 40 drag counts larger than the lifting-line $C_{Di}$. However, the wake-integral $C_{Di}$’s computed at 5 chord lengths downstream are 0.0043
and 0.0045 for Grid 1 and Grid 2 respectively. Both under-predict the lifting-line $C_{Di}$ by 13 or fewer drag counts and are more accurate than the near-field predicted values. With VC, the induced drag coefficients increase to 0.0048 and 0.0052 for Grid 1 and Grid 2 respectively, within 8 or fewer drag counts of lifting-line $C_{Di}$.

The physical decomposition of wake-integral predicted drag into induced and entropy components is a distinct advantage over the near-field integration of surface pressure. The near-field method lumps any spuriously generated entropy within the pressure integral which explains why such methods typically over-predict induced drag. Furthermore, as the entropy is lumped within the pressure integral, the near-field method is much more sensitive near-surface grid resolution compared to the wake-integral method. A reduction in spurious entropy generation by increased near-surface grid density manifests itself in lower wake-integral entropy drag while the wake-integral induced drag remains fairly constant. Such a reduction in wake-integral entropy drag following increased near-surface grid resolution is apparent in Figure 6.7. Going from Figure 6.7a for Grid 1 to Figure 6.7b for Grid 2, the increased grid resolution near the wing surface suppresses the overall level of entropy drag in the wake by approximately 100 drag counts.

However, the significant feature of Figure 6.7 is the behavior of the wake-integral entropy drag coefficient predictions following the application of VC. Without confinement, the entropy drag coefficient generally increases as the Trefftz plane position moves farther downstream. This is indicative of the transfer from induced to entropy drag accompanying the numerical dissipation of the wing’s trailing vortices.
However, VC preserves the trailing vortices and helps counteract such behavior. VC causes the trend in wake-integral entropy drag to 'level off' and become nearly invariant with Trefftz plane location. This is true for Grid 2 as the linear curve-fit of entropy drag coefficients with confinement vs. Trefftz plane location in Figure 6.7b is nearly horizontal. For Grid 1, the linear curve-fit (Figure 6.7a) suggests that the entropy drag coefficient is decreasing as the Trefftz plane position moves downstream. However, this is due to an outlying entropy drag coefficient value at 1 chord length downstream and is skewing the overall trend in the data. Removing this outlier, Grid 1 entropy drag results would exhibit similar invariance to Trefftz plane location following the application of VC as seen for Grid 2.
Figure 6.1: Rectangular planform wing and computational plenum

Figure 6.2: Computational Grid 1: (a) wing surface, (b) wake region
Figure 6.3: Computational Grid 2: (a) wing surface, (b) wake region
Figure 6.4: Rectangular planform wing trailing vortex at varying Trefftz plane locations, Grid 1, isolines of vorticity ($|\omega|$ varying from 0 to 300, $\Delta|\omega| = 15$): (a)-(c) No Confinement, (d)-(f) Scaled Confinement $c_{sc} = 0.075$
Figure 6.5: Rectangular planform wing trailing vortex at varying Trefftz plane locations, Grid 2, isolines of vorticity ($|\omega|$ varying from 0 to 300, $\Delta|\omega| = 15$): (a)-(c) No Confinement, (d)-(f) Scaled Confinement $c_{vc} = 0.075$
Figure 6.6: Rectangular planform wing wake-integral induced drag coefficient at various Trefftz plane locations in chords downstream of trailing edge, linear curve fitting: (a) First-Order, (b) Second-Order, lifting-line $C_{Di} = 0.0056$

Figure 6.7: Rectangular planform wing wake-integral entropy drag coefficient at various Trefftz plane locations in chords downstream of trailing edge, linear curve fitting: (a) Grid 1, (b) Grid 2, lifting-line $C_{Di} = 0.0056$
CHAPTER VII

CONCLUSIONS AND FURTHER WORK

This work has illustrated the use of the wake-integral technique coupled with Vorticity Confinement as an efficient method of drag prediction from an Euler CFD simulation. The commercial CFD code ANSYS FLUENT was used to simulate a thin wing with VC body force terms added to the solver with a UDF and wake-integral drag quantities computed with the ANSYS FLUENT post-processor. The induced drag prediction of the rectangular planform wing showed improvement (more closely approximated theoretical lifting-line) over the standard wake-integral technique without confinement and more significant improvement over traditional surface integration of pressure. Furthermore, VC essentially eliminated the induced drag prediction dependency on the downstream location of the Trefftz plane and simultaneously suppressed the level of spurious entropy drag in the wake. The preservation of the induced drag integral eliminated the shift from induced to spurious entropy drag as the trailing vortices were convected downstream of the wing. Results from the wing simulation were published in [46].

Both the improved drag prediction and visual inspection of the wing trailing vortex supported that VC with the scaled confinement parameter of [24] performed as desired. Moreover, the assumption that the trailing vortices were 'small' such
they could be modeled rather than resolved appeared to be reasonable for the planar vortical wake shed by the wing. The application of VC resulted in trailing vortices that appeared almost as point vortices typical of Lagrangian type methods and with no apparent detriment to the surrounding flowfield. Additionally, as VC was employed to improve wake-integral drag prediction, any subtle changes to the flowfield incurred by modeling rather than resolving the trailing vortices were likely smoothed by the integration.

Prior to the application of the VC method in the preservation of the trailing vortex of a wing, the method was verified for a two-dimensional vortex advection test case. The vortex advection test case served as both a validation case for the research Euler CFD solver 2DFlow and as a test bed for the implementation of Vorticity Confinement. The grid for the advecting vortex test case was cartesian, therefore a constant confinement parameter was employed as in early applications of the VC method [20, 25]. Vortex strength was shown to be significantly improved and the vortex core confined to a smaller number of computational cells following the application of VC. Visual inspection of the improved preservation of the advecting vortex was also supported by the temporal evolution of vortex circulation.

A significant feature of the VC method highlighted with the two-dimensional advecting vortex was the modest increase in computational runtime accompanying the addition of VC body force terms to the computational solver. With only a marginal increase in computational overhead, the preservation of vortex circulation was much improved with confinement. By contrast, the use of a more refined grid to
better resolve the vortex required significantly more computer runtime to achieve approximately the same degree of vortex preservation. Such results regarding computer runtime could readily be extrapolated to three-dimensions and the preservation of wingtip trailing vortices. Therefore, it could be concluded that the modest computational cost of VC to counteract numerical diffusion does not impose any restriction on the prediction of drag with a higher-fidelity Euler CFD simulation in aerodynamic design.

Partially addressed in this work was the specification the heuristic confinement parameter, $\epsilon$, which controls the level of vortex confinement. In order to adapt to changes in the grid cell size of the unstructured grid of the rectangular planform wing, the scaled confinement parameter formulation of [24] was employed. However, as with the constant confinement parameter, appropriate specification of the scaled confinement parameter remains somewhat empirical. Further work involves the development of a more automatic specification of the confinement parameter, possibly based on an explicit numerical diffusion estimate. Additionally, this research avenue partially motivated the development of the research code 2DFlow such that numerical solution, which influences the specification of $\epsilon$, could be closely controlled.

In addition to developing a less heuristic method to specify the confinement parameter, future work would be the application of the wake-integral technique for drag prediction coupled with VC to flows with physical sources of entropy drag. For the subsonic Euler flow in the present work, the wake-integral drag predictions could readily be corrected for the numerical diffusion of the trailing vortices by the afor-
mentioned entropy correction method. While the extension of the entropy correction method to flows with physical sources of entropy may not be possible, the same is not true of VC. Induced drag predictions with VC would not be contaminated by the presence of physical sources of entropy as may be the case with entropy correction.
BIBLIOGRAPHY


APPENDICES
#include "udf.h"

enum
{
  w1,
  w2,
  w3,
  mag_w,
  grad_mag_w_x,
  grad_mag_w_y,
  grad_mag_w_z,
  mag_grad_w,
  h,
  eps,
  x_mom,
  y_mom,
  z_mom
};

DEFINE_ADJUST(adjust_fcn, domain)
{
Thread *t;
cell_t c;
int n;
Node *node;
face_t f;
real x;
real y;
real z;
real xmax;
real ymax;
real zmax;
real xmin;
real ymin;
real zmin;

thread_loop_c(t, domain)
{
begin_c_loop(c, t)
{
C_UDSI(c, t, w1)=C_W_RG(c, t)[1]-C_V_RG(c, t)[2];
\[ C_{\text{UDSI}}(c,t,w2) = C_{\text{U RG}}(c,t)[2] - C_{\text{W RG}}(c,t)[0]; \]
\[ C_{\text{UDSI}}(c,t,w3) = C_{\text{V RG}}(c,t)[0] - C_{\text{U RG}}(c,t)[1]; \]

\}

end\_c\_loop(c,t)

}\}

thread\_loop\_c(t,domain)
{
begin\_c\_loop (c,t)
{
\[
C_{\text{UDSI}}(c,t,\text{mag}_w) = \sqrt{C_{\text{UDSI}}(c,t,w1) \times C_{\text{UDSI}}(c,t,w1) + C_{\text{UDSI}}(c,t,w2) \times C_{\text{UDSI}}(c,t,w2) + C_{\text{UDSI}}(c,t,w3) \times C_{\text{UDSI}}(c,t,w3)};
\]
}

end\_c\_loop(c,t)

}\}

thread\_loop\_c(t,domain)
{
begin\_c\_loop (c,t)
{
\[
C_{\text{UDSI}}(c,t,\text{grad} \_\text{mag}_w \_x) = C_{\text{UDSI G}}(c,t,\text{mag}_w)[0];
\]
\[
C_{\text{UDSI}}(c,t,\text{grad} \_\text{mag}_w \_y) = C_{\text{UDSI G}}(c,t,\text{mag}_w)[1];
\]
\[
C_{\text{UDSI}}(c,t,\text{grad} \_\text{mag}_w \_z) = C_{\text{UDSI G}}(c,t,\text{mag}_w)[2];
\]
}
end_c_loop(c,t)
}

thread_loop_c(t,domain)
{
  begin_c_loop(c,t)
  {
    C_UDSI(c,t,mag_grad_w)=
sqrt(C_UDSI(c,t,grad_mag_w_x)*C_UDSI(c,t,grad_mag_w_x))
+ C_UDSI(c,t,grad_mag_w_y)*C_UDSI(c,t,grad_mag_w_y)
+ C_UDSI(c,t,grad_mag_w_z)*C_UDSI(c,t,grad_mag_w_z));
  }
  end_c_loop(c,t)
}

thread_loop_c(t,domain)
{
  begin_c_loop(c,t)
  {
    c_node_loop(c,t,n)
    {
      node = C_NODE(c,t,n);
      x = NODE_X(node);
      y = NODE_Y(node);
      z = NODE_Z(node);
if (x > -1E06 && n == 1)
{
    xmax = x;
}

if (x > xmax && n != 1)
{
    xmax = x;
}

if (x < 1E06 && n == 1)
{
    xmin = x;
}

if (x < xmin && n != 1)
{
    xmin = x;
}

if (y > -1E06 && n == 1)
{
    ymax = y;
}

if (y > ymax && n != 1)
{
    ymax = y;
if (y < 1E06 && n == 1) {
    ymin = y;
}
if (y < ymin && n != 1) {
    ymin = y;
}
if (z > -1E06 && n == 1) {
    zmax = z;
}
if (z > zmax && n != 1) {
    zmax = z;
}
if (z < 1E06 && n == 1) {
    zmin = z;
}
if (z < zmin && n != 1) {
    zmin = z;
}
zmin = z;
}
}
}
C_UDSI(c,t,h) = sqrt((xmax-xmin)*(xmax-xmin)
+(ymax-ymin)*(ymax-ymin)
+(zmax-zmin)*(zmax-zmin));
}
end_c_loop(c,t)
}

thread_loop_c(t,domain)
{
begin_c_loop(c,t)
{
C_UDSI(c,t,eps)=
0.075*C_UDSI(c,t,h)*C_UDSI(c,t,h)*C_UDSI(c,t,mag_grad_w);
}
end_c_loop(c,t)
}

thread_loop_c(t,domain)
{
begin_c_loop(c,t)
{
{C_UDSI(c,t,x_mom)=

103
(0.075*C_UDSI(c,t,h)*C_UDSI(c,t,h)*C_UDSI(c,t,mag_grad_w)*C_R(c,t))*((C_UDSI(c,t,grad_mag_w_y)*C_UDSI(c,t,w3)/C_UDSI(c,t,mag_grad_w))-(C_UDSI(c,t,grad_mag_w_z)*C_UDSI(c,t,w2)/C_UDSI(c,t,mag_grad_w))); C_UDSI(c,t,y_mom)= 
(0.075*C_UDSI(c,t,h)*C_UDSI(c,t,h)*C_UDSI(c,t,mag_grad_w)*C_R(c,t))*((C_UDSI(c,t,grad_mag_w_z)*C_UDSI(c,t,w1)/C_UDSI(c,t,mag_grad_w))-(C_UDSI(c,t,grad_mag_w_x)*C_UDSI(c,t,w3)/C_UDSI(c,t,mag_grad_w))); C_UDSI(c,t,z_mom)= 
(0.075*C_UDSI(c,t,h)*C_UDSI(c,t,h)*C_UDSI(c,t,mag_grad_w)*C_R(c,t))*((C_UDSI(c,t,grad_mag_w_x)*C_UDSI(c,t,w2)/C_UDSI(c,t,mag_grad_w))-(C_UDSI(c,t,grad_mag_w_y)*C_UDSI(c,t,w1)/C_UDSI(c,t,mag_grad_w))); }
end_c_loop(c,t)

#define EPS 0.075
DEFINE_SOURCE(xmom_source, c, t, dS, eqn)
{
real source;
real xc[ND_ND];
C_CENTROID(xc,c,t);

if (xc[0] > 1.25)
{

source =

(EPS*C_UDSI(c,t,h)*C_UDSI(c,t,h)*C_UDSI(c,t,mag_grad_w)
*C_R(c,t))*((C_UDSI(c,t,grad_mag_w_y)*C_UDSI(c,t,w3)
/C_UDSI(c,t,mag_grad_w))-(C_UDSI(c,t,grad_mag_w_z)
*C_UDSI(c,t,w2)/C_UDSI(c,t,mag_grad_w)));

dS[eqn]=0;
}
else

source = dS[eqn] = 0.;

return source;
}

DEFINE_SOURCE(ymom_source,c,t,dS,eqn)
{
}

real source;
real xc[ND_ND];

C_CENTROID(xc,c,t);

if (xc[0] > 1.25)
{

source =

(EPS*C_UDSI(c,t,h)*C_UDSI(c,t,h)*C_UDSI(c,t,mag_grad_w)
\[ *C_R(c,t)*((C_{UDSI}(c,t,\text{grad} \_\text{mag}_w_z)*C_{UDSI}(c,t,w1)
\quad /C_{UDSI}(c,t,\text{mag}_\text{grad}_w)-(C_{UDSI}(c,t,\text{grad} \_\text{mag}_w_x)
\quad *C_{UDSI}(c,t,w3)/C_{UDSI}(c,t,\text{mag}_\text{grad}_w)));\]

dS[eqn]=0;
}

else

source = dS[eqn] = 0.;
return source;
}

DEFINE_SOURCE(zmom_source,c,t,dS,eqn)
{
real source;
real xc[ND\_ND];
C\_CENTROID(xc,c,t);
if (xc[0] > 1.25)
{
source =
(EPS*C_{UDSI}(c,h)*C_{UDSI}(c,h)*C_{UDSI}(c,t,\text{mag}_\text{grad}_w)
\quad *C_R(c,t))*((C_{UDSI}(c,t,\text{grad} \_\text{mag}_w_x)*C_{UDSI}(c,t,w2)
\quad /C_{UDSI}(c,t,\text{mag}_\text{grad}_w)-(C_{UDSI}(c,t,\text{grad} \_\text{mag}_w_y)
\quad *C_{UDSI}(c,t,w1)/C_{UDSI}(c,t,\text{mag}_\text{grad}_w));

dS[eqn]=0;
}
else

source = dS[eqn] = 0.;

return source;

}
This is the finite-volume Euler solver 2DFlow written in Fortran 90. The main program is solver.f90 which calls subroutines to perform specific tasks. These subroutines are included as well.

program solver

!Main program of 2DFlow

!---performs temporal integration

!---calls subrotines

!PROGRAM SET-UP SPECIFIC TO ADVECTING VORTEX TEST CASE

implicit none

integer:: I,J,T,NSHIFT

integer:: IMAX,JMAX,ITER,COUNTER

real,dimension(:,:),allocatable :: X1,X2,MACH

real,dimension(:,:),allocatable :: S1X,S1Y,S2X,S2Y,&
S3X,S3Y,S4X,S4Y,DS1,DS2,&
DS3,DS4,N1X,N1Y,N2X,N2Y,&
N3X,N3Y,N4X,N4Y,VOL,DT,&
real, dimension(:), allocatable :: TSTEPS, RES_1, &
RES_2, RES_3, RES_4

real, parameter :: T0 = 300.0, P0 = 101325, R = 287.0, G = 1.4
real, parameter :: CFL = 0.8
real :: DTMIN, TOTALTIME, RADIUS, CIRC, start, finish
real, dimension(2) :: DTMINLOC

call cpu_time(start)

! dimension domain
IMAX = 500
JMAX = 200

allocate (X1(IMAX, JMAX), X2(IMAX, JMAX))
allocate (S1X(IMAX-1, JMAX-1), S1Y(IMAX-1, JMAX-1))
allocate(S2X(IMAX-1,JMAX-1),S2Y(IMAX-1,JMAX-1))
allocate(S3X(IMAX-1,JMAX-1),S3Y(IMAX-1,JMAX-1))
allocate(S4X(IMAX-1,JMAX-1),S4Y(IMAX-1,JMAX-1))
allocate(DS1(IMAX-1,JMAX-1),DS2(IMAX-1,JMAX-1))
allocate(DS3(IMAX-1,JMAX-1),DS4(IMAX-1,JMAX-1))
allocate(N1X(IMAX-1,JMAX-1),N1Y(IMAX-1,JMAX-1))
allocate(N2X(IMAX-1,JMAX-1),N2Y(IMAX-1,JMAX-1))
allocate(N3X(IMAX-1,JMAX-1),N3Y(IMAX-1,JMAX-1))
allocate(N4X(IMAX-1,JMAX-1),N4Y(IMAX-1,JMAX-1))
allocate(VOL(IMAX-1,JMAX-1))
allocate(CC1(IMAX-1,JMAX-1),CC2(IMAX-1,JMAX-1))
allocate(VMAG(IMAX-1,JMAX-1))
allocate(CV(IMAX-1,JMAX-1,4))
allocate(DV(IMAX-1,JMAX-1,6))
allocate(FPOS(IMAX-1,JMAX-1,4),FNEG(IMAX-1,JMAX-1,4))
allocate(GPOS(IMAX-1,JMAX-1,4),GNEG(IMAX-1,JMAX-1,4))
allocate(DT(IMAX-1,JMAX-1))
allocate(RESID(IMAX-3,JMAX-3,4))
allocate(MACH(IMAX-1,JMAX-1))
allocate(VORT(IMAX-1,JMAX-1))
allocate(VCSOURCE1(IMAX-1,JMAX-1))
allocate(VCSOURCE2(IMAX-1,JMAX-1))
allocate(VCFLAG(IMAX-1,JMAX-1),EPS1(IMAX-1,JMAX-1))

!compute node x,y locations
call nodes(IMAX,JMAX,X1,X2)

!compute face area vectors, face area,
!and unit normal vectors

call farea(IMAX,JMAX,X1,X2,S1X,S1Y,S2X,S2Y, &
S3X,S3Y,S4X,S4Y,DS1,DS2, &
DS3,DS4,N1X,N1Y,N2X,N2Y, &
N3X,N3Y,N4X,N4Y,VOL,CC1,CC2)

!initialize flow -- primitive variables
call isentropic_vortex_patch2(IMAX,JMAX,&
T0,P0,R,DV,X1,X2,CC1,CC2)

!compute conservative variables from initialized primitives
call con_var(IMAX,JMAX,G,CV,DV)

ITER = 3005!number of time steps

allocate(TSTEPS(ITER),RES_1(ITER),RES_2(ITER))
allocate(RES_3(ITER),RES_4(ITER))

!initialize total time and temporal iteration counter

TOTALTIME = 0.0
COUNTER = 1
NSHIFT = 0

do T=1,ITER !temporal (outer) loop

   call steger_warm(G,IMAX,JMAX,DV,FPOS,FNEG,GPOS,GNEG)
   call upwind1(IMAX,JMAX,S2X,S4X,S1Y,S3Y, 
                  FPOS,FNEG,GPOS,GNEG,RESID)
   call timestep(CFL,IMAX,JMAX,DTMIN,&
                DTMINLOC,N1X,N2X,N3X,N4X,&
                N1Y,N2Y,N3Y,N4Y,DS1,DS2,DS3,DS4, &
                VOL,DT,DV)

   !DTMIN = 0.0001 !override timestep computed by timestep.f90

   !compute mach number
   do I=1,IMAX-1
      do J=1,JMAX-1
         MACH(I,J) = sqrt(DV(I,J,2)*DV(I,J,2) +&
                 112
& DV(I,J,3)*DV(I,J,3)) / DV(I,J,6)

    end do

end do

! compute velocity magnitude

do I=1,IMAX-1
    do J=1,JMAX-1
        VMAG(I,J) = sqrt(abs(DV(I,J,2)*DV(I,J,2)) +
        & abs(DV(I,J,3)*DV(I,J,3)))
    end do
end do

! compute vorticity

call vorticity(IMAX,JMAX,DV,CC1,CC2,VORT)

! write TECPLLOT data

! call tec_output1(IMAX,JMAX,COUNTER,CC1,CC2,DV,VMAG,VORT)

call tec_output_isentropic_vortex(IMAX,JMAX,COUNTER,CC1,&
& CC2,DV,VMAG,VORT,VCSOURCE1,VCSOURCE2,VCFLAG,EPS1)

call gnuplot_output_isentropic_vortex(IMAX,JMAX,COUNTER,CC1,&
CC2,DV,VMAG,VORT,VCSOURCE1,VCSOURCE2,VCFLAG,EPS1)

! compute conservative variables at next time step
do I=3,IMAX-3
    do J=3,JMAX-3
        CV(I,J,1) = CV(I,J,1) + (DTMIN)/VOL(I,J)*RESID(I,J,1)
        CV(I,J,2) = CV(I,J,2) + (DTMIN)/VOL(I,J)*RESID(I,J,2)
        CV(I,J,3) = CV(I,J,3) + (DTMIN)/VOL(I,J)*RESID(I,J,3)
        CV(I,J,4) = CV(I,J,4) + (DTMIN)/VOL(I,J)*RESID(I,J,4)
    end do
end do
end do

!call confinement subroutine to compute VC source terms
if (T >= 5) then
    call confinement1(IMAX,JMAX,DV,CC1,CC2,VOL,&
    VORT,VCSOURCE1,VCSOURCE2,VCFLAG,EPS1)
else
end if

!add VC source terms to conservative variables
if (T >= 1) then
    do I=3,IMAX-3
        do J=3,JMAX-3
            CV(I,J,2) = CV(I,J,2) + (DTMIN)/VOL(I,J)*VCSOURCE1(I,J)
            CV(I,J,3) = CV(I,J,3) + (DTMIN)/VOL(I,J)*VCSOURCE2(I,J)
        end do
    end do
end if
call prim_var(IMAX,JMAX,R,G,CV,DV)
call bc_extrap(IMAX,JMAX,DV,CC1,CC2)
call con_var(IMAX,JMAX,G,CV,DV)

TOTALTIME = TOTALTIME + DTMIN !update total time

!write timestep data to file
if (T == 1) then
   open(unit=50,file="timesteps.dat")
   write(50,*), " COUNTER,TOTALTIME,TIMESTEP"
end if
write(50,*), COUNTER,TOTALTIME,DTMIN
if (T == ITER) then
   close(50)
end if

!computation of circulation
CIRC = 0.0
do I=1,IMAX-1
   do J=1,JMAX-1
      !compute radius, initial center (10.0,10.0)
      RADIUS = sqrt(abs((CC1(I,J)-10.0-100*TOTALTIME)*(CC1(I,J)-&
&10.0-100*TOTALTIME)) + abs((CC2(I,J)-10.0)*(CC2(I,J)-10.0)))

    if (RADIUS <= 1.0) then
      CIRC = CIRC + VORT(I,J)*VOL(I,J)
    end if

end do
end do

print *, T, TOTALTIME, DDMIN, CIRC, maxval(EPS1)

if (T == 1) then
  open(unit=49, file="circulation.dat")
  write(49,*), 'VARIABLES = "ITER","TIME","CIRCULATION"
  write(49,*), 'ZONE I=',ITER,'DATAPACKING=POINT'
end if

write(49,*), COUNTER, TOTALTIME, CIRC
if (T == ITER) then
  close(49)
end if

COUNTER = COUNTER + 1
end do
call cpu_time(finish)

open(unit=2,file="runtime.dat")

write(2,*) , finish-start

close(2)

end program solver

subroutine nodes(IMAX,JMAX,X1,X2)
    implicit none
    integer :: I,J
    integer,intent(in) :: IMAX,JMAX
    real,dimension(IMAX,JMAX),intent(out) :: X1,X2

!distribute nodes in x,y space each with associated
!i,j index

    do I=1,IMAX
        do J=1,JMAX
            X1(I,J) = 0.05*real(I) - 0.05
            X2(I,J) = 0.05*real(J) - 0.05
        end do
    end do

end subroutine nodes
subroutine farea(IMAX,JMAX,X1,X2,S1X,S1Y,S2X,S2Y, &
S3X,S3Y,S4X,S4Y,DS1,DS2, &
DS3,DS4,N1X,N1Y,N2X,N2Y, &
N3X,N3Y,N4X,N4Y,VOL,CC1,CC2)

integer,intent(in) :: IMAX,JMAX
real,dimension(IMAX,JMAX),intent(in) :: X1,X2
real,dimension(IMAX-1,JMAX-1),intent(out) :: S1X,S1Y,&
S2X,S2Y, S3X,S3Y,S4X,S4Y,DS1,DS2, &
DS3,DS4,N1X,N1Y,N2X,N2Y, &
N3X,N3Y,N4X,N4Y,VOL,CC1,CC2

real :: A123,A134,RC123X,RC134X,RC123Y,RC134Y

do I=1,(IMAX-1)
  do J=1,(JMAX-1)
    !face vectors
    S1X(I,J) = X2(I+1,J ) - X2(I ,J )
    S1Y(I,J) = X1(I ,J ) - X1(I+1,J )
    S2X(I,J) = X2(I+1,J+1) - X2(I+1,J )
    S2Y(I,J) = X1(I+1,J ) - X1(I+1,J+1)
    S3X(I,J) = X2(I ,J+1) - X2(I+1,J+1)
    S3Y(I,J) = X1(I ,J+1) - X1(I+1,J+1)
    
  end do
end do
S4X(I,J) = X2(I,J) - X2(I,J+1)
S4Y(I,J) = X1(I,J+1) - X1(I,J)

!face areas

DS1(I,J) = sqrt(S1X(I,J)*S1X(I,J) + S1Y(I,J)*S1Y(I,J))
DS2(I,J) = sqrt(S2X(I,J)*S2X(I,J) + S2Y(I,J)*S3Y(I,J))
DS3(I,J) = sqrt(S3X(I,J)*S3X(I,J) + S3Y(I,J)*S3Y(I,J))
DS4(I,J) = sqrt(S4X(I,J)*S4X(I,J) + S4Y(I,J)*S4Y(I,J))

!compute unit normal vectors

N1X(I,J) = S1X(I,J)/DS1(I,J)
N1Y(I,J) = S1Y(I,J)/DS1(I,J)
N2X(I,J) = S2X(I,J)/DS2(I,J)
N2Y(I,J) = S2Y(I,J)/DS2(I,J)
N3X(I,J) = S3X(I,J)/DS3(I,J)
N3Y(I,J) = S3Y(I,J)/DS3(I,J)
N4X(I,J) = S4X(I,J)/DS4(I,J)
N4Y(I,J) = S4Y(I,J)/DS4(I,J)

!compute volume, assume unit depth

VOL(I,J) = 0.5*((X1(I,J)-X1(I+1,J+1))*&
(X2(I+1,J)-X2(I,J+1)) + &
(X1(I,J+1)-X1(I+1,J))*(X2(I,J)-X2(I+1,J+1)))

!compute cell centers, divide quad into triangles

!areas of sub-triangles
A123 = 0.5*((X1(I,J)-X1(I+1,J))*(X2(I,J)+X2(I+1,J))&
            +(X1(I+1,J)-X1(I+1,J+1))*(X2(I+1,J)+X2(I+1,J+1))&
            +(X1(I+1,J+1)-X1(I,J))*(X2(I+1,J+1)+X2(I,J)))

A134 = 0.5*((X1(I,J)-X1(I+1,J+1))*(X2(I,J)+X2(I+1,J+1))&
            +(X1(I+1,J+1)-X1(I,J+1))*(X2(I+1,J+1)+X2(I,J+1))&
            +(X1(I,J+1)-X1(I,J))*(X2(I,J+1)+X2(I,J)))

!compute centers of sub-triangles
RC123X = 0.33333333333*(X1(I,J)+X1(I+1,J)+X1(I+1,J+1))
RC123Y = 0.33333333333*(X2(I,J)+X2(I+1,J)+X2(I+1,J+1))
RC134X = 0.33333333333*(X1(I,J)+X1(I+1,J+1)+X1(I,J+1))
RC134Y = 0.33333333333*(X2(I,J)+X2(I+1,J+1)+X2(I,J+1))

!cell centers of quads
CC1(I,J) = (A123*RC123X + A134*RC134X)/(A123 + A134)
CC2(I,J) = (A123*RC123Y + A134*RC134Y)/(A123 + A134)

end do
end do

end subroutine farea

subroutine isentropic_vortex_patch2(IMAX,JMAX,&
    T0,P0,R,G,DV,X1,X2,CC1,CC2)

    implicit none

end subroutine isentropic_vortex_patch2
!---SUBROUTINE TASK(S)
!
1. initialize primitive variables for isentropic vortex

integer, intent(in) :: IMAX, JMAX
real, intent(in) :: T0, P0, R, G
real, dimension(IMAX-1, JMAX-1, 6), intent(out) :: DV
real, dimension(IMAX, JMAX), intent(in) :: X1, X2
real, dimension(IMAX-1, JMAX-1), intent(in) :: CC1, CC2

integer :: I, J
real :: VORT, X, Y, RADIUS, THETA, CORE_RADIUS, DUM
real, parameter :: PI = 3.14159
real, parameter :: BETA = 50.0
real :: CENTERX, CENTERY

do I = 1, IMAX - 1
  do J = 1, JMAX - 1
    CENTERX = CC1(200, 200)
    CENTERY = CC2(200, 200)
    RADIUS = sqrt((CC1(I, J) - CENTERX) * (CC1(I, J) - CENTERX) + &
                   (CC2(I, J) - CENTERY) * (CC2(I, J) - CENTERY))
if ((CC1(I,J) - CENTERX) < 0.0) then

THETA = PI - asin(Y/RADIUS)

else

THETA = asin(Y/RADIUS)

end if

! 2D VORTEX IN ISENTROPIC FLOW

DV(I,J,2) = 100.0 + BETA/(2.0*PI)**

EXP((1-RADIUS**2.0)/2.0)*(CENTERY-CC2(I,J))

DV(I,J,3) = 0.0 + BETA/(2.0*PI)**

EXP((1-RADIUS**2.0)/2.0)*(CC1(I,J)-CENTERX)

DV(I,J,5) = 101325.0

DV(I,J,4) = 288.203086*(1.0 - (G-1.0)**BETA/(8.0*G*PI**2)&
*exp(1.0-RADIUS**2.0))

DV(I,J,1) = 1.225*(DV(I,J,4)/288.203086)**(1.0/(G-1.0))

DV(I,J,6) = sqrt(G*R*DV(I,J,4))

end do

end do

! specify conditions for r = 0

DV(200,200,1) = 0.225516718
DV(200,200,2) = 100.0

DV(200,200,3) = 0.0

DV(200,200,5) = 101325.0

DV(200,200,4) = 146.458873266

DV(200,200,6) = sqrt(G*R*DV(50,50,4))

end subroutine isentropic_vortex_patch2

subroutine con_var(IMAX,JMAX,G,CV,DV)

!---SUBROUTINE TASK(S)

! 1. compute conservative variables from flow primitives

! CV(:,:,1) = density

! CV(:,:,2) = density*u-velocity

! CV(:,:,3) = density*v-velocity

! CV(:,:,4) = density*Energy

! = density*(c^2/(g*(g-1))

! + 0.5*(u-velocity^2 + v-velocity^2))

implicit none

end subroutine con_var
integer, intent(in) :: IMAX, JMAX
real, intent(in) :: G
real, dimension(IMAX-1, JMAX-1, 4), intent(out) :: CV
real, dimension(IMAX-1, JMAX-1, 6), intent(in) :: DV

integer :: I, J

do I = 1, IMAX - 1
  do J = 1, JMAX - 1
    CV(I, J, 1) = DV(I, J, 1)
    CV(I, J, 2) = DV(I, J, 1) * DV(I, J, 2)
    CV(I, J, 3) = DV(I, J, 1) * DV(I, J, 3)
    CV(I, J, 4) = DV(I, J, 1) * (DV(I, J, 6) * DV(I, J, 6) / (G * (G - 1)) + 0.5 * (DV(I, J, 2) * DV(I, J, 2) + DV(I, J, 3) * DV(I, J, 3)))
  end do
end do

end subroutine con_var

subroutine vorticity(IMAX, JMAX, DV, CC1, CC2, VORT)

implicit none

end subroutine vorticity
integer, intent(in) :: IMAX, JMAX
real, dimension(IMAX-1, JMAX-1, 6), intent(in) :: DV
real, dimension(IMAX-1, JMAX-1), intent(in) :: CC1, CC2
real, dimension(IMAX-1, JMAX-1), intent(out) :: VORT
integer :: I, J
real :: DVDX, DUDY

! computed dv/dx

do I = 2, IMAX - 2
  do J = 2, JMAX - 2
    DVDX = 0.5 * ( (DV(I+1, J, 3) - DV(I, J, 3)) / (CC1(I+1, J) - CC1(I, J)) +
                  (DV(I, J, 3) - DV(I-1, J, 3)) / (CC1(I, J) - CC1(I-1, J)) )
    DUDY = 0.5 * ( (DV(I, J+1, 2) - DV(I, J, 2)) / (CC2(I, J+1) - CC2(I, J)) +
                  (DV(I, J, 2) - DV(I, J-1, 2)) / (CC2(I, J) - CC2(I, J-1)) )
    VORT(I, J) = DVDX - DUDY
  end do
end do

end subroutine vorticity

subroutine steger_warm(G, IMAX, JMAX, DV, FPOS, FNEG, GPOS, GNEG)
implicit none

integer :: I,J
real :: eigu1,eigu2,eigu3,eigv1,eigv2,eigv3,alpha

real,intent(in) :: G
integer,intent(in) :: IMAX,JMAX
real,dimension(IMAX-1,JMAX-1,6),intent(in) :: DV
real,dimension(IMAX-1,JMAX-1,4),intent(inout) :: FPOS,FNEG,&
   GPOS,GNEG

do I=1,IMAX-1
   do J=1,JMAX-1
      !compute split fluxes at cell-centers, Steger-Warming
      
      !see page 440 in Numerical Computation of Internal
      !and External Flows, Vol. 2, by Hirsch

      !subsonic--positive u-velocity
      if (abs(DV(I,J,2))<DV(I,J,6) .AND. DV(I,J,2)>= 0.0)then
         eigu1 = DV(I,J,2)
         eigu2 = DV(I,J,2) + DV(I,J,6)
         eigu3 = 0.0
      end if
alpha = 2*(G-1)*eigu1+eigu2+eigu3
FPOS(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)
FPOS(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2) &
+ DV(I,J,6)*(eigu2-eigu3))
FPOS(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3))
FPOS(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5* &
(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &
DV(I,J,2)*DV(I,J,6)*(eigu2-eigu3) + &
DV(I,J,6)*DV(I,J,6)*(eigu2+eigu3)/(G-1))
eigu1 = 0.0
eigu2 = 0.0
eigu3 = DV(I,J,2) - DV(I,J,6)
alpha = 2*(G-1)*eigu1+eigu2+eigu3
FNEG(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)
FNEG(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2) &
+ DV(I,J,6)*(eigu2-eigu3))
FNEG(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3))
FNEG(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5* &
(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &
DV(I,J,2)*DV(I,J,6)*(eigu2-eigu3) + &
DV(I,J,6)*DV(I,J,6)*(eigu2+eigu3)/(G-1))

!subsonic--negative u-velocity
elseif (abs(DV(I,J,2))<DV(I,J,6) .AND. DV(I,J,2)<0.0)then

127
eigu1 = DV(I,J,2)
eigu2 = DV(I,J,2) + DV(I,J,6)
eigu3 = 0.0

alpha = 2*(G-1)*eigu1+eigu2+eigu3
FPOS(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)
FPOS(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2) & + DV(I,J,6)*(eigu2-eigu3))
FPOS(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3))
FPOS(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5 & *(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &

DV(I,J,2)*DV(I,J,6)*(eigu2-eigu3) + &

DV(I,J,6)*DV(I,J,6)*(eigu2+eigu3)/(G-1))

eigu1 = 0.0

eigu2 = 0.0

eigu3 = DV(I,J,2) - DV(I,J,6)
alpha = 2*(G-1)*eigu1+eigu2+eigu3

FNEG(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)
FNEG(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2) & + DV(I,J,6)*(eigu2-eigu3))
FNEG(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3))
FNEG(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5 & *(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &

DV(I,J,2)*DV(I,J,6)*(eigu2-eigu3) + &

128
!supersonic--positive u-velocity

elseif (abs(DV(I,J,2))>DV(I,J,6) .AND. DV(I,J,2)>0.0) then

  eigu1 = DV(I,J,2)
  eigu2 = DV(I,J,2) + DV(I,J,6)
  eigu3 = DV(I,J,2) - DV(I,J,6)
  
  alpha = 2*(G-1)*eigu1+eigu2+eigu3

  FPOS(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)
  FPOS(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2) +
  DV(I,J,6)*(eigu2-eigu3)) +

  FPOS(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3))
  FPOS(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5*

  (DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &

  DV(I,J,2)*DV(I,J,6)*eigu2-eigu3) + &

  DV(I,J,6)*DV(I,J,6)*eigu2+eigu3)/(G-1))

  eigu1 = 0.0
  eigu2 = 0.0
  eigu3 = 0.0

  alpha = 2*(G-1)*eigu1+eigu2+eigu3

  FNEG(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)
  FNEG(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2) &

  + DV(I,J,6)*eigu2-eigu3))

  FNEG(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3))
FNEG(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5*&
(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &
DV(I,J,2)*DV(I,J,6)*(eigu2-eigu3) + &
DV(I,J,6)*DV(I,J,6)*(eigu2+eigu3)/(G-1))

!supersonic--negative u-velocity

elseif (abs(DV(I,J,2))>DV(I,J,6) .AND. DV(I,J,2)<0.0)then

eigu1 = DV(I,J,2)
eigu2 = 0.0
eigu3 = 0.0

alpha = 2*(G-1)*eigu1+eigu2+eigu3

FPOS(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)
FPOS(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2)&
+ DV(I,J,6)*(eigu2-eigu3))
FPOS(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3))
FPOS(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5*&
(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &
DV(I,J,2)*DV(I,J,6)*(eigu2-eigu3) + &
DV(I,J,6)*DV(I,J,6)*(eigu2+eigu3)/(G-1))

eigu1 = 0.0
eigu2 = DV(I,J,2) + DV(I,J,6)
eigu3 = DV(I,J,2) - DV(I,J,6)

alpha = 2*(G-1)*eigu1+eigu2+eigu3

FNEG(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)

130
\[
\begin{align*}
\text{FNEG}(I,J,2) &= \frac{\text{DV}(I,J,1)}{2.0 \times G} (\alpha \times \text{DV}(I,J,2) + \text{DV}(I,J,6) (eigu2 - eigu3)) \\
\text{FNEG}(I,J,3) &= \frac{\text{DV}(I,J,1)}{2.0 \times G} (\alpha \times \text{DV}(I,J,3)) \\
\text{FNEG}(I,J,4) &= \frac{\text{DV}(I,J,1)}{2.0 \times G} (\alpha \times 0.5 \times (\text{DV}(I,J,2) \times \text{DV}(I,J,2) + \text{DV}(I,J,3) \times \text{DV}(I,J,3)) + \\
&\quad \text{DV}(I,J,2) \times \text{DV}(I,J,6) (eigu2 - eigu3) + \\
&\quad \text{DV}(I,J,6) \times \text{DV}(I,J,6) (eigu2 + eigu3)/(G-1)) \\
\end{align*}
\]

end if

! subsonic--positive v-velocity

if (abs(DV(I,J,3)) < DV(I,J,6) .AND. DV(I,J,3) >= 0.0) then

\[
\begin{align*}
\text{eigv1} &= \text{DV}(I,J,3) \\
\text{eigv2} &= \text{DV}(I,J,3) + \text{DV}(I,J,6) \\
\text{eigv3} &= 0.0 \\
\alpha &= 2 \times (G-1) \times \text{eigv1} + \text{eigv2} + \text{eigv3} \\
\text{GPOS}(I,J,1) &= \frac{\text{DV}(I,J,1)}{2.0 \times G} (\alpha) \\
\text{GPOS}(I,J,2) &= \frac{\text{DV}(I,J,1)}{2.0 \times G} (\alpha \times \text{DV}(I,J,2)) \\
\text{GPOS}(I,J,3) &= \frac{\text{DV}(I,J,1)}{2.0 \times G} (\alpha \times \text{DV}(I,J,3)) + \\
&\quad \text{DV}(I,J,6) (\text{eigv2} - \text{eigv3}) \\
\text{GPOS}(I,J,4) &= \frac{\text{DV}(I,J,1)}{2.0 \times G} (\alpha \times 0.5 \times (\text{DV}(I,J,2) \times \text{DV}(I,J,2) + \text{DV}(I,J,3) \times \text{DV}(I,J,3)) + \\
&\quad \text{DV}(I,J,3) \times \text{DV}(I,J,6) (\text{eigv2} - \text{eigv3}) + \\
&\quad \text{DV}(I,J,6) \times \text{DV}(I,J,6) (\text{eigv2} + \text{eigv3})/(G-1)) \\
\end{align*}
\]
eigv1 = 0.0

eigv2 = 0.0

eigv3 = DV(I,J,3) - DV(I,J,6)

alpha = 2*(G-1)*eigv1+eigv2+eigv3

GNEG(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)

GNEG(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2))

GNEG(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3)&
                      + DV(I,J,6)*(eigv2-eigv3))

GNEG(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5*&(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) &
                      DV(I,J,3)*DV(I,J,6)*(eigv2-eigv3) &
                      DV(I,J,6)*DV(I,J,6)*(eigv2+eigv3)/(G-1))

!subsonic--negative v-velocity

elseif (abs(DV(I,J,3))<DV(I,J,6) .AND. DV(I,J,3)<0.0)then

  eigv1 = DV(I,J,3)

  eigv2 = DV(I,J,3) + DV(I,J,6)

  eigv3 = 0.0

  alpha = 2*(G-1)*eigv1+eigv2+eigv3

  GPOS(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)

  GPOS(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2))

  GPOS(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3)&
                      + DV(I,J,6)*(eigv2-eigv3))

  GPOS(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5*&
                      (DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) &
                      DV(I,J,3)*DV(I,J,6)*(eigv2-eigv3) &
                      DV(I,J,6)*DV(I,J,6)*(eigv2+eigv3)/(G-1))
(DV(I,J,2) * DV(I,J,2) + DV(I,J,3) * DV(I,J,3)) + & 
DV(I,J,3) * DV(I,J,6) * (eigv2 - eigv3) + & 
DV(I,J,6) * DV(I,J,6) * (eigv2 + eigv3) / (G - 1))

eigv1 = 0.0

eigv2 = 0.0

eigv3 = DV(I,J,3) - DV(I,J,6)

alpha = 2 * (G - 1) * eigv1 + eigv2 + eigv3

GNEG(I,J,1) = DV(I,J,1) / (2.0 * G) * (alpha)

GNEG(I,J,2) = DV(I,J,1) / (2.0 * G) * (alpha * DV(I,J,2))

GNEG(I,J,3) = DV(I,J,1) / (2.0 * G) * (alpha * DV(I,J,3) &
(DV(I,J,2) * DV(I,J,2) + DV(I,J,3) * DV(I,J,3)) + & 
DV(I,J,3) * DV(I,J,6) * (eigv2 - eigv3) + & 
DV(I,J,6) * DV(I,J,6) * (eigv2 + eigv3) / (G - 1))

!supersonic--positive v-velocity

elseif (abs(DV(I,J,3)) > DV(I,J,6) .AND. DV(I,J,3) > 0.0) then

eigv1 = DV(I,J,3)

alpha = 2 * (G - 1) * eigv1 + eigv2 + eigv3

GPOS(I,J,1) = DV(I,J,1) / (2.0 * G) * (alpha)

GPOS(I,J,2) = DV(I,J,1) / (2.0 * G) * (alpha * DV(I,J,2))
GPOS(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3) &
+ DV(I,J,6)*(eigv2-eigv3))

GPOS(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5* &
(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &
DV(I,J,3)*DV(I,J,6)*(eigv2-eigv3) + &
DV(I,J,6)*DV(I,J,6)*(eigv2+eigv3)/(G-1))

eigv1 = 0.0

eigv2 = 0.0

eigv3 = 0.0

alpha = 2*(G-1)*eigv1+eigv2+eigv3

GNEG(I,J,1) = DV(I,J,1)/(2.0*G)*(alpha)

GNEG(I,J,2) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,2))

GNEG(I,J,3) = DV(I,J,1)/(2.0*G)*(alpha*DV(I,J,3) &
+ DV(I,J,6)*(eigv2-eigv3))

GNEG(I,J,4) = DV(I,J,1)/(2.0*G)*(alpha*0.5* &
(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + &
DV(I,J,3)*DV(I,J,6)*(eigv2-eigv3) + &
DV(I,J,6)*DV(I,J,6)*(eigv2+eigv3)/(G-1))

!supersonic--negative v-velocity

elseif (abs(DV(I,J,3))>DV(I,J,6) .AND. DV(I,J,3)<0.0) then

eigv1 = DV(I,J,3)

eigv2 = 0.0

eigv3 = 0.0
\[
\begin{align*}
\alpha &= 2(G-1)eigv1 + eigv2 + eigv3 \\
GPOS(I,J,1) &= DV(I,J,1)/(2.0*G)*(\alpha) \\
GPOS(I,J,2) &= DV(I,J,1)/(2.0*G)*(\alpha*DV(I,J,2)) \\
GPOS(I,J,3) &= DV(I,J,1)/(2.0*G)*(\alpha*DV(I,J,3) + DV(I,J,6)*(eigv2-eigv3)) \\
GPOS(I,J,4) &= DV(I,J,1)/(2.0*G)*(\alpha*0.5*(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + DV(I,J,3)*DV(I,J,6)*(eigv2-eigv3) + DV(I,J,6)*DV(I,J,6)*(eigv2+eigv3)/(G-1)) \\
eigv1 &= 0.0 \\
eigv2 &= DV(I,J,3) + DV(I,J,6) \\
eigv3 &= DV(I,J,3) - DV(I,J,6) \\
\alpha &= 2(G-1)eigv1 + eigv2 + eigv3 \\
GNEG(I,J,1) &= DV(I,J,1)/(2.0*G)*(\alpha) \\
GNEG(I,J,2) &= DV(I,J,1)/(2.0*G)*(\alpha*DV(I,J,2)) \\
GNEG(I,J,3) &= DV(I,J,1)/(2.0*G)*(\alpha*DV(I,J,3) + DV(I,J,6)*(eigv2-eigv3)) \\
GNEG(I,J,4) &= DV(I,J,1)/(2.0*G)*(\alpha*0.5*(DV(I,J,2)*DV(I,J,2)+DV(I,J,3)*DV(I,J,3)) + DV(I,J,3)*DV(I,J,6)*(eigv2-eigv3) + DV(I,J,6)*DV(I,J,6)*(eigv2+eigv3)/(G-1)) \\
\end{align*}
\]
endif

dendif

d do

d do

d subroutine steger_warm

subroutine upwind1(IMAX,JMAX,S2X,S4X,S1Y,S3Y, &
                   FPOS,FNEG,GPOS,GNEG,RESID)

implicit none

integer :: I,J

integer,intent(in) :: IMAX,JMAX
real,dimension(IMAX-1,JMAX-1),intent(in) :: S2X,S4X,S1Y,S3Y
real,dimension(IMAX-1,JMAX-1,4),intent(in) :: FPOS,FNEG,GPOS,GNEG
real,dimension(IMAX-3,JMAX-3,4),intent(inout) :: RESID

do I=3,IMAX-3
  do J=3,JMAX-3
    ! modify with x-component of flux
    RESID(I,J,1) = (-1.0)*((FPOS(I,J,1)+FNEG(I+1,J,1))*S2X(I,J)+&
                        (FNEG(I,J,1)+FPOS(I-1,J,1))*S4X(I,J))
  end do
end do
end subroutine upwind1
RESID(I,J,2) = (-1.0)*((FPOS(I,J,2)+FNEG(I+1,J,2))*S2X(I,J)+
(FNEG(I,J,2)+FPOS(I-1,J,2))*S4X(I,J))

RESID(I,J,3) = (-1.0)*((FPOS(I,J,3)+FNEG(I+1,J,3))*S2X(I,J)+
(FNEG(I,J,3)+FPOS(I-1,J,3))*S4X(I,J))

RESID(I,J,4) = (-1.0)*((FPOS(I,J,4)+FNEG(I+1,J,4))*S2X(I,J)+
(FNEG(I,J,4)+FPOS(I-1,J,4))*S4X(I,J))

!modify with y-component of flux

RESID(I,J,1) = RESID(I,J,1) + &
(-1.0)*((GPOS(I,J,1)+GNEG(I,J+1,1))*S3Y(I,J)+
(GNEG(I,J,1)+GPOS(I,J-1,1))*S1Y(I,J))

RESID(I,J,2) = RESID(I,J,2) + &
(-1.0)*((GPOS(I,J,2)+GNEG(I,J+1,2))*S3Y(I,J)+
(GNEG(I,J,2)+GPOS(I,J-1,2))*S1Y(I,J))

RESID(I,J,3) = RESID(I,J,3) + &
(-1.0)*((GPOS(I,J,3)+GNEG(I,J+1,3))*S3Y(I,J)+
(GNEG(I,J,3)+GPOS(I,J-1,3))*S1Y(I,J))

RESID(I,J,4) = RESID(I,J,4) + &
(-1.0)*((GPOS(I,J,4)+GNEG(I,J+1,4))*S3Y(I,J)+
(GNEG(I,J,4)+GPOS(I,J-1,4))*S1Y(I,J))

end do

end do
end subroutine upwind1

subroutine timestep(CFL,IMAX,JMAX,DTMIN,DTMINLOC,N1X,N2X,N3X,N4X, &
                   N1Y,N2Y,N3Y,N4Y,DS1,DS2,DS3,DS4, &
                   VOL,DT,DV)

implicit none

integer :: I,J
real :: AVGN1X,AVGN1Y,AVGN2X,AVGN2Y,AVGSX,AVGSY,U,V,G1,G2
real,intent(in) :: CFL
integer,intent(in) :: IMAX,JMAX
real,intent(out) :: DTMIN
real,dimension(2),intent(out) :: DTMINLOC
real,dimension(IMAX-1,JMAX-1),intent(in) :: N1X,N2X, &
                                         N3X,N4X,N1Y,N2Y,N3Y,N4Y,DS1,DS2,DS3,DS4,VOL
real,dimension(IMAX-1,JMAX-1,6),intent(inout) :: DT
real,dimension(IMAX-1,JMAX-1,6),intent(in) :: DV

do I=1,IMAX-1
   do J=1,JMAX-1
      AVGN1X = 0.5*(N4X(I,J) - N2X(I,J))
   enddo
enddo
AVGN1Y = 0.5*(N4Y(I,J) - N2Y(I,J))
AVGN2X = 0.5*(N1X(I,J) - N3X(I,J))
AVGN2Y = 0.5*(N1Y(I,J) - N3Y(I,J))
AVGSX = 0.5*(DS4(I,J) + DS2(I,J))
AVGSY = 0.5*(DS1(I,J) + DS3(I,J))

U = AVGN1X*DV(I,J,2) + AVGN1Y*DV(I,J,3)
V = AVGN2X*DV(I,J,2) + AVGN2Y*DV(I,J,3)

G1 = (abs(U) + DV(I,J,6))*AVGSX
G2 = (abs(V) + DV(I,J,6))*AVGSY

DT(I,J) = CFL*VOL(I,J)/(G1+G2)

end do
end do

DTMIN = minval(DT)
DTMINLOC(:) = minloc(DT)

end subroutine timestep

subroutine confinement1(IMAX,JMAX,DV,CC1,CC2,VOL,VORT,&
VCSOURCE1,VCSOURCE2,VCFLAG,EPS1)

implicit none

end subroutine confinement1
integer, intent(in) :: IMAX, JMAX

real, dimension(IMAX-1, JMAX-1, 6), intent(in) :: DV

real, dimension(IMAX-1, JMAX-1), intent(in) :: CC1, CC2, VOL

real, dimension(IMAX-1, JMAX-1), intent(in) :: VORT

real, dimension(IMAX-1, JMAX-1), intent(out) :: VCSOURCE1, VCSOURCE2

real, dimension(IMAX-1, JMAX-1), intent(out) :: VCFLAG, EPS1

! real, intent(out) :: junk

integer :: I, J

real :: GRADX, GRADY, GRAD_MAG, N1, N2, FLAG, H, VORTICITY_MAX

! limit confinement to regions of significant vorticity

VORTICITY_MAX = maxval(abs(VORT))

do I = 3, IMAX - 3

do J = 3, JMAX - 3

! compute gradient of vorticity

GRADX = 0.5 * ((VORT(I+1, J) - VORT(I, J)) / (CC1(I+1, J) - CC1(I, J)) +
               (VORT(I, J) - VORT(I-1, J)) / (CC1(I, J) - CC1(I-1, J)))

GRADY = 0.5 * ((VORT(I, J+1) - VORT(I, J)) / (CC2(I, J+1) - CC2(I, J)) +
               (VORT(I, J) - VORT(I, J-1)) / (CC2(I, J) - CC2(I, J-1)))

GRAD_MAG = sqrt(abs(GRADX*GRADX) + abs(GRADY*GRADY))
VCFLAG(I,J) = 0.5*(abs(GRADX) + abs(GRADY))

N1 = GRADX/(GRAD_MAG + 1.0E-9)
N2 = GRADY/(GRAD_MAG + 1.0E-9)

if (VORT(I,J) >= 0.1*VORTICITY_MAX) then
  EPS1(I,J) = 100.0
else
  EPS1(I,J) = 0.0
end if

VCSOURCE1(I,J) = (1.0)* VOL(I,J)*DV(I,J,1)*&
EPS1(I,J)*N2*VORT(I,J)

VCSOURCE2(I,J) = -(1.0)*VOL(I,J)*DV(I,J,1)*&
EPS1(I,J)*N1*VORT(I,J)

end do
end do

end subroutine confinement1

subroutine bc_extrap(IMAX,JMAX,DV,CC1,CC2)
!---SUBROUTINE TASK(S)

!  1. Compute boundary conditions by extrapolating
!      from the interior.

implicit none

integer,intent(in) :: IMAX,JMAX
real,dimension(IMAX-1,JMAX-1,6),intent(inout) :: DV
real,dimension(IMAX-1,JMAX-1),intent(in) :: CC1,CC2
integer :: I,J

!bottom boundary

do I=3,IMAX-3
J=2
DV(I,J,1) = DV(I,J+1,1) + (DV(I,J+1,1)-DV(I,J+2,1))
/(CC2(I,J+1)-CC2(I,J+2))* &
(CC2(I,J)-CC2(I,J+1))
DV(I,J,2) = DV(I,J+1,2) + (DV(I,J+1,2)-DV(I,J+2,2))
/(CC2(I,J+1)-CC2(I,J+2))* &
(CC2(I,J)-CC2(I,J+1))
DV(I,J,3) = DV(I,J+1,3) + (DV(I,J+1,3)-DV(I,J+2,3))
/(CC2(I,J+1)-CC2(I,J+2))* &
\[ DV(I,J,4) = DV(I,J+1,4) + \frac{(DV(I,J+1,4)-DV(I,J+2,4))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J)-CC2(I,J+1)) \]

\[ DV(I,J,5) = DV(I,J+1,5) + \frac{(DV(I,J+1,5)-DV(I,J+2,5))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J)-CC2(I,J+1)) \]

\[ DV(I,J,6) = DV(I,J+1,6) + \frac{(DV(I,J+1,6)-DV(I,J+2,6))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J)-CC2(I,J+1)) \]

\[ DV(I,J-1,1) = DV(I,J+1,1) + \frac{(DV(I,J+1,1)-DV(I,J+2,1))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J-1)-CC2(I,J+1)) \]

\[ DV(I,J-1,2) = DV(I,J+1,2) + \frac{(DV(I,J+1,2)-DV(I,J+2,2))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J-1)-CC2(I,J+1)) \]

\[ DV(I,J-1,3) = DV(I,J+1,3) + \frac{(DV(I,J+1,3)-DV(I,J+2,3))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J-1)-CC2(I,J+1)) \]

\[ DV(I,J-1,4) = DV(I,J+1,4) + \frac{(DV(I,J+1,4)-DV(I,J+2,4))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J-1)-CC2(I,J+1)) \]

\[ DV(I,J-1,5) = DV(I,J+1,5) + \frac{(DV(I,J+1,5)-DV(I,J+2,5))}{(CC2(I,J+1)-CC2(I,J+2))} \times (CC2(I,J-1)-CC2(I,J+1)) \]
/(CC2(I,J+1)-CC2(I,J+2)) * &

(CC2(I,J-1)-CC2(I,J+1))

DV(I,J-1,6) = DV(I,J+1,6) + (DV(I,J+1,6)-DV(I,J+2,6)) &

/(CC2(I,J+1)-CC2(I,J+2)) * &

(CC2(I,J-1)-CC2(I,J+1))

end do

! right boundary

I=IMAX-2

 do J=3,JMAX-3

 DV(I,J,1) = DV(I-1,J,1) + (DV(I-1,J,1)-DV(I-2,J,1)) &

/(CC1(I-1,J)-CC1(I-2,J)) * &

(CC1(I,J)-CC1(I-1,J))

DV(I,J,2) = DV(I-1,J,2) + (DV(I-1,J,2)-DV(I-2,J,2)) &

/(CC1(I-1,J)-CC1(I-2,J)) * &

(CC1(I,J)-CC1(I-1,J))

DV(I,J,3) = DV(I-1,J,3) + (DV(I-1,J,3)-DV(I-2,J,3)) &

/(CC1(I-1,J)-CC1(I-2,J)) * &

(CC1(I,J)-CC1(I-1,J))

DV(I,J,4) = DV(I-1,J,4) + (DV(I-1,J,4)-DV(I-2,J,4)) &

/(CC1(I-1,J)-CC1(I-2,J)) * &

(CC1(I,J)-CC1(I-1,J))

DV(I,J,5) = DV(I-1,J,5) + (DV(I-1,J,5)-DV(I-2,J,5)) &

/(CC1(I-1,J)-CC1(I-2,J)) * &
\[ \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

\[ DV(I,J,6) = DV(I-1,J,6) + (DV(I-1,J,6) - DV(I-2,J,6)) \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

\[ DV(I+1,J,1) = DV(I-1,J,1) + (DV(I-1,J,1) - DV(I-2,J,1)) \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

\[ DV(I+1,J,2) = DV(I-1,J,2) + (DV(I-1,J,2) - DV(I-2,J,2)) \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

\[ DV(I+1,J,3) = DV(I-1,J,3) + (DV(I-1,J,3) - DV(I-2,J,3)) \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

\[ DV(I+1,J,4) = DV(I-1,J,4) + (DV(I-1,J,4) - DV(I-2,J,4)) \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

\[ DV(I+1,J,5) = DV(I-1,J,5) + (DV(I-1,J,5) - DV(I-2,J,5)) \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

\[ DV(I+1,J,6) = DV(I-1,J,6) + (DV(I-1,J,6) - DV(I-2,J,6)) \frac{(CC_1(I,J) - CC_1(I-1,J))}{(CC_1(I-1,J) - CC_1(I-2,J))} \]

end do
!top boundary

do I=3,IMAX-3

J=JMAX-2

DV(I,J,1) = DV(I,J-1,1) + (DV(I,J-1,1)-DV(I,J-2,1))&
/(CC2(I,J-1)-CC2(I,J-2))* &

(CC2(I,J)-CC2(I,J-1))

DV(I,J,2) = DV(I,J-1,2) + (DV(I,J-1,2)-DV(I,J-2,2))&
/(CC2(I,J-1)-CC2(I,J-2))* &

(CC2(I,J)-CC2(I,J-1))

DV(I,J,3) = DV(I,J-1,3) + (DV(I,J-1,3)-DV(I,J-2,3))&
/(CC2(I,J-1)-CC2(I,J-2))* &

(CC2(I,J)-CC2(I,J-1))

DV(I,J,4) = DV(I,J-1,4) + (DV(I,J-1,4)-DV(I,J-2,4))&
/(CC2(I,J-1)-CC2(I,J-2))* &

(CC2(I,J)-CC2(I,J-1))

DV(I,J,5) = DV(I,J-1,5) + (DV(I,J-1,5)-DV(I,J-2,5))&
/(CC2(I,J-1)-CC2(I,J-2))* &

(CC2(I,J)-CC2(I,J-1))

DV(I,J,6) = DV(I,J-1,6) + (DV(I,J-1,6)-DV(I,J-2,6))&
/(CC2(I,J-1)-CC2(I,J-2))* &

(CC2(I,J)-CC2(I,J-1))

DV(I,J+1,1) = DV(I,J-1,1) + (DV(I,J-1,1)-DV(I,J-2,1))&
\( \frac{(CC2(I,J-1) - CC2(I,J-2))}{(CC2(I+1,J) - CC2(I,J))} \)

\[ DV(I,J+1,2) = DV(I,J-1,2) + \frac{(DV(I,J-1,2) - DV(I,J-2,2))}{(CC2(I,J-1) - CC2(I,J-2))} \]

\( \frac{(CC2(I,J-1) - CC2(I,J-2))}{(CC2(I+1,J) - CC2(I,J))} \)

\[ DV(I,J+1,3) = DV(I,J-1,3) + \frac{(DV(I,J-1,3) - DV(I,J-2,3))}{(CC2(I,J-1) - CC2(I,J-2))} \]

\( \frac{(CC2(I,J-1) - CC2(I,J-2))}{(CC2(I+1,J) - CC2(I,J))} \)

\[ DV(I,J+1,4) = DV(I,J-1,4) + \frac{(DV(I,J-1,4) - DV(I,J-2,4))}{(CC2(I,J-1) - CC2(I,J-2))} \]

\( \frac{(CC2(I,J-1) - CC2(I,J-2))}{(CC2(I+1,J) - CC2(I,J))} \)

\[ DV(I,J+1,5) = DV(I,J-1,5) + \frac{(DV(I,J-1,5) - DV(I,J-2,5))}{(CC2(I,J-1) - CC2(I,J-2))} \]

\( \frac{(CC2(I,J-1) - CC2(I,J-2))}{(CC2(I+1,J) - CC2(I,J))} \)

\[ DV(I,J+1,6) = DV(I,J-1,6) + \frac{(DV(I,J-1,6) - DV(I,J-2,6))}{(CC2(I,J-1) - CC2(I,J-2))} \]

end do

!left boundary

I=2

do J=3,JMAX-3

\[ DV(I,J,1) = DV(I+1,J,1) + \frac{(DV(I+1,J,1) - DV(I+2,J,1))}{(CC1(I+1,J) - CC1(I+2,J))} \]

147
(CC1(I,J)-CC1(I+1,J))

DV(I,J,2) = DV(I+1,J,2) + (DV(I+1,J,2)-DV(I+2,J,2))
/(CC1(I+1,J)-CC1(I+2,J)) \* &

(CC1(I,J)-CC1(I+1,J))

DV(I,J,3) = DV(I+1,J,3) + (DV(I+1,J,3)-DV(I+2,J,3))
/(CC1(I+1,J)-CC1(I+2,J)) \* &

(CC1(I,J)-CC1(I+1,J))

DV(I,J,4) = DV(I+1,J,4) + (DV(I+1,J,4)-DV(I+2,J,4))
/(CC1(I+1,J)-CC1(I+2,J)) \* &

(CC1(I,J)-CC1(I+1,J))

DV(I,J,5) = DV(I+1,J,5) + (DV(I+1,J,5)-DV(I+2,J,5))
/(CC1(I+1,J)-CC1(I+2,J)) \* &

(CC1(I,J)-CC1(I+1,J))

DV(I,J,6) = DV(I+1,J,6) + (DV(I+1,J,6)-DV(I+2,J,6))
/(CC1(I+1,J)-CC1(I+2,J)) \* &

(CC1(I,J)-CC1(I+1,J))

DV(I-1,J,1) = DV(I+1,J,1) + (DV(I+1,J,1)-DV(I+2,J,1))
/(CC1(I+1,J)-CC1(I+2,J)) \* &

(CC1(I-1,J)-CC1(I+1,J))

DV(I-1,J,2) = DV(I+1,J,2) + (DV(I+1,J,2)-DV(I+2,J,2))
/(CC1(I+1,J)-CC1(I+2,J)) \* &

(CC1(I-1,J)-CC1(I+1,J))

DV(I-1,J,3) = DV(I+1,J,3) + (DV(I+1,J,3)-DV(I+2,J,3))

148
$\frac{(CC1(I+1,J)-CC1(I+2,J))}{(CC1(I-1,J)-CC1(I+1,J))}$

$DV(I-1,J,4) = DV(I+1,J,4) + \frac{(DV(I+1,J,4)-DV(I+2,J,4))}{(CC1(I+1,J)-CC1(I+2,J))}$

$\frac{(CC1(I+1,J)-CC1(I+2,J))}{(CC1(I-1,J)-CC1(I+1,J))}$

$DV(I-1,J,5) = DV(I+1,J,5) + \frac{(DV(I+1,J,5)-DV(I+2,J,5))}{(CC1(I+1,J)-CC1(I+2,J))}$

$\frac{(CC1(I+1,J)-CC1(I+2,J))}{(CC1(I-1,J)-CC1(I+1,J))}$

$DV(I-1,J,6) = DV(I+1,J,6) + \frac{(DV(I+1,J,6)-DV(I+2,J,6))}{(CC1(I+1,J)-CC1(I+2,J))}$

end do

!bottom left corner

$DV(2,2,1) = 0.5*(DV(3,2,1) + DV(2,3,1))$

$DV(2,2,2) = 0.5*(DV(3,2,2) + DV(2,3,2))$

$DV(2,2,3) = 0.5*(DV(3,2,3) + DV(2,3,3))$

$DV(2,2,4) = 0.5*(DV(3,2,4) + DV(2,3,4))$

$DV(2,2,5) = 0.5*(DV(3,2,5) + DV(2,3,5))$

$DV(2,2,6) = 0.5*(DV(3,2,6) + DV(2,3,6))$

$DV(2,1,1) = 0.5*(DV(3,1,1) + DV(2,2,1))$

$DV(2,1,2) = 0.5*(DV(3,1,2) + DV(2,2,2))$
\[ DV(2,1,3) = 0.5 \times (DV(3,1,3) + DV(2,2,3)) \]
\[ DV(2,1,4) = 0.5 \times (DV(3,1,4) + DV(2,2,4)) \]
\[ DV(2,1,5) = 0.5 \times (DV(3,1,5) + DV(2,2,5)) \]
\[ DV(2,1,6) = 0.5 \times (DV(3,1,6) + DV(2,2,6)) \]

\[ DV(1,2,1) = 0.5 \times (DV(1,3,1) + DV(2,2,1)) \]
\[ DV(1,2,2) = 0.5 \times (DV(1,3,2) + DV(2,2,2)) \]
\[ DV(1,2,3) = 0.5 \times (DV(1,3,3) + DV(2,2,3)) \]
\[ DV(1,2,4) = 0.5 \times (DV(1,3,4) + DV(2,2,4)) \]
\[ DV(1,2,5) = 0.5 \times (DV(1,3,5) + DV(2,2,5)) \]
\[ DV(1,2,6) = 0.5 \times (DV(1,3,6) + DV(2,2,6)) \]

\[ DV(1,1,1) = 0.5 \times (DV(2,1,1) + DV(1,2,1)) \]
\[ DV(1,1,2) = 0.5 \times (DV(2,1,2) + DV(1,2,2)) \]
\[ DV(1,1,3) = 0.5 \times (DV(2,1,3) + DV(1,2,3)) \]
\[ DV(1,1,4) = 0.5 \times (DV(2,1,4) + DV(1,2,4)) \]
\[ DV(1,1,5) = 0.5 \times (DV(2,1,5) + DV(1,2,5)) \]
\[ DV(1,1,6) = 0.5 \times (DV(2,1,6) + DV(1,2,6)) \]

!bottom right corner

\[ DV(IMAX-2,2,1) = 0.5 \times (DV(IMAX-3,2,1) + DV(IMAX-2,3,1)) \]
\[ DV(IMAX-2,2,2) = 0.5 \times (DV(IMAX-3,2,2) + DV(IMAX-2,3,2)) \]
\[ DV(IMAX-2,2,3) = 0.5 \times (DV(IMAX-3,2,3) + DV(IMAX-2,3,3)) \]
DV(IMAX-2,2,4) = 0.5*(DV(IMAX-3,2,4) + DV(IMAX-2,3,4))
DV(IMAX-2,2,5) = 0.5*(DV(IMAX-3,2,5) + DV(IMAX-2,3,5))
DV(IMAX-2,2,6) = 0.5*(DV(IMAX-3,2,6) + DV(IMAX-2,3,6))

DV(IMAX-2,1,1) = 0.5*(DV(IMAX-3,1,1) + DV(IMAX-2,2,1))
DV(IMAX-2,1,2) = 0.5*(DV(IMAX-3,1,2) + DV(IMAX-2,2,2))
DV(IMAX-2,1,3) = 0.5*(DV(IMAX-3,1,3) + DV(IMAX-2,2,3))
DV(IMAX-2,1,4) = 0.5*(DV(IMAX-3,1,4) + DV(IMAX-2,2,4))
DV(IMAX-2,1,5) = 0.5*(DV(IMAX-3,1,5) + DV(IMAX-2,2,5))
DV(IMAX-2,1,6) = 0.5*(DV(IMAX-3,1,6) + DV(IMAX-2,2,6))

DV(IMAX-1,2,1) = 0.5*(DV(IMAX-1,3,1) + DV(IMAX-2,2,1))
DV(IMAX-1,2,2) = 0.5*(DV(IMAX-1,3,2) + DV(IMAX-2,2,2))
DV(IMAX-1,2,3) = 0.5*(DV(IMAX-1,3,3) + DV(IMAX-2,2,3))
DV(IMAX-1,2,4) = 0.5*(DV(IMAX-1,3,4) + DV(IMAX-2,2,4))
DV(IMAX-1,2,5) = 0.5*(DV(IMAX-1,3,5) + DV(IMAX-2,2,5))
DV(IMAX-1,2,6) = 0.5*(DV(IMAX-1,3,6) + DV(IMAX-2,2,6))

DV(IMAX-1,1,1) = 0.5*(DV(IMAX-2,1,1) + DV(IMAX-1,2,1))
DV(IMAX-1,1,2) = 0.5*(DV(IMAX-2,1,2) + DV(IMAX-1,2,2))
DV(IMAX-1,1,3) = 0.5*(DV(IMAX-2,1,3) + DV(IMAX-1,2,3))
DV(IMAX-1,1,4) = 0.5*(DV(IMAX-2,1,4) + DV(IMAX-1,2,4))
DV(IMAX-1,1,5) = 0.5*(DV(IMAX-2,1,5) + DV(IMAX-1,2,5))
\[
DV(IMAX-1,1,6) = 0.5*(DV(IMAX-2,1,6) + DV(IMAX-1,2,6))
\]

!top right corner

\[
DV(IMAX-2,JMAX-2,1) = 0.5*(DV(IMAX-3,JMAX-2,1) + DV(IMAX-2,JMAX-3,1))
\]
\[
DV(IMAX-2,JMAX-2,2) = 0.5*(DV(IMAX-3,JMAX-2,2) + DV(IMAX-2,JMAX-3,2))
\]
\[
DV(IMAX-2,JMAX-2,3) = 0.5*(DV(IMAX-3,JMAX-2,3) + DV(IMAX-2,JMAX-3,3))
\]
\[
DV(IMAX-2,JMAX-2,4) = 0.5*(DV(IMAX-3,JMAX-2,4) + DV(IMAX-2,JMAX-3,4))
\]
\[
DV(IMAX-2,JMAX-2,5) = 0.5*(DV(IMAX-3,JMAX-2,5) + DV(IMAX-2,JMAX-3,5))
\]
\[
DV(IMAX-2,JMAX-2,6) = 0.5*(DV(IMAX-3,JMAX-2,6) + DV(IMAX-2,JMAX-3,6))
\]

\[
DV(IMAX-2,JMAX-1,1) = 0.5*(DV(IMAX-3,JMAX-1,1) + DV(IMAX-2,JMAX-2,1))
\]
\[
DV(IMAX-2,JMAX-1,2) = 0.5*(DV(IMAX-3,JMAX-1,2) + DV(IMAX-2,JMAX-2,2))
\]
\[
DV(IMAX-2,JMAX-1,3) = 0.5*(DV(IMAX-3,JMAX-1,3) + DV(IMAX-2,JMAX-2,3))
\]
\[
DV(IMAX-2,JMAX-1,4) = 0.5*(DV(IMAX-3,JMAX-1,4) + DV(IMAX-2,JMAX-2,4))
\]
\[
DV(IMAX-2,JMAX-1,5) = 0.5*(DV(IMAX-3,JMAX-1,5) + DV(IMAX-2,JMAX-2,5))
\]
\[
DV(IMAX-2,JMAX-1,6) = 0.5*(DV(IMAX-3,JMAX-1,6) + DV(IMAX-2,JMAX-2,6))
\]
DV(IMAX-2, JMAX-1, 5) = 0.5*(DV(IMAX-3, JMAX-1, 5) &
+ DV(IMAX-2, JMAX-2, 5))

DV(IMAX-2, JMAX-1, 6) = 0.5*(DV(IMAX-3, JMAX-1, 6) &
+ DV(IMAX-2, JMAX-2, 6))

DV(IMAX-1, JMAX-2, 1) = 0.5*(DV(IMAX-1, JMAX-3, 1) &
+ DV(IMAX-2, JMAX-2, 1))

DV(IMAX-1, JMAX-2, 2) = 0.5*(DV(IMAX-1, JMAX-3, 2) &
+ DV(IMAX-2, JMAX-2, 2))

DV(IMAX-1, JMAX-2, 3) = 0.5*(DV(IMAX-1, JMAX-3, 3) &
+ DV(IMAX-2, JMAX-2, 3))

DV(IMAX-1, JMAX-2, 4) = 0.5*(DV(IMAX-1, JMAX-3, 4) &
+ DV(IMAX-2, JMAX-2, 4))

DV(IMAX-1, JMAX-2, 5) = 0.5*(DV(IMAX-1, JMAX-3, 5) &
+ DV(IMAX-2, JMAX-2, 5))

DV(IMAX-1, JMAX-2, 6) = 0.5*(DV(IMAX-1, JMAX-3, 6) &
+ DV(IMAX-2, JMAX-2, 6))

DV(IMAX-1, JMAX-1, 1) = 0.5*(DV(IMAX-2, JMAX-1, 1) &
+ DV(IMAX-1, JMAX-2, 1))

DV(IMAX-1, JMAX-1, 2) = 0.5*(DV(IMAX-2, JMAX-1, 2) &
+ DV(IMAX-1, JMAX-2, 2))
\[ DV(IMAX-1,JMAX-1,3) = 0.5*(DV(IMAX-2,JMAX-1,3) \quad + \quad DV(IMAX-1,JMAX-2,3)) \]
\[ DV(IMAX-1,JMAX-1,4) = 0.5*(DV(IMAX-2,JMAX-1,4) \quad + \quad DV(IMAX-1,JMAX-2,4)) \]
\[ DV(IMAX-1,JMAX-1,5) = 0.5*(DV(IMAX-2,JMAX-1,5) \quad + \quad DV(IMAX-1,JMAX-2,5)) \]
\[ DV(IMAX-1,JMAX-1,6) = 0.5*(DV(IMAX-2,JMAX-1,6) \quad + \quad DV(IMAX-1,JMAX-2,6)) \]

!top left corner
\[ DV(2,JMAX-2,1) = 0.5*(DV(2,JMAX-3,1) + DV(3,JMAX-2,1)) \]
\[ DV(2,JMAX-2,2) = 0.5*(DV(2,JMAX-3,2) + DV(3,JMAX-2,2)) \]
\[ DV(2,JMAX-2,3) = 0.5*(DV(2,JMAX-3,3) + DV(3,JMAX-2,3)) \]
\[ DV(2,JMAX-2,4) = 0.5*(DV(2,JMAX-3,4) + DV(3,JMAX-2,4)) \]
\[ DV(2,JMAX-2,5) = 0.5*(DV(2,JMAX-3,5) + DV(3,JMAX-2,5)) \]
\[ DV(2,JMAX-2,6) = 0.5*(DV(2,JMAX-3,6) + DV(3,JMAX-2,6)) \]

\[ DV(1,JMAX-2,1) = 0.5*(DV(1,JMAX-3,1) + DV(2,JMAX-2,1)) \]
\[ DV(1,JMAX-2,2) = 0.5*(DV(1,JMAX-3,2) + DV(2,JMAX-2,2)) \]
\[ DV(1,JMAX-2,3) = 0.5*(DV(1,JMAX-3,3) + DV(2,JMAX-2,3)) \]
\[ DV(1,JMAX-2,4) = 0.5*(DV(1,JMAX-3,4) + DV(2,JMAX-2,4)) \]
\[ DV(1,JMAX-2,5) = 0.5*(DV(1,JMAX-3,5) + DV(2,JMAX-2,5)) \]
\[ DV(1,JMAX-2,6) = 0.5*(DV(1,JMAX-3,6) + DV(2,JMAX-2,6)) \]
\[ DV(2, J\text{MAX}-1, 1) = 0.5 \times (DV(3, J\text{MAX}-1, 1) + DV(2, J\text{MAX}-2, 1)) \]
\[ DV(2, J\text{MAX}-1, 2) = 0.5 \times (DV(3, J\text{MAX}-1, 2) + DV(2, J\text{MAX}-2, 2)) \]
\[ DV(2, J\text{MAX}-1, 3) = 0.5 \times (DV(3, J\text{MAX}-1, 3) + DV(2, J\text{MAX}-2, 3)) \]
\[ DV(2, J\text{MAX}-1, 4) = 0.5 \times (DV(3, J\text{MAX}-1, 4) + DV(2, J\text{MAX}-2, 4)) \]
\[ DV(2, J\text{MAX}-1, 5) = 0.5 \times (DV(3, J\text{MAX}-1, 5) + DV(2, J\text{MAX}-2, 5)) \]
\[ DV(2, J\text{MAX}-1, 6) = 0.5 \times (DV(3, J\text{MAX}-1, 6) + DV(2, J\text{MAX}-2, 6)) \]
\[ DV(1, J\text{MAX}-1, 1) = 0.5 \times (DV(1, J\text{MAX}-2, 1) + DV(2, J\text{MAX}-1, 1)) \]
\[ DV(1, J\text{MAX}-1, 2) = 0.5 \times (DV(1, J\text{MAX}-2, 2) + DV(2, J\text{MAX}-1, 2)) \]
\[ DV(1, J\text{MAX}-1, 3) = 0.5 \times (DV(1, J\text{MAX}-2, 3) + DV(2, J\text{MAX}-1, 3)) \]
\[ DV(1, J\text{MAX}-1, 4) = 0.5 \times (DV(1, J\text{MAX}-2, 4) + DV(2, J\text{MAX}-1, 4)) \]
\[ DV(1, J\text{MAX}-1, 5) = 0.5 \times (DV(1, J\text{MAX}-2, 5) + DV(2, J\text{MAX}-1, 5)) \]
\[ DV(1, J\text{MAX}-1, 6) = 0.5 \times (DV(1, J\text{MAX}-2, 6) + DV(2, J\text{MAX}-1, 6)) \]

end subroutine bc_extrap