Chapter Four: Numerical Procedure

Basics

Because the purpose of the current investigation is to explore the current state of the art in computational fluid dynamics (CFD) for mixing problems in supersonic flow, a decision was made to use a readily available flow solver package instead of writing a code expressly for the investigation. In this way much difficulty was avoided, not only in writing the code but also in separating the effects of errors in implementation from those of fundamental weaknesses of the solution scheme. That is to say, a flow solver package in wide use and thoroughly documented by its creators and other users is less likely to contain unknown errors than a new, untested code used only by its one creator. In addition, a widely used flow solver package is more likely to be representative of the state of the art than a research-level code written for a single application. To this end, all calculations were performed using GASP version 2.2, a commercial flow solver package created by AeroSoft, Incorporated.

GASP v.2.2 is a complex flow solver written in the FORTRAN and C programming languages. It contains 584 subroutines and more than 88,000 lines of code[48]. The fundamental purpose of the package is to solve the time-dependent, three-dimensional, chemically reacting Favre-Averaged Navier-Stokes Equations, though it can also solve subsets such as the Thin-Layer Navier-Stokes Equations, Parabolized Navier-Stokes Equations, the Euler Equations, and two-dimensional and axi-symmetric problems[48]. GASP v.2.2 is a cell-centered, finite volume code, fully conservative and shock-capturing. It contains a number of flow solver options, including six thermodynamic models, at least three flux-split options (in addition to the full flux option), six limiters, more than twenty chemistry models, and four time integration schemes[48]. Of particular interest are the four turbulence models, consisting of the Baldwin-Lomax algebraic model and three versions of the two-equation $K-\varepsilon$ model.

A number of computers were employed for this research project. Preliminary calculations were performed on the National Aerodynamic Simulator's Von Neumann machine, a Cray C-90, and AeroSoft, Inc.'s Trex, a Cray EL98. Final data was produced on three Department of Defense Cray C-90's, all part of the Major Shared Resource Center. Namely, these machines are the Navy's VLSC, the Air Force's HPC01, and the Army's PK. Though each machine is capable of processing multi-tasking jobs, no multi-tasking was employed in this investigation.

Renormalized Group Theory Additions to GASP™

Because GASP v.2.2 is such a complete flow solver package, the only significant modification that was necessary to accommodate the present investigation was the addition of the turbulence models derived from renormalized group theory (RNG) and discussed in the preceding chapter. As mentioned in the previous discussion, the RNG turbulence models implemented and used in the present investigation are two-equation $K-\varepsilon$ models very similar to the "standard" $K-\varepsilon$ model, so that the new models could be created by modification of the old instead of construction of extensive new routines. A total of ten new subroutines was sufficient to incorporate the three new turbulence models. An annotated listing of the new subroutines is provided in Appendix One. A brief discussion of each is provided below.

Because GASP v.2.2 performs calculations one plane at a time even for three-dimensional problems and because the logical direction of that plane (i, j, or k) can vary from zone to zone in the same problem, some subroutines must be created in triplicate, with a different subroutine for each planar direction. Included in the ten subroutines created for the RNG models are two such sets of three.

The fundamental building-blocks of the RNG additions to the turbulence production term are spatial first derivatives of the velocity components, and those are calculated in the new subroutines *derivsi*, *derivsj*, and *derivsk*. (*Derivsi* operates on an "i" plane,

derivsj performs identical calculations on a "j" plane, and so forth. In a given solution scheme only one is used.) The various *derivs* routines calculate and store the first spatial derivatives of all velocity components in all directions at all interior points plus the first row of ghost cells at each boundary, for a total of nine derivative calculations per cell. The ghost cell values are used in the calculation of higher-order derivatives in *lsttrmi*, *lsttrmj*, and *lsttrmk*. (*Lsttrm* is shortened from "last-term", a reference to the part of the turbulence production expression in which they are used.) In addition, the *derivs* routines extract values of K, the turbulence kinetic energy, and ε, the turbulence dissipation rate, from the stored q-vector at each cell and store them separately for later use. Cell metrics are reflected across boundaries to create values for ghost cells. All derivatives are second-order central differences.

Once the first derivatives have been stored, the calculation of all terms except the C_{R2} term in the RNG turbulence production expression is a simple matter. The subroutine *fsttrm* (from "first-terms") calculates the RNG additions to the turbulence production term except the C_{R2} term for both the turbulence kinetic energy and dissipation equations, as well as the corresponding turbulence Jacobian terms. Calculations are performed only at interior points (not at ghost cells), and the production terms are not multiplied by cell volume or density at this time. Density and volume effects are taken into account in the subroutine *addres*, which will be discussed shortly.

The subroutine triad *lsttrmi*, *lsttrmj*, and *lsttrmk* calculate turbulence production and Jacobian contributions for both equations due to the C_{R2} term in Equation 3.37. The strategy for calculating the higher derivatives in this term is to differentiate the stored first derivatives, which requires knowledge of more than one plane of data. Since operations are performed on one plane at a time and derivatives stored on previous planes are lost, the subroutine *derivs* (*i*, *j*, or *k*) must be called two additional times to obtain derivatives on adjacent planes before subroutine *lsttrm* (*i*, *j*, or *k*) can be called.

For example, if calculations are being performed on a plane $k = k_o$, *derivsk* must also be called for the planes k_o+1 and k_o-1 before *lsttrmk* can operate.

Within *lsttrm* (*i*, *j*, or *k*), the derivative strategy is much like that in *derivs* (*i*, *j*, or *k*). Central differences are used and metrics are reflected across boundaries, though the higher-order derivatives are needed (and, therefore, are calculated) only at interior points, not ghost cells. Once the derivatives are calculated the entire C_{R2} term can be assembled within the same routine. Subroutines *lsttrm* (*i*, *j*, and *k*) and *fsttrm* use for the "constant" coefficients the values given in Chapter Three.

The difference between “full” RNG and “fast” RNG is that “fast” RNG neglects the C_{R2} term in the production expression. As a result, “fast” RNG has no need to call any of the *lsttrm* subroutines and calls subroutine *derivs* only once instead of three times. The savings in computational costs should be evident.

Once the RNG contribution to the production term in the turbulence equations has been calculated, it must be added to the production term calculated elsewhere by the existing code. Subroutine *addres* multiplies the RNG production-term contributions by the cell volume and density, as required, then adds them to the residual for the turbulence kinetic energy and dissipation-rate equations. Subroutine *addjac* multiplies the RNG Jacobian contributions by density and volume, then adds them into the Jacobians already stored in a multi-dimensional array. Both subroutines must accommodate established sign definitions in the code. *Addjac* is only called if input flags are set to calculate turbulence Jacobians.

The one remaining subroutine is *modmut*, the subroutine used in "mixing" RNG to calculate a modified turbulence viscosity. The calculations follow Equation 3.35. In the event that the denominator of Equation 3.35 is equal to zero, the "modified" turbulence viscosity is equal to the "original" turbulence viscosity calculated by Equation 3.37. This routine is called only by the "mixing" RNG variation.

There is certainly room for improvement of the RNG subroutines, most particularly in the area of coding efficiency. Subroutines *fsttrm* and *lsttrm* (*i*, *j*, and *k*) involve extensive, nested loops and summations, and the efficiency of the looping strategy is almost certainly not optimal. Moreover, the entire procedure for calculating the higher derivatives in *lsttrm* (*i*, *j*, and *k*) is expensive in both computer processor time and memory required, and a superior alternative surely exists. However, the coding as implemented suffices for the present investigation.

Integration and Interpolation

Two different integration techniques were used in this investigation: space marching and global iteration. Space marching is an efficient and inexpensive technique that is limited to a particular class of flows. Global iteration is more versatile but unfortunately much more expensive.

The space marching technique is based upon characteristic analysis of the Euler equations, which show that information in a flow propagates from a point in characteristic waves. In a one dimensional analysis, there are three waves that travel with speeds u, u+a, and u-a, where u is the flow speed and a is the local speed of sound. If the local flow speed is everywhere greater than the local speed of sound, all the wave speeds have the same algebraic sign as the local flow speed, and information will propagate only in the direction of the flow, not against it. As a result, the flow properties at any given point will depend only on flow conditions upstream of that point and will be independent of the solution downstream. If the flow is steady one can then calculate the flow by beginning at the upstream end and converging the solution one point at a time while "marching" downstream, using the solution at each station as an upstream boundary condition for the next station downstream. The technique can be extended to multiple dimensions, as well.

So long as the flow to be solved is inviscid, the only requirement for space-marching is that the flowfield be supersonic

in the sweep direction. For viscous problems, the presence of subsonic flow in boundary layers complicates the matter and requires additional constraints. In addition to supersonic flow in the inviscid portion of the flowfield, there must be no streamwise separation of the boundary layer and the streamwise pressure gradient in the subsonic portion of the boundary layer must be suppressed or its influence on the streamwise momentum equation controlled in some fashion[66]. If these requirements plus the steadiness requirement are met, the unsteady and streamwise diffusion terms can be omitted from the Navier-Stokes equations to form the Parabolized Navier-Stokes equations, and space marching can be employed[66]. Under such circumstances the number of simultaneous equations to be solved decreases dramatically from that of a general, three-dimensional solution, and computer time and memory requirements are greatly reduced. For a full discussion of space marching techniques, see Reference 66.

A number of different researchers have considered the problem of the streamwise pressure gradient in space-marching situations. If included in subsonic portions of the flowfield, the pressure gradient will allow information to propagate upstream, invalidating the space-marching technique and changing the mathematical nature of the equations. If the pressure gradient is completely suppressed solutions can possibly be obtained, but not without the introduction of errors, which can be large in flows with strong gradients. The compromise employed in the present investigation is the Vigernon technique[67], which separates the pressure gradient into two parts. A fraction of the pressure gradient is included normally in the calculation, and the rest is omitted or treated as a source-term with a backward difference. An eigenvalue analysis of the resulting governing equations is used to determine how large a fraction can be included while still guarenteeing a parabolic equation set.

Flow situations not known to meet the criteria for space marching must be globally iterated; that is, simultaneous solutions

must be evolved at all points throughout the three-dimensional computational domain. While global iteration is usually several times more expensive in calculation time and computer memory than space marching, it is essential for physically realistic solutions when information is free to propagate in all directions. In the present investigation, space marching is possible in zones upstream of the injection zones and in those far downstream of the primary mixing region. Global iteration must be used in the zones in which injection takes place, because of the possibility of reversed flow.

The time integration scheme used in the present investigation was the well-known Euler Implicit scheme, which functions in the following manner: The governing equations presented and discussed in Chapter Three can be written symbolically as

$$\frac{\partial \overline{Q}}{\partial t} + \overline{R}(\overline{q}) = 0 \tag{4.1}$$

where $\overline{R}(\overline{q})$ is the residual vector. The exact definition of the residual will be discussed shortly. (Notice that the residual is a measure of the deviation of the flow properties from steady state. As such it can be a useful tool in measuring convergence, as will be discussed in a later chapter.) $\overline{Q}$ is the vector of conservative variables,

$$\overline{Q} = \left\{ \overline{\rho}_1, \overline{\rho}_2, ..., \overline{\rho}\,\tilde{U}, \overline{\rho}\,\tilde{V}, \overline{\rho}\,\tilde{W}, \overline{\rho}\,\tilde{e}_o \right\}^T \tag{4.2}$$

and $\overline{q}$ is the vector of primitive variables,

$$\overline{q} = \left\{ \overline{\rho}_1, \overline{\rho}_2, ..., \tilde{U}, \tilde{V}, \tilde{W}, \overline{P} \right\}^T \tag{4.3}$$

(For the sake of clarity and generalization, variables and equations relating to differential-equation turbulence modeling are not included in vector or matrix definitions in the present discussion. When differential equations are used for turbulence calculations, they are added to the vector of variables and fluxes in a manner directly analogous to that presented here.)

Euler Implicit integration expresses Equation 4.1 as

$$\left.\frac{\partial \overline{Q}}{\partial \overline{q}}\right|^n \frac{\Delta^n \overline{q}}{\Delta t} + \overline{R}\left(\overline{q}^n\right) + \left.\frac{\partial \overline{R}}{\partial \overline{q}}\right|^n \Delta^n \overline{q} = 0 \tag{4.4}$$

The superscripts indicate function evaluations at time-level n, and Δ^n indicates a forward difference in time. Equation 4.4 can then be written as

$$\left(\left.\frac{\partial \overline{Q}}{\partial \overline{q}}\right|^n \frac{1}{\Delta t} + \left.\frac{\partial \overline{R}}{\partial \overline{q}}\right|^n \right) \Delta^n \overline{q} = -\overline{R}\left(\overline{q}^n\right) \tag{4.5}$$

and solved for $\Delta^n \overline{q}$. The updated vector of primitive variables is then $\overline{q}^{n+1} = \overline{q}^n + \Delta^n \overline{q}$.

Euler Implicit integration is first-order accurate in time, a fact that is of little relevance in the calculation of steady-state conditions. Both $\partial \overline{Q}/\partial \overline{q}$ and $\partial \overline{R}/\partial \overline{q}$ are square matrices equal in size to the number of primitive variables. Equation 4.5 is applied at each cell in the computational domain, so that the overall size of the matrix to be inverted is equal to the number of primitive variables times the number of cells being solved simultaneously.

Whether solving simultaneously for flow variables at all points in the three-dimensional domain or considering only one plane of data, the matrix created from the linearized governing equations (Equation 4.5) is usually too large for direct inversion. Consequently, many if not most CFD solution schemes use some approximation or relaxation technique to reduce the size of the problem. A two-factor approximate factorization technique was chosen to eliminate the need for large matrix inversions.

An understanding of approximate factorization begins with a closer look at Equation 4.1 and a definition of the residual.

$$\overline{R}(\overline{q}) = \frac{\partial}{\partial x}\left(\overline{F} - \overline{F}_v\right) + \frac{\partial}{\partial y}\left(\overline{G} - \overline{G}_v\right) + \frac{\partial}{\partial z}\left(\overline{H} - \overline{H}_v\right) \tag{4.6}$$

for nonreacting flows (so that there are no source terms in the governing equations.) $\overline{F}$, $\overline{G}$, and $\overline{H}$ are the inviscid flux vectors in each coordinate direction, defined by

$$\overline{F}=\begin{Bmatrix}\overline{\rho}_1\tilde{U}\\ \overline{\rho}_2\tilde{U}\\ \cdots\\ \overline{\rho}\tilde{U}^2+\overline{P}\\ \overline{\rho}\tilde{U}\tilde{V}\\ \overline{\rho}\tilde{U}\tilde{W}\\ \overline{\rho}\tilde{U}\tilde{h}_o\end{Bmatrix}\qquad \overline{G}=\begin{Bmatrix}\overline{\rho}_1\tilde{V}\\ \overline{\rho}_2\tilde{V}\\ \cdots\\ \overline{\rho}\tilde{U}\tilde{V}\\ \overline{\rho}\tilde{V}^2+\overline{P}\\ \overline{\rho}\tilde{V}\tilde{W}\\ \overline{\rho}\tilde{V}\tilde{h}_o\end{Bmatrix}\qquad \overline{H}=\begin{Bmatrix}\overline{\rho}_1\tilde{W}\\ \overline{\rho}_2\tilde{W}\\ \cdots\\ \overline{\rho}\tilde{U}\tilde{W}\\ \overline{\rho}\tilde{V}\tilde{W}\\ \overline{\rho}\tilde{W}^2+\overline{P}\\ \overline{\rho}\tilde{W}\tilde{h}_o\end{Bmatrix} \tag{4.7}$$

$\overline{F}_v$, $\overline{G}_v$, and $\overline{H}_v$ are the corresponding viscous flux vectors, defined as

$$\overline{F}_v=\begin{Bmatrix}\left(\frac{\mu_T}{Sc_T}+\frac{\mu}{Sc}\right)\frac{\partial}{\partial x}\left(\frac{\overline{\rho}_1}{\overline{\rho}}\right)\\ \left(\frac{\mu_T}{Sc_T}+\frac{\mu}{Sc}\right)\frac{\partial}{\partial x}\left(\frac{\overline{\rho}_2}{\overline{\rho}}\right)\\ \cdots\\ \overline{\tau}_{xx}+\overline{\tau}^R_{xx}\\ \overline{\tau}_{xy}+\overline{\tau}^R_{xy}\\ \overline{\tau}_{xz}+\overline{\tau}^R_{xz}\\ \left(\overline{\tau}_{xx}+\overline{\tau}^R_{xx}\right)\tilde{U}+\left(\overline{\tau}_{xy}+\overline{\tau}^R_{xy}\right)\tilde{V}+\left(\overline{\tau}_{xy}+\overline{\tau}^R_{xy}\right)\tilde{W}-q_x-\frac{\gamma\mu_T}{Pr_T}\frac{\partial\tilde{e}}{\partial x}+\left(\mu+\frac{\mu_T}{\sigma_K}\right)\tilde{K}\end{Bmatrix} \tag{4.8a}$$

$$\overline{G}_v=\begin{Bmatrix}\left(\frac{\mu_T}{Sc_T}+\frac{\mu}{Sc}\right)\frac{\partial}{\partial y}\left(\frac{\overline{\rho}_1}{\overline{\rho}}\right)\\ \left(\frac{\mu_T}{Sc_T}+\frac{\mu}{Sc}\right)\frac{\partial}{\partial y}\left(\frac{\overline{\rho}_2}{\overline{\rho}}\right)\\ \cdots\\ \overline{\tau}_{yx}+\overline{\tau}^R_{yx}\\ \overline{\tau}_{yy}+\overline{\tau}^R_{yy}\\ \overline{\tau}_{yz}+\overline{\tau}^R_{yz}\\ \left(\overline{\tau}_{yx}+\overline{\tau}^R_{yx}\right)\tilde{U}+\left(\overline{\tau}_{yy}+\overline{\tau}^R_{yy}\right)\tilde{V}+\left(\overline{\tau}_{yz}+\overline{\tau}^R_{yz}\right)\tilde{W}-q_y-\frac{\gamma\mu_T}{Pr_T}\frac{\partial\tilde{e}}{\partial y}+\left(\mu+\frac{\mu_T}{\sigma_K}\right)\tilde{K}\end{Bmatrix} \tag{4.8b}$$

and

$$\overline{H}_v = \begin{Bmatrix} \left(\frac{\mu_T}{Sc_T}+\frac{\mu}{Sc}\right)\frac{\partial}{\partial z}\left(\frac{\overline{\rho}_1}{\overline{\rho}}\right) \\ \left(\frac{\mu_T}{Sc_T}+\frac{\mu}{Sc}\right)\frac{\partial}{\partial z}\left(\frac{\overline{\rho}_2}{\overline{\rho}}\right) \\ \cdots \\ \overline{\tau}_{zx}+\overline{\tau}^R_{zx} \\ \overline{\tau}_{zy}+\overline{\tau}^R_{zy} \\ \overline{\tau}_{zz}+\overline{\tau}^R_{zz} \\ \left(\overline{\tau}_{zx}+\overline{\tau}^R_{zx}\right)\tilde{U}+\left(\overline{\tau}_{zy}+\overline{\tau}^R_{zy}\right)\tilde{V}+\left(\overline{\tau}_{zz}+\overline{\tau}^R_{zz}\right)\tilde{W}-q_z-\frac{\gamma\mu_T}{Pr_T}\frac{\partial\tilde{e}}{\partial z}+\left(\mu+\frac{\mu_T}{\sigma_K}\right)\tilde{K} \end{Bmatrix} \quad (4.8c)$$

Notice that the definitions of the viscous flux vectors presented above incorporate the various flow models and simplifications introduced and discussed in Chapter 3. Among the terms presented in modeled form are the laminar and turbulent mass diffusion terms in the species continuity equations and the turbulent heat flux vector. A complete discussion of the origins and significance of this form of the governing equations can be found in Reference 41 and almost any other thorough CFD text.

Using the definition of the residual given in Equation 4.6, Equation 4.5 can be written in semi-discrete form as

$$\left(\frac{1}{\Delta t}\frac{\partial\overline{Q}}{\partial\overline{q}}+\frac{\partial}{\partial x}\left(\frac{\partial\left(\overline{F}-\overline{F}_v\right)}{\partial\overline{q}}\right)+\frac{\partial}{\partial y}\left(\frac{\partial\left(\overline{G}-\overline{G}_v\right)}{\partial\overline{q}}\right)+\frac{\partial}{\partial z}\left(\frac{\partial\left(\overline{H}-\overline{H}_v\right)}{\partial\overline{q}}\right)\right)\Delta^n\overline{q}=-\overline{R}(\overline{q}) \quad (4.9)$$

with all functions being evaluated at time-level n.

Relaxation in one direction, say the "z" direction, with sweeping in the positive “k” direction (where the index “k” varies in the “z” direction) treats the flux vectors in that direction, $\overline{H}$ and $\overline{H}_v$ in this case, implicitly in the Gauss-Seidel sense as follows: The equation set is to be solved on a given plane k for values of the primitive-variable vector $\overline{q}$ at time-level n+1. Values of these variables at time-level n+1 are already known for previous planes k-1, k-2, etc. Updated (i.e., time-level n+1) values are not known for planes k+1, k+2, etc. Because the updated (n+1) values are already known for planes k-1, k-2, etc., the portions of the residual vector requiring data from

those planes can be calculated using the updated information directly, and the $\Delta^n\bar{q}$ terms evaluated on planes k-1, k-2, etc. drop out of the left hand side of Equation 4.9. Because updated values are not known for planes k+1, k+2, etc., $\bar{q}^{n+1}$ are approximated as $\bar{q}^n$, and the $\Delta^n\bar{q}$ terms evaluated on planes k+1, k+2, etc. are set to zero. The result is

$$\left(M + \frac{\partial}{\partial x}\left(\frac{\partial\left(\bar{F} - \bar{F}_v\right)}{\partial \bar{q}} \right) + \frac{\partial}{\partial y}\left(\frac{\partial\left(\bar{G} - \bar{G}_v\right)}{\partial \bar{q}} \right) \right) \Delta^n \bar{q} = -\bar{R}\left(\bar{q}^n, \bar{q}^{n+1}\right) \tag{4.10}$$

where

$$M = \frac{1}{\Delta t}\frac{\partial \bar{Q}}{\partial q} + \frac{\partial \bar{H}_{k+\frac{1}{2}}}{\partial q_k} + \frac{\partial \bar{H}_{k-\frac{1}{2}}}{\partial q_k} - \frac{\partial \bar{H}_{v_{k+\frac{1}{2}}}}{\partial q_k} - \frac{\partial \bar{H}_{v_{k-\frac{1}{2}}}}{\partial q_k} \tag{4.11}$$

The subscripts in the denominator indicate that only contributions corresponding to plane "k" are to be included. The notation $R\left(\bar{q}^n, \bar{q}^{n+1}\right)$ simply implies that all variables from planes k-1, k-2, etc., are to be evaluated at time-level n+1, whereas variables from planes k, k+1, etc., are to be evaluated at time-level n. In other words,

$$R\left(\bar{q}^n, \bar{q}^{n+1}\right) = R\left(\bar{q}^n_{i,j,k}, \bar{q}^n_{i\pm1,j,k}, \bar{q}^n_{i,j\pm1,k}, \bar{q}^n_{i\pm2,j,k}, \bar{q}^n_{i,j\pm2,k}, \bar{q}^{n+1}_{i,j,k-1}, \bar{q}^{n+1}_{i,j,k-2}, \bar{q}^n_{i,j,k+1}, \bar{q}^n_{i,j,k+2}\right) \tag{4.12}$$

Sweeping in the negative "k" direction reversed the roles of k+1, k+2 and k-1, k-2 in the preceding discussion. Two-factor approximate factorization further simplifies the system of governing equations by replacing Equation 4.10 with

$$\left[M + \frac{\partial}{\partial x}\left(\frac{\partial\left(\bar{F} - \bar{F}_v\right)}{\partial \bar{q}} \right) \right] M^{-1} \left[M + \frac{\partial}{\partial y}\left(\frac{\partial\left(\bar{G} - \bar{G}_v\right)}{\partial \bar{q}} \right) \right] \Delta^n \bar{q} = -\bar{R}(\bar{q}) \tag{4.13}$$

which differs from Equation 4.10 by the presence of a factorization error $\left[\frac{\partial}{\partial x}\left(\frac{\partial\left(\bar{F} - \bar{F}_v\right)}{\partial \bar{q}} \right) \right] M^{-1} \left[\frac{\partial}{\partial y}\left(\frac{\partial\left(\bar{G} - \bar{G}_v\right)}{\partial \bar{q}} \right) \right] \Delta^n \bar{q}$ on the left hand side. Equation 4.13, however, can be solved stepwise for $\Delta^n\bar{q}$ as follows:

$$\left[M + \frac{\partial}{\partial x}\left(\frac{\partial\left(\bar{F} - \bar{F}_v\right)}{\partial \bar{q}} \right) \right] \Delta^n \bar{q}^* = -\bar{R}(\bar{q}) \tag{4.14}$$

$$\left[M + \frac{\partial}{\partial y}\left(\frac{\partial\left(\bar{G} - \bar{G}_v\right)}{\partial \bar{q}} \right) \right] \Delta^n \bar{q} = M \Delta^n \bar{q}^* \tag{4.15}$$

Notice that each solution step is a one-dimensional calculation only, which is inherently less expensive than a multi-dimensional calculation.

For planar calculations such as those performed when space-marching, two-factor approximation was sufficient without the need for relaxation. For three-dimensional global calculations, the third dimension was relaxed as discussed above.

There are two main options for specifying the time increment to be used in the integration of the governing equations. Global timestepping is perhaps the most straight-forward, because it simply applies a given timestep to every computational cell in the domain. (Or at least, to all those being iterated together in a single calculation.) The drawback of global timestepping is that the timestep must be chosen conservatively enough to satisfy stability on the smallest (most sensitive) cells and is usually quite overconservative for the bulk of the computational domain. The alternative is local timestepping, a technique that specifies not the timestep itself but the CFL (Courant-Friedrichs-Levy) number. The CFL number, $CFL = s\Delta t/\Delta L$, arises from stability analysis of the governing equations and can be used to take into account the effect of cell size on stability. (Here s is the speed of the fastest characteristic velocity and ΔL is a characteristic length of the cell.) When specifying a constant CFL number, the actual timestep varies with the size of the cell. Smaller cells evolve more slowly in time to accommodate their stability requirements, but larger cells advance quickly in time, reaching a steady-state, converged solution sooner and (hopefully) accelerating overall convergence by providing a better solution with which the small cells can exchange data. Temporal accuracy is lost in local timestepping, so the technique is strictly applied only to steady-state flows, for which the temporal accuracy of a converged solution is irrelevant. Because local timestepping is easier to start and may provide quicker overall convergence, this technique was used in all calculations.

The formulation of the inviscid flux terms is critical to the stability of the computational scheme. Care must be taken to draw information primarily from the direction from which waves transporting that iformation physically flow, whether that be forward or backward. Consider an inviscid flux vector $\bar{F}(\bar{q})$ and its spatial difference $\frac{\partial}{\partial x}\bar{F}(\bar{q})$, where $\bar{q}$ is the vector of primitive variables. For a finite volume formulation, $\frac{\partial}{\partial x}\bar{F}(\bar{q})$ can be represented discretely by the central difference

$$\bar{\delta}_i \bar{F}(\bar{q}_i) = \left[\bar{F}\left(\bar{q}_{i+\frac{1}{2}}\right) - \bar{F}\left(\bar{q}_{i-\frac{1}{2}}\right)\right] / \Delta x \qquad (4.16)$$

The subscripts $i \pm \frac{1}{2}$ denote cell-face values that must be interpolated, since variables are stored at cell centers. For a given point in a given flow, the term $\bar{F}\left(\bar{q}_{i+\frac{1}{2}}\right)$ might be evaluated using information primarily from the left side of the cell face ($\bar{q}_i$ plus corrections) or from the right side of the cell face ($\bar{q}_{i+1}$ plus corrections), or from a combination of the two. The distinction must be based on the physical propagation of information-carrying waves at that point. Space-marched flows have a known positive flow direction and a known direction of information flow, and fully upwind fluxes are easily calculated using only left-state information. Therefore, inviscid fluxes in the marching direction in this investigation were treated with a full upwind flux calculation with the Vigernon technique needed for viscous implementation. Global iteration problems and inviscid flux calculations in planes other than that of space-marching are not nearly so clear-cut and require that the flow of information be checked at each point, and that the differencing scheme be adjusted as necessary. Such inviscid fluxes in this investigation were treated with an extension of Roe's flux-difference splitting scheme[48, 68], which is based on the solution of local Riemann problems.

The spatial order of the variable interpolation is important for the quality of the solution. The highest accuracy available in GASP v.2.2 for interpolation in space-marching directions is second order,

and for other directions (including global iteration) it is third order. Therefore, all inviscid flux calculations were locally third-order accurate (second-order globally) except for those in space-marching direction, which were second order.

In flow regions where sharp gradients are present, such higher order interpolation can introduce oscillations that interfere with convergence and negatively affect stability. To eliminate these oscillations and improve stability, limiters are often employed. Such limiters reduce the spatial order of the calculation in the vicinity of sharp gradients but allow higher order calculations where the flow is smoother. Consider any primitive variable q at cell i. We can interpolate a value at the $i+\frac{1}{2}$ cell boundary with information taken primarily from the left side of the boundary as

$$\left(q_L\right)_{i+\frac{1}{2}} = q_i + \frac{\phi}{4}\left[(1-\kappa)\nabla + (1+\kappa)\Delta\right]q_i \tag{4.17}$$

where

$$\Delta q_i = q_{i+1} - q_i \tag{4.18}$$

$$\nabla q_i = q_i - q_{i-1} \tag{4.19}$$

and κ and ϕ are constants that control the order of the interpolation. Similarly, we can interpolate a value at the $i-\frac{1}{2}$ cell boundary with information taken primarily from the right side of the boundary as

$$\left(q_R\right)_{i-\frac{1}{2}} = q_i - \frac{\phi}{4}\left[(1+\kappa)\nabla + (1-\kappa)\Delta\right]q_i \tag{4.20}$$

$\kappa = \frac{1}{3}$ and $\phi = 1$ correspond to a third-order flux. The order of interpolation can be reduced to first by setting ϕ to zero, and this technique could be used to limit oscillations at locations where they were known to be present. Many more sophisticated techniques involve replacing the constants κ and/or ϕ with functions of the local flow properties, so that the change in order occurs automatically when some criterion is met. One such limiter is Van Albada's limiter, which replaces ϕ with ϕ_{va}, where

$$\phi_{va} q = \frac{2\left(\Delta\nabla q + \varepsilon\right)}{\Delta^2 q + \nabla^2 q + 2\varepsilon} \tag{4.21}$$

and ε is a small number. Van Albada's limiter was used for nine-hole injector array calculations. The Min-Mod limiter, which was

used for the four-hole ramp calculations, functions somewhat differently. Instead of adjusting the constants κ and ϕ, the Min-Mod limiter replaces the difference operators Δ and ∇ with other operators $\overline{\Delta}$ and $\overline{\nabla}$, respectively. The new operators are defined as

$$\overline{\Delta}q = \min\mathrm{mod}(\Delta q, \beta\nabla q) \tag{4.22}$$

$$\overline{\nabla}q = \min\mathrm{mod}(\nabla q, \beta\Delta q) \tag{4.23}$$

where

$$\min\mathrm{mod}(a,b) = \mathrm{sign}(a) * \max(0, \min(a * \mathrm{sign}(b), b * \mathrm{sign}(a))) \tag{4.24}$$

for any values a and b. One can see that the minmod function returns a value of zero if the two input variables have differing signs. Otherwise, the minimum modulus of the two input variables is returned. The coefficient β is given by

$$\beta = \frac{3-\kappa}{1-\kappa} \tag{4.25}$$

Each of these limiters possesses its own unique virtues. The Min-Mod limiter is somewhat dissipative, but it is very robust and allows convergence to solutions in a wide variety of flow situations. The Van Albada limiter is sometimes less robust but is also less likely to introduce oscillations that can themselves interfere with smooth convergence. For these reasons the Min-Mod limiter proved to be useful with the four-hole ramp problem, for which stability of the starting solution was somewhat difficult to establish. The Van Albada limiter was employed for the nine-hole array problem, for which robustness was not so critical a requirement.

The Van Albada and Min-Mod limiters were used to limit flow oscillations in all logical directions except those in which space-marching was being performed. Because flow in space-marching directions is known to be unidirectional and data for interpolation is taken only from the upwind direction, the forward difference Δ is not available in those directions and Equations 4.17 and 4.20 can be applied only with $\phi = 0$ (for a first-order interpolation) or $\kappa = -1$ (for a second-order upwind interpolation). Since most conventional limiters, including Van Albada and Min-Mod, make use of Δ in some manner, a different type of limiter is needed for space marching.

One acceptable type of limiter simply checks certain flow variables at each point and limits interpolation to first order if nonphysical values are detected. The space-marching limiter used in the present investigation behaves in this manner and checks pressure and density for negative values. Two variations were used, one checking the mixture density and the other checking each species density separately. Because all calculations in this investigation were two-species problems (air and helium), and because the second species (injectant helium) was present in relatively small portions of the computational domain only, the two variations were expected to show very similar behavior.

All viscous derivatives in GASP v.2.2 are approximated using second-order central differences. On boundary faces, viscous gradients were calculated using second-order accurate one-sided differences.

Models and Constants

The multi-species Navier-Stokes equations given in Equations 3.1 through 3.4 are a set of n+4 coupled equations, where n is the number of chemical species. The unknowns are ρ_i, U_1, U_2, U_3, P, T, e, μ, D_{l_i}, and k, for a total of 2n+8. Obviously a number of additional relations are needed. Among them are an equation of state and a thermodynamics model to relate the thermodynamic variables P, T, and e. In addition, theoretical or empirical models must be included to specify values of μ, D_{l_i}, and k as functions of other flow variables. In the present investigation the equations and models were simplified by considering only two chemical species, each of which could be considered a calorically perfect gas. The first was a fictitious single species with the thermodynamic properties of a perfect-gas air mixture. The molecular weight was 28.97978 and reflects a composition of approximately 76% nitrogen (by mass), 23% oxygen (by mass), and trace elements. The second chemical species was helium, which was also considered to behave as a perfect gas. Because the current investigation was of cold flows with inert

injectants, no chemical reactions would take place and "frozen flow" conditions were used. The single-species air model was expected to allow adequate resolution of all relevant flow data without the expense of additional equations, and temperatures were expected to remain sufficiently low to justify the perfect gas assumption.

The remaining unknowns represent unspecified transport properties, and the system must be closed by providing expressions relating those properties to other flow variables. Sutherland's formula was used in this investigation for the coefficient of laminar viscosity, μ. Coefficients for each species were calculated using

$$\mu_i = T^{1.5} \frac{E_i}{T + F_i} \tag{4.26}$$

for each species i, and an overall mixture value μ was calculated from the species values using Wilke's rule, a semi-empirical extension to Sutherland's formula based in part on the kinetic theory of gases. E_i and F_i are empirical coefficients. A similar, Sutherland-like expression was used to obtain the coefficient of laminar thermal conductivity for each species, k_i.

$$k_i = T^{1.5} \frac{e_i}{T + f_i} \tag{4.27}$$

where e_i and f_i are again empirically derived constants. The mixture value for k was calculated from k_i using Wilke's rule.

Mass diffusion is a complex phenomenon that depends physically on a number of different factors, including the thermochemical nature of the diffusing particle and of the medium into which it diffuses, mass or mole concentration gradients, pressure and temperature gradients, and body forces. Unfortunately, realistic treatment of such complexity is beyond the scope of current calculations. Following the analysis presented in Chapter Three (See Equations 3.4-3.6, 3.13, and 3.17-3.19.), the present investigation considered all species diffusion coefficients D_{l_i} to be equal to each other and specified by

$$D_{l_i} = D_l = \frac{\mu}{\rho Sc} \tag{4.28}$$

Sc is the (laminar) Schmidt number, also considered to be constant for a given flow. For the present investigations Sc was set equal to 0.22, an empirical value for helium diffusion into air.[69]

As discussed in Chapter Three above, Favre-averaging the Navier-Stokes equations for use with turbulent flows results in the modeling of several intractable terms. Three of these, the turbulent heat flux, the turbulent diffusion, and the molecular diffusion plus turbulent transport terms require the use of "adjustable constants". In the turbulent heat flux term the adjustable constant is the turbulent thermal conductivity, k_t, which is specified via a turbulent Prandtl number,

$$k_t = \frac{\mu_t C_p}{Pr_t} \tag{4.29}$$

In the present investigation the turbulent Prandtl number Pr_t was considered a constant equal to 0.9, a common value recommended by several sources for good comparison with experiment in problems of this type (e.g., References 6 and 48). The turbulent diffusion parameter D_t was defined by the turbulent viscosity (calculated by the turbulence model) and a constant turbulent Schmidt number in a manner analogous to the laminar diffusion. The turbulent Schmidt number Sc_t was set equal to 0.50 in the relationship

$$D_t = \frac{\mu_t}{\rho\, Sc_t} \tag{4.30}$$

Once again this value is recommended by several sources (e.g. References 44 and 48) as appropriate for calculations of flows of this type. The parameter σ_K in the molecular diffusion and turbulent transport expression was set to 1.0, following References 45 and 46.

Grids and Mesh Sequencing

One of the first and most time-consuming steps in any CFD solution process is the generation of suitable grids. In the ideal case the final solution should be independent of the grid used in its calculation; however in complex flow situations such as these independence is seldom proven. Proving true grid independence (or

grid convergence) requires extremely dense meshes, especially for highly viscous, complex flows, and even with presently available computers a sufficiently large number of grid points is seldom feasible. Care must be taken to avoid errors due to minimal resolution, if for no other reason than to prevent their being mistaken for errors due to turbulence modeling[4]. At the same time, the number of grid points used in any calculation must be low enough to fit the available computer memory and processor time. While many researchers acknowledge that full grid convergence is unlikely given current resources (e.g., Reference 1), the approach used in this investigation is to attempt grid convergence and to use mesh sequencing to illustrate its achievement or lack thereof.

Most of the gridding work for this investigation was done using the GridGen software package, versions nine and eleven, which is marketed by Pointwise, Incorporated[70]. Stretching was accomplished via an inverse hyperbolic tangent function. Both elliptic and parabolic stretching were employed.

All the injector nozzles studied in the present investigation are physically elliptical, and care was taken during gridding to preserve their elliptical shape. While this deviation from rectangular cells does slightly complicate the process of grid creation and increase the truncation error of the calculation[71], it was deemed necessary for two reasons. First, there is clear evidence that rectangular nozzles show different mixing characteristics than elliptical ones in some physical tests. (See References 20 - 25.) Current limitations on grid resolution may or may not allow for computed solutions that adequately capture these differences, but given the option, the conservative and careful choice is to use a more physically accurate grid. Second, as the state of the art for CFD advances, as newer computers allow larger grids and higher accuracy in solution schemes, and as CFD is applied to a wider class of flows, purely rectangular grids will almost certainly become too simple for general use, if in fact they are not already. Therefore, purely rectangular

grids were not considered realistic test cases for state-of-the-art turbulent flow calculations.

Wall Injector Array:

Before the grid could be generated, the domain to be modeled had to be chosen. The computational region to be modeled was required to extend downstream to the measurement station, but the restrictions on placement of the other faces of the grid were less concrete. Upstream of the injectors two options were available: either the domain could extend far enough upstream to allow the development of a boundary layer from a computational "leading edge" where none was present, or actual boundary layer velocity profiles could have been specified as a fixed inflow closer to the injectors. Since no boundary layer data was available to specify as fixed inflow, the grid was extended upstream to allow the development of a boundary layer. Spanwise, an assumption of crossflow symmetry allowed the modeling of only one-half of the wall injector array. Therefore, one spanwise edge of the grid was a centerline symmetry plane and the other was the outer wall of the wind tunnel. Vertically, the boundaries were the top and bottom walls of the wind tunnel. The overall size of the region modeled was 11.0 in (27.9 cm) in the streamwise direction and 4.50 in (11.4 cm) in the normal and spanwise direction.

The injectors that make up the wall injector array could be treated in a number of different manners. Since the thermodynamic properties of the injectant helium were known from experiment, those properties could be applied as a fixed boundary condition on the surface of the main grid itself, eliminating the need for including the injector ports in the calculation. This method would have been physically and numerically proper if the component of the injectant velocity normal to the wall had been sonic or supersonic, for then information from the freestream would not have been expected to propagate downward to affect the conditions at the injectant inflow boundary. While the injectant helium in this case was sonic, the low

angles of the injectors resulted in a vertical component of velocity that was quite subsonic. Under those conditions, the application of fixed inflow boundary conditions would have been physically unrealistic and numerically unstable. The best alternative was to include a short distance inside each injector port as part of the computational domain. By doing so, grids could be created for the ports such that the inflow planes were normal to the injectant helium. Then the component of velocity normal to the inflow planes would be sonic, and full specification of helium properties on the boundaries would be both physically and numerically reasonable. Figure 4.1 illustrates the outline of the full computational domain, including the injector ports. Figure 4.2 shows the details of the surface grid in the injector area.

Grid points were concentrated near the bottom wall of the primary computational space to allow the resolution of the boundary layer, but the side wall and top wall were modeled as inviscid walls and grid points were not concentrated there. This decision was based upon the expectation that the injectant flow would not spread sufficiently to interact with either of the neglected boundary layers, and experimental data have supported the expectation.

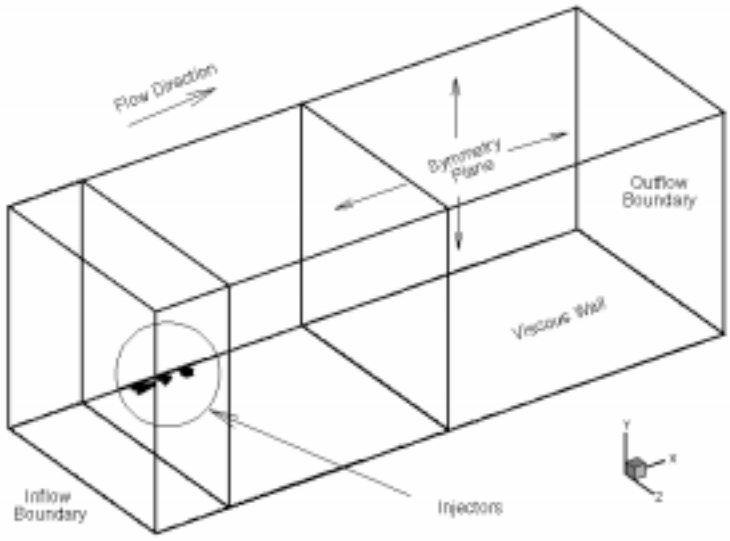

Figure 4.1. Outline of Grid for Nine-Hole Array Calculations, Showing Location of Injector Ports.

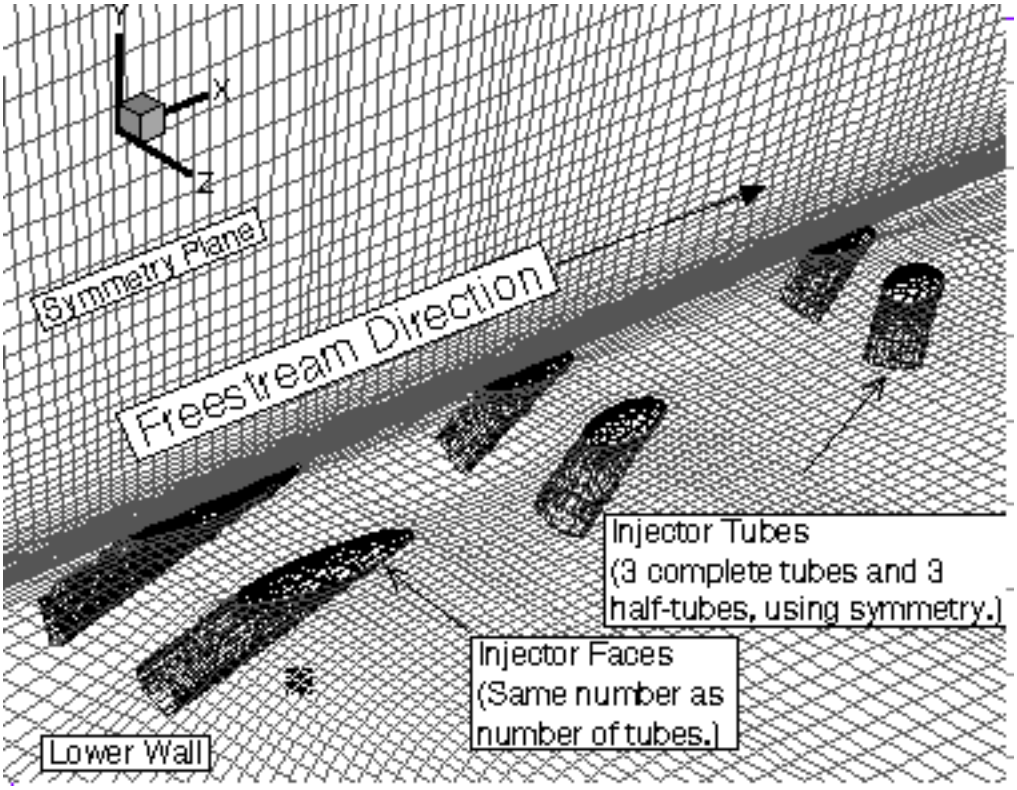

Figure 4.2. Detail of Injector Region for Nine-Hole Injector Array.

Any attempt to include enough grid points in the domain to resolve the wall boundary layers within the injector ports themselves would have tremendously increased the number of grid points required for the calculation, not only in the injector ports, but also in the primary computational domain. For this reason the boundary layers within the injector ports were neglected, and the port-grids created with inviscid flow in mind. Nonetheless, continuity in grid spacing (i.e., the need to prevent sudden changes in cell size) necessitated the injector-port grids be closely spaced where they came in contact with the more viscous grid spacing of the primary computational region. Figure 4.2 provides a closer look at the outlines of the port-grid portions of the computational domain shown in Figure 4.1, including the surface meshes that the injector ports share with the primary computational region. Notice the rather

large grid spacing the inviscid simplification allows. Figure 4.2 also shows the surface mesh of primary computational region in the viscinity of the injector ports. Notice how the elliptical ports blend into the quadrilateral surface mesh.

Nine independent computational zones were used in the solution of this flow. The first zone was upstream of the wall injector array and existed solely to develop the boundary layer. The second zone contained the injector array itself and the primary mixing region. The third through eighth zones were each one of the injector ports. The last zone extended the calculation downstream of the primary mixing region to the measurement station. Table 4.1 illustrates the streamwise placement and sizes of the primary-flow zones. Injector-port zones are described in Table 4.2. In both tables the streamwise coordinate is measured from an origin in the center of the first (most upstream) injector, and positive distances are downstream.

Mesh sequencing was employed in all zones not only as a measure of grid convergence but also as a means of convergence acceleration. Three levels of mesh sequencing were used, with the

Table 4.1. Streamwise Placement of Injector Array Primary Zones.

Zone	Streamwise Coordinate			
	Upstrm. face (in)	Upstrm. face (cm)	Dnstrm. face (in)	Dnstrm. face (cm)
1	-2.88	-7.31	-1.30	-3.31
2	-1.30	-3.31	3.42	8.69
9	3.42	8.69	9.33	23.69

Table 4.2. Position and Orientation of Injector-Port Zones of Injector Array

Zone	Injector Row	Injector Placement	Transverse Angle in degrees	Yaw Angle in degrees*
3	Upstream	Centerline	15	0
4	Upstream	Outer	15	0
5	Middle	Centerline	30	0
6	Middle	Outer	30	-15
7	Downstream	Centerline	45	0
8	Downstream	Outer	45	-30

* Yaw angles are measured positive away from the centerline.

fine mesh corresponding to a y+ of approximately 0.9 or lower at the center of the first cell off the wall. Sequencing in the “i” and “k” three logical directions was accomplished identically: the medium mesh was created by deleting every other point from the fine, and the coarse mesh was created by deleting every other point from the medium. The $K-\varepsilon$ turbulence model, however, is sensitive to cell placement in boundary layers, and mesh sequencing in the "j" logical direction (normal to the lower surface of the wind tunnel) could not be accomplished in the primary computational zones. As a result the nine-hole injector array calculations were mesh sequenced only in the "i" and "k" logical directions in these zones. The number and placement of cells in the "j" direction was identical for all three meshes. The injector-port zones, because they did not contain boundary layers, were expected to be less sensitive to cell placement, but were sequenced to match the primary zone with which they interfaced. In these zones the "i" logical direction was normal to the injector faces, not the "j". See Table 4.3, in which Mesh 1 corresponds to the finest grid.

Table 4.3. Mesh Sequencing for Injector Array.

Zone	Mesh	# of cells in each direction:		
		i	j	k
1	3	4	80	19
1	2	8	80	38
1	1	16	80	76
2	3	28	80	19
2	2	56	80	38
2	1	112	80	76
3	3	20	3	1
3	2	20	6	2
3	1	20	12	4
4	3	20	3	1
4	2	20	6	2
4	1	20	12	4
5	3	20	2	1
5	2	20	4	2
5	1	20	8	4
6	3	20	2	1
6	2	20	4	2
6	1	20	8	4
7	3	20	1	1
7	2	20	2	2
7	1	20	4	4
8	3	20	1	1
8	2	20	2	2
8	1	20	4	4
9	3	6	80	19
9	2	12	80	38
9	1	24	80	76

Ramp Injector:

For the purposes of the calculation the ramp injector was modeled as one half a ramp with symmetry to account for the other half. Because the injectant fluid was supersonic and very nearly normal to the inflow plane, the injector inflow boundary conditions could be specified on the surface of the primary computational domain and the injector ports themselves need not be modeled. The computational domain for the ramp injector began 10.0 in (25.4 cm) upstream of the face of the ramp itself, once again to allow the development of a boundary layer from a computational leading edge, and ended at the last measurement station, a distance of 20.0 ramp heights (h) downstream. The first computational zone was a constant-area duct ahead of the ramp. The second and fourth zones were narrow trapezoidal areas above the ramp and the associated expansion (with zone 2 upstream of zone 4), and the third and fifth corresponded to the wedge-shaped expanding flow area beside the ramp. (Zone 3 occupied the narrower, upstream portion of the wedge, and zone 5 was a wider, trapezoidal area at the base of the wedge.) The sixth zone was the primary mixing region downstream of the face of the ramp. It was 2.5h long, a distance expected to be sufficient for the redevelopment of parabolic flow following the perturbation created by the ramp. (Calculations later confirmed this distance as sufficient.) The seventh through tenth zones extended the computational domain downstream to the last measurement station. See Figures 4.3 - 4.5.

Once again mesh sequencing was performed to check for grid convergence and to reduce calculation time. On the fine mesh, the y^+ at the center of the first cell off the "j0" wall was generally less than 0.7 and frequently less than 0.5. Similar spacings were employed on the vertical faces of the ramp (the “k0” wall in Zone 5 and the “i0” wall in Zone 6). The finer spacing allowed mesh sequencing in all three directions for this case, even normal to viscous walls. See Table 4.4 for details of mesh sequencing.

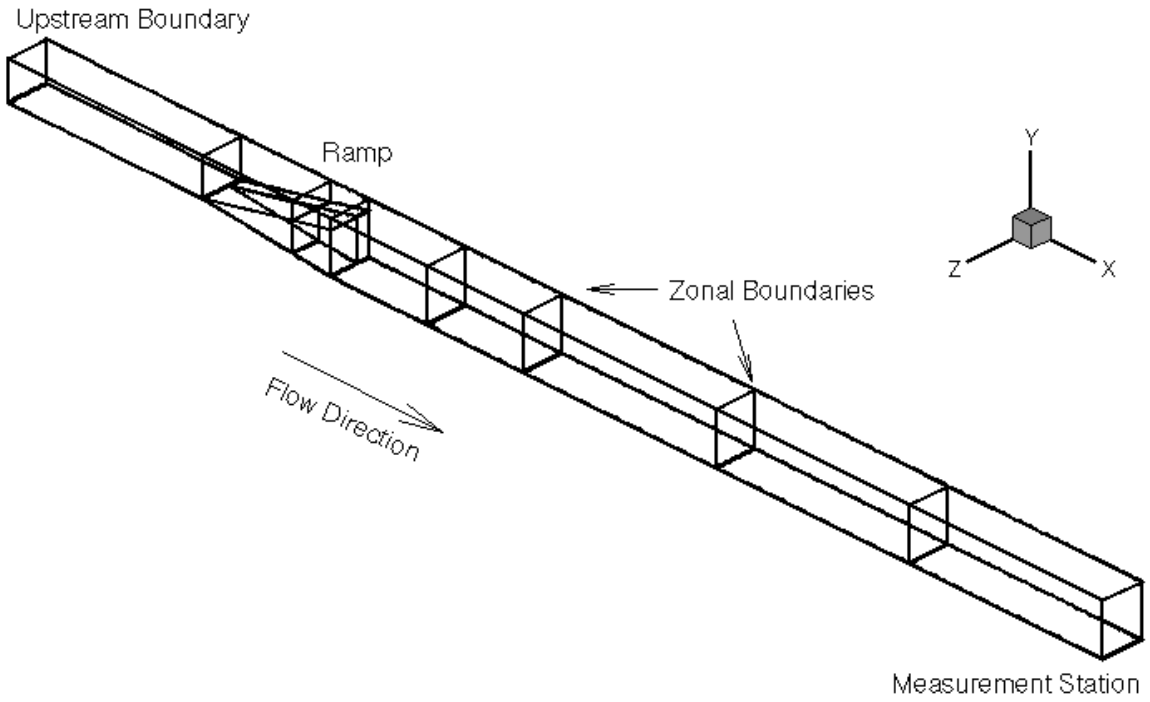

Figure 4.3. Outline of Grid for Four-Hole Ramp Calculations, Showing Zonal Boundaries and Location of Ramp.

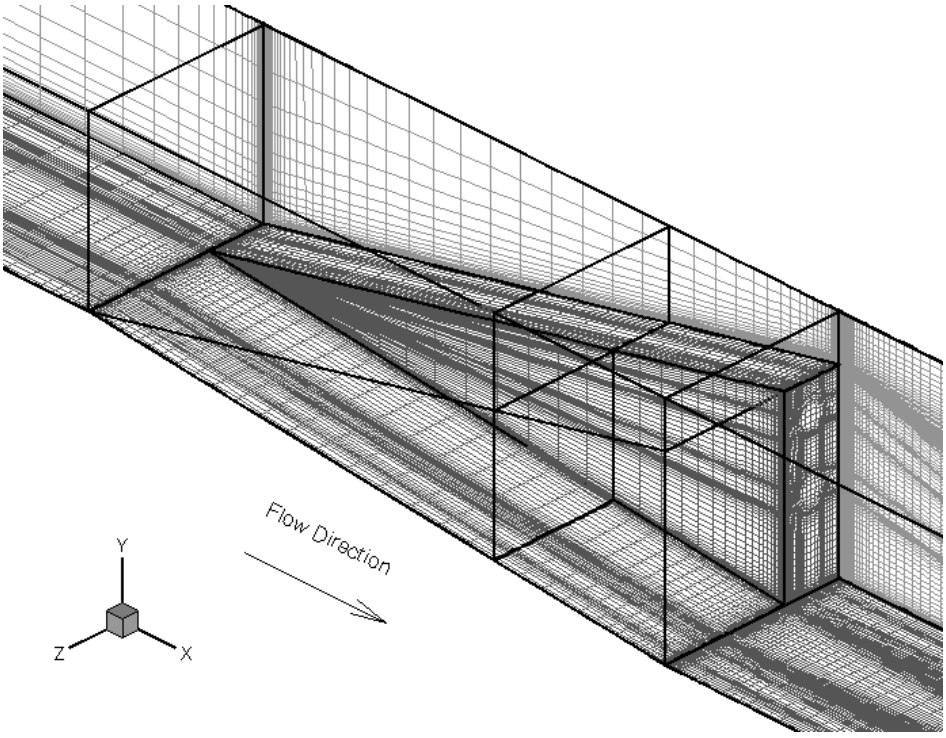

Figure 4.4. Detail of Ramp Region for Four-Hole Ramp Array, Showing Centerline, Wall, and Ramp Grids and Zonal Boundaries.

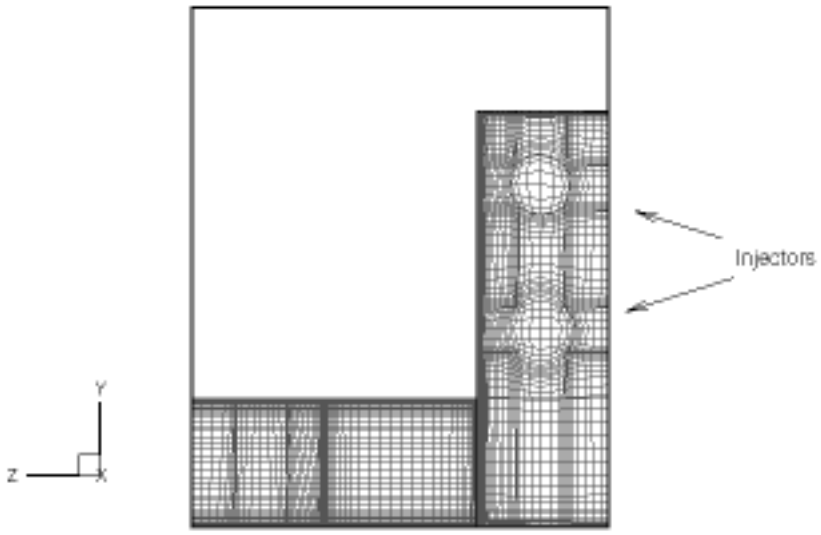

Figure 4.5. Detail of Ramp Region for Four-Hole Ramp Array, Showing Lower-Wall and Ramp Grids. Flow Comes Out of the Page.

Table 4.4 Mesh Sequencing for Ramp Injector

Zone	Mesh	# i-cells	# j-cells	# k-cells
1	3	10	8	36
1	2	20	16	72
1	1	40	32	144
2	3	10	8	36
2	2	20	16	72
2	1	40	32	144
3	3	10	20	26
3	2	20	40	52
3	1	40	80	104
4	3	8	8	36
4	2	16	16	72
4	1	32	32	144
5	3	3	20	26
5	2	16	40	52
5	1	32	80	104
6	3	29	28	36
6	2	58	56	72
6	1	116	112	144
7	3	8	28	36
7	2	16	56	72
7	1	32	112	144
8	3	13	28	36
8	2	26	56	72
8	1	52	112	144
9	3	13	28	36
9	2	26	56	72
9	1	52	112	144
10	3	13	28	36
10	3	26	56	72
10	1	52	112	144

Test Conditions

Wall Injector Array

As mentioned above, the computational domain for the wall injector array consisted of nine zones. The first zone was space-marched in the positive "i" direction (downstream), and the rest were globally iterated with two-factor approximate factorization and relaxation in the "i" direction. The "j0" boundary condition for all primary zones (zones one, two, and nine) was specified as a no-slip, constant-temperature wall, though the injector cells in the second zone were zonal interfaces as discussed below. Because each wind tunnel test (run) was very short for this case (on the order of ten seconds), the wind-tunnel walls did not have time to reach equilibrium with the static temperature of the flow and the wall temperature at the beginning of the run was chosen as the computational boundary condition. In this case, the wall temperature at the beginning of the run was equal to the total temperature of the flow. The "jdim" and "kdim" boundary conditions of the primary zones were tangency conditions (inviscid walls) as discussed above, and the "k0" boundary was a plane of symmetry. For zone one the inflow ("i0") boundary was fixed at freestream, and for zone nine the outflow ("idim") boundary was a second-order extrapolation from the interior.

Also as mentioned above, the injector-port zones, zones three through eight, were created for and run with fixed inflow conditions as the "i0" boundaries. The "idim" boundaries were zonal boundaries that connected to zone two. The other four boundaries were simple tangency (inviscid wall) boundary conditions. (Centerplane injectors used symmetry boundary conditions on their “k0” faces.)

Freestream conditions were fixed to experimental values by specifying a streamwise Mach number of 3.0, other Mach number components of 0.0, freestream temperature of -268.2 °F (106.4 K), and air static density of .03645 lb./ft^3 (0.58389 kg/m^3). The cells on the "i0" boundaries of the injector-port zones were provided with fixed values of total temperature, pressure, and composition, and

allowed to accelerate to sonic velocities within the injector tubes themselves. The actual values of total temperature and total pressure were perturbed slightly from their baseline values of 290K (62° F) and 7.0 atm (with changes no larger than a few percent of the total) to match the computed mass flow rate to the measured value. (The changes were necessary to account for the differences in cross-sectional area between the perfectly circular injector tubes in the experiment and the nearly circular, polygonal injector tubes in the computation.)

GASP v.2.2 provides the option of specifying separately the terms to be included in the viscous flux calculations, and this feature was used to try to ensure that all relevant terms were included in each zone without unnecessary calculations. In zone one, a simple, flat-plate marching zone, the only viscous flux terms included were thin-layer terms in the "j" direction, normal to the surface of the plate. Zones two and nine were also nominally flat-plate regions, but the flow in those zones was complicated by multi-species injection and mixing. For this reason thin-layer terms and cross derivatives were included in the "j" and "k" logical directions. Streamwise viscous flux ("i"-direction) terms were neglected, as they were expected to be much smaller than the corresponding inviscid fluxes. Viscous fluxes in the injector zones also included “j” and “k” thin-layer and cross-derivative terms, though poor grid resolution and the use of inviscid walls prohibited any significant viscous influence on the injector flow.

Over the course of the calculations, CFL's were often changed to accommodate evolving stability of the calculation. (For example, the CFL required for stability at the beginning of a calculation was usually smaller than that possible near the end.) Documentation of exact values used would result in a complicated pattern of limited usefulness. The issue of CFL's is discussed more thoroughly in the next chapter.

Ramp Injector

The computational domain for the ramp injector was divided into ten zones, as discussed above. The first zone was space-marched in the "i" (streamwise) direction. The second and third zones, together with the last planes of the first, were globally iterated with two-factor approximate factorization and relaxation in the "i" direction, because zones two and three shared the same streamwise position in the flow and physically information was free to pass between them. The last planes of zone one were included to allow for the possible upstream propagation of information about the ramp and trough, which could travel through the subsonic portions of the boundary layer and result in an oblique shock just ahead of the ramp. Zones four and five were globally iterated (once again with two-factor approximate factorization and relaxation in the "i" direction, as were all global zones in this calculation) with zone six, so that backflow and other physically possible forms of upstream propagation would be free to occur as necessary. Zones seven through ten were also globally iterated.

Flow through the injectors in zone six was specified by providing species densities, velocity components, static pressure, turbulence kinetic energy, and dissipation rate on each cell. The molecular weight of the injectant mixture was 14.86, and the helium mass fraction was 0.1783. This corresponds to a helium mole fracture of .5621. The static density of injectant air was fixed at .03076 lb./ft^3 (0.4932 kg/m^3), the helium static density was .005484 lb./ft^3 (.08793 kg/m^3), the magnitude of the velocity was 2387 ft/sec (727.5 m/s), and the static pressure of the injectant was 4.178 psi (28.81 kPa). The turbulence kinetic energy was specified at 14240 ft^2/s^2 (1323 m^2/s^2) using the expression

$$K \approx (t.i.)^2 V^2 \tag{4.31}$$

where t.i. is the turbulence intensity, set to 0.05, and V is the magnitude of the injectant velocity. The dissipation rate was

110.4e+6 ft^2/s^3 (10.26e+6 m^2/s^3), an estimate determined using the expression

$$\varepsilon \approx K^{1.5} / l \qquad (4.32)$$

with l being a length scale taken to be approximately equal to the diameter of one injector.

Freestream properties for the ramp injector (which were specified as inflow conditions for the most upstream zone) were specified to match those of the corresponding experiment. The streamwise ("i") Mach number component was 4.5 and the other components were zero. The freestream static temperature was -245.7 °F (118.9 K), and the freestream static air density was 0.01350 lb./ft^3 (0.2163 kg/m^3). Walls for this problem were considered both viscous and adiabatic, because longer tests allowed the wind-tunnel walls to reach equilibrium with the flow.

Once again viscous flux terms were chosen zone by zone in an attempt to include all relevant terms without unnecessarily increasing computational cost. All zones except zone six included thin-layer plus cross derivative terms in the direction normal to the lower wall (the "j" direction) and in the spanwise ("k") direction. Zone six, the primary mixing zone, was treated as thin-layer only in all directions, due to stability constraints. In other zones the "i" viscous flux terms and their cross-derivatives were neglected.

As with the wall injector array problem local time-stepping was employed by specifying a CFL number. The specified CFL number was applied to calculations at every cell in the given zone, but could be (and was) altered periodically to suit immediate stability characteristics of the turbulence model, flow, and calculation in general. The pattern of CFL usage in the ramp injector problem was somewhat more straight-forward than that of the wall injector array problem, but nonetheless the issue of CFL's will be held for discussion in the following chapter.

Flat-Plate Test Case

The hope for every new turbulence model is that it will improve the quality of the solution generated for complex or troublesome flows for which previous turbulence models have been inadequate. Equally important, however, is that a new turbulence model return reasonable solutions to simple cases for which other turbulence models do work relatively well. This is particularly important for RNG, which is simply a higher-order correction to the classical $K-\varepsilon$ turbulence model. For "simple" flows the higher-order terms should be negligible and the model should reduce to classical $K-\varepsilon$. Therefore, a fundamental and important test of RNG $K-\varepsilon$ is to verify the quality of its behavior in a simple flow and to ensure that it does in fact produce solutions of comparable quality to standard $K-\varepsilon$ model.

The most fundamental test case investigated was the flow of air over a two-dimensional adiabatic flat plate. The Mach number was 2.0 and a freestream static temperature was 125 K. The static density was .300 kg/m^3. A five percent turbulence intensity was assumed for the freestream. The computational domain was 0.04 m high (normal to the wall), and 1.00 m long. Two calculations were performed using standard Chien $K-\varepsilon$, in order to determine the mesh requirements for a grid-converged solution. The finer, better-resolved computational mesh had 201 cells in the normal direction and 201 in the streamwise direction, and the coaser mesh had half those numbers. Two solutions were computed on this finest mesh, a standard Chien $K-\varepsilon$ solution and a “fast” RNG solution. “Mixing” RNG was not tested on this problem.