NASA Contractor Report 198440

NASA-CR-198440
19960010178

TADS–A CFD-Based Turbomachinery and Analysis Design System With GUI

Volume I—Method and Results

D.A. Topp, R.A. Myers, and R.A. Delaney
Allison Engine Company
Indianapolis, Indiana

December 1995

Prepared for
Lewis Research Center
Under Contract NAS3–25950

National Aeronautics and
Space Administration

Contents

List of Figures

List of Tables

Chapter 1

Summary

The primary objective of this study was the development of a CFD (Computational Fluid Dynamics) based turbomachinery airfoil analysis and design system, controlled by a GUI (Graphical User Interface). The computer codes resulting from this effort are referred to as *TADS* (Turbomachinery Analysis and Design System). This document is the Final Report describing the theoretical basis and analytical results from the *TADS* system, developed under Task 18 of NASA Contract NAS3-25950, *ADPAC* System Coupling to Blade Analysis & Design System GUI.

TADS couples a throughflow solver (*ADPAC*) with a quasi-3D blade-to-blade solver (*RVCQ3D*) in an interactive package. Throughflow analysis capability was developed in *ADPAC* through the addition of blade force and blockage terms to the governing equations. A GUI was developed to simplify user input and automate the many tasks required to perform turbomachinery analysis and design. The coupling of the various programs was done in such a way that alternative solvers or grid generators could be easily incorporated into the *TADS* framework. Results of aerodynamic calculations using the *TADS* system are presented for a highly loaded fan, a compressor stator, a low speed turbine blade and a transonic turbine vane.

Chapter 2

Introduction

The aerodynamic design of turbomachinery airfoils is one avenue to improved engine performance, efficiency, and weight. Flow over turbomachinery airfoils is 3-dimensional (3-D) and viscous, with complicated flow features arising from shock waves, tip clearances, seal cavities, and cooling passages. Airfoil design also involves trade-offs between aerodynamic performance and requirements from stress, heat transfer, and other mechanical considerations.

Traditional airfoil design approximates the 3-D flow by the quasi-3D flow in two perpendicular surfaces. One surface (S1) is in the blade-to-blade plane, and models the flow between the airfoils along a streamline in the meridional plane. The other surface (S2) is in the meridional plane, and models the radial distribution of flow. This is often called the throughflow analysis. The shape of the S2 surface is determined from the S1 surface, and the shape of the S1 surface is determined from the S2 surface. Convergence of the scheme can be achieved by iteration. Frequently, only one iteration is performed: the shape of the S2 surface is set from the airfoil shape and deviation and loss correlations, and the blade-to-blade conditions are determined from the S2 solution. This approach, introduced by Wu, Ref. [21], forms the basis of most turbomachinery airfoil design systems in use today.

In the last few years, advances in CFD have enabled the use of 3-D codes to model the flow in turbomachinery blade rows. While modern CFD codes are capable of modeling the important features of these complicated flows, they are relatively slow and use large amounts of computer memory. Advances in computer technology and in solution algorithms are reducing the penalties associated with 3-D modeling, but routine design is still not

practical with these tools.

The advantage of 3-D modeling is obvious: more of the flow features are calculated, instead of being prescribed by correlations. The advantage of the traditional approach is that the airfoil can be designed as a stack of 2-D sections. There is a large experience base in the design of 2-D sections, and the associated design parameters are well understood. While 3-D analysis is common, 3-D design is not. Currently, 3-D design is accomplished by adjusting 2-D parameters in response to 3-D analysis.

Recently, there has been considerable interest in updating the traditional design methods with modern CFD tools. There is a large gap in capability between the traditional design system and full 3-D viscous flow analysis. Much of this gap can be closed by incorporating the latest CFD techniques into the the traditional approach. For instance, the deviation angle in the blade-to-blade solution need not be specified if a Navier-Stokes solver is used to compute the detailed flow solution for the airfoil section. Similarly, the effects of upstream total temperature and pressure profiles can be captured by a CFD based throughflow analysis. The effects of neighboring blade rows can also be economically modeled by an axisymmetric representation of the flow. The work of Spurr, Ref. [18], and Jennions and Stow, Ref. [9] in the 1980's laid the groundwork for a number of recent publications. Yao and Hirsch, Ref. [23], developed a throughflow analysis based on CFD techniques. Damle, Dang, and Reddy, Ref. [5], developed a throughflow analysis with capability for both analysis and design. Sayari and Bolcs, Ref. [15]. investigated the effects of different averaging procedures and blockage models in the throughflow analysis.

These papers on throughflow analysis differ in focus, but follow a common strategy: the presence of the airfoil in the passage is modeled by body force terms and a blockage term. As the flow proceeds through the bladed region, the body forces model the change in swirl velocity imparted by the airfoil. The blockage term models the acceleration and deceleration of the flow, caused by the thickness of the airfoil in the passage, and by deviation of the flow from the airfoil surface. A new model for body forces and blockage was developed in the *ADPAC* solver for this purpose. *ADPAC* is a 3-D Euler/Navier-Stokes analysis which is capable of performing axisymmetric calculations, Ref. [8].

Quasi 3-D blade-to-blade solvers have special features for solving flow between airfoils along a meridional streamline. These features include ro-

tational terms, radius terms, and stream tube thickness terms. The radius and stream tube thickness terms differentiate a 2-D solver from a quasi 3-D solver. These terms allow the blade-to-blade flow to feel the effects of the changes in the meridional flow path. The radius terms account for the change in blade pitch associated with changes in radius, and the stream tube height terms account for the change in the distance between neighboring streamlines. *RVCQ3D*, Ref. [2] and Ref. [3], is a good example of a quasi-3D analysis.

The objective of the present work is to produce a turbomachinery airfoil design and analysis package built on the traditional approach, but using modern analytical techniques. This new Turbomachinery Analysis and Design System (*TADS*) is controlled by a Graphical User Interface (GUI), which simplifies user input and automates the many required tasks. *TADS* couples a throughflow solver (*ADPAC*) with a quasi-3D blade-to-blade solver (*RVCQ3D*) in an interactive package. The coupling is done in such a way that alternative solvers or grid generators can be easily incorporated into the *TADS* framework.

Chapter 3

Analysis Coupling

A coupled throughflow and blade-to-blade analysis requires many steps, repeated iteratively. Figure 3.1 shows the work flow of a typical analysis. A converged analysis is achieved when the meridional streamlines are settled in the throughflow analysis and when the mean stream surface is settled in the blade-to-blade analysis. Each analysis provides the solution surface for the other, and iteration is required to determine the final shapes. In practice, only one iteration is required to achieve an acceptable solution in many cases.

3.1 Solution Procedure

Since the coupled analysis is an iterative procedure, there is more than one possible path. There are two possibilities: start with the blade-to-blade analysis, or start with the throughflow analysis. Which one to choose is a function of the airfoil shape design program and of user preference. In either case, there is some critical information which must be fabricated as an initial guess. The throughflow analysis requires a mean stream surface which is found from the blade-to-blade solutions, and the blade-to-blade solutions are performed along streamlines provided by the throughflow calculation. *TADS* begins with the throughflow analysis, using the mean camber line and, optionally, Carter's deviation angle rule to set the mean stream surface.

The first step in the analysis is to acquire a description of the airfoil and of the flow path. Certain aerodynamic data are also required, such as the upstream total pressure and temperature, upstream flow angle, and

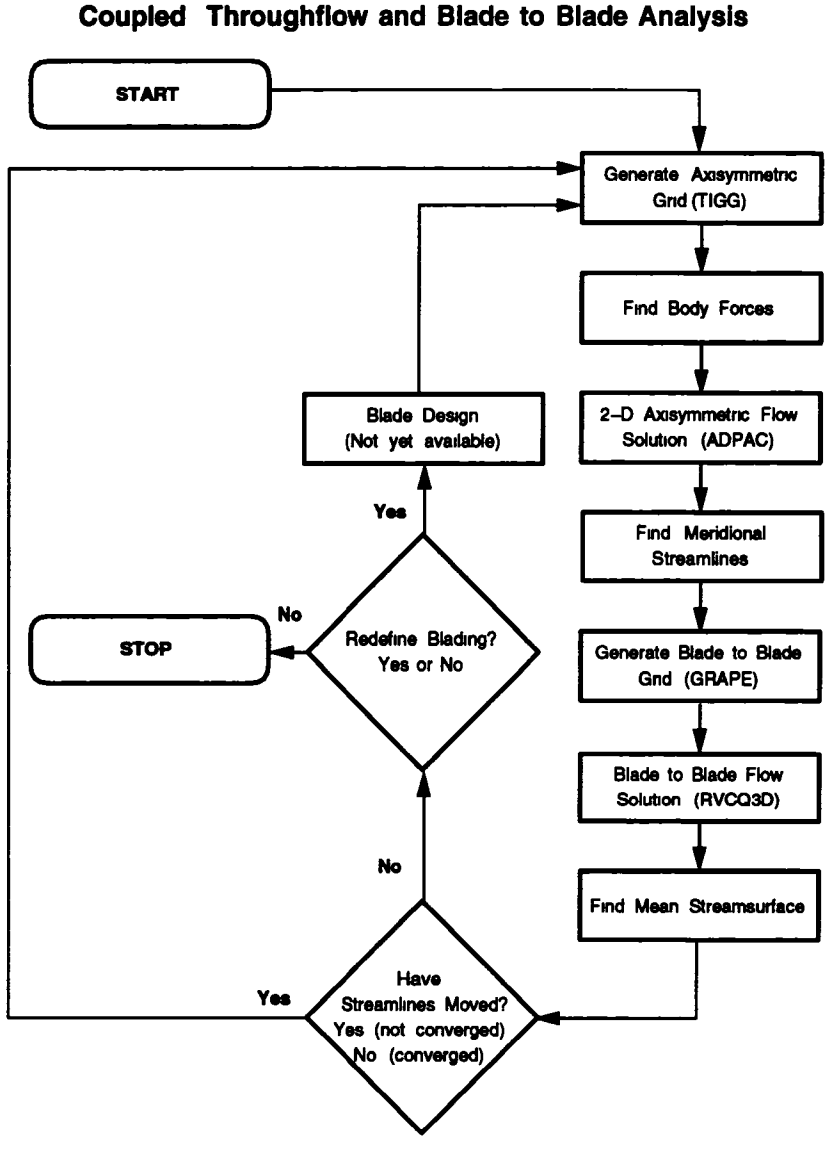


Figure 3.1: The coupled throughflow and blade-to-blade analysis is an iterative, multi-step process.

downstream static pressure. Typically, airfoil design programs specify the aerodynamic inflow and outflow quantities at the leading and trailing edges, respectively. *TADS* follows this convention and extrapolates the required data to the upstream and downstream grid boundaries when required. Actually, only the throughflow analysis utilizes this aerodynamic data: the blade-to-blade analysis takes its aerodynamic input by interpolation from the throughflow solution.

The second step is to generate a grid for the throughflow calculation. This requires the flow path and the meridional projection of the airfoil leading and trailing edges. The axisymmetric grid generator used in *TADS* is *TIGGC3D*, which is related to *TIGGERC*, Ref. [12]. The output is a planar axisymmetric grid with grid lines coinciding with the leading and trailing edges.

The third step is to run the throughflow analysis, *ADPAC*. *ADPAC* requires as input the grid, an input file containing controlling parameters, a boundary condition file, and a body force file. The grid must be modified to show the shape of the mean stream surface in the bladed region. *ADPAC* forces the flow to be tangent to the grid in the bladed region, and computes the body forces required for flow tangency. A separate program is used to apply the mean stream surface shape to the grid from *TIGGC3D*. Another program is used to generate the boundary condition file, and the input file is constructed from the GUI. The user sees only the input panel on the GUI; the rest is transparent to the user. After the analysis is run, some checking is appropriate for convergence and for solution quality.

The fourth step is to find the meridional streamlines from the throughflow solution. Only the number and distribution of the streamlines are required as input. The streamlines are found by accumulating flow from hub to tip along radial grid lines. The flows are then normalized, and contours are traced from inlet to exit at values of constant mass flow.

The fifth step is to slice the airfoil along the meridional streamlines. This step requires no new input. The output of this step are the airfoil sections along the meridional streamlines which are to be used in the blade-to-blade analysis.

The sixth step is to generate blade-to-blade grids for each airfoil section. The input is controlled by the GUI, and includes parameters for the grid size, upstream and downstream extents, number of blades, etc.

The seventh step is to run the blade-to-blade solver for each airfoil section. This step is typically the most time consuming part of the analysis. The

input is controlled by the GUI, and includes parameters for the number of iterations, the size of time step, turbulence model choices, etc. These solutions should also be checked for convergence and quality. One good check is to sum the mass flows from the blade-to-blade solutions, and compare with the output of the throughflow analysis.

The eighth and final step is to compute the mean streamline between the airfoils for each airfoil section. This involves stacking the quasi 3-D solutions into an equivalent 3-D file, finding streamlines on the blade-to-blade surfaces, and interpolating the shape onto the throughflow grid. This step can be omitted if no iteration is to be performed.

These eight steps can be repeated, iteratively, until the mean stream surface used in the throughflow analysis and the radial streamlines used in the blade-to-blade analysis are settled.

3.2 Programming Philosophy and Standards

The *TADS* system is an amalgamation of many different programs under a single GUI. One of the objectives in the development of *TADS* was to enable new modules to be added to perform any of the tasks without major coding effort. That is, additional choices for grid generators or flow solvers could be added in a modular fashion. The biggest obstacle to modularity is that each program has its own set of standards. Each has its own input and output format, its own coordinate system, its own non-dimensionalization, etc.

One approach is to make each program a subroutine called by the GUI. This way, all data could be passed internally and the system would be tightly coupled. There are many disadvantages to this approach, however. First, each code would require significant modification to be integrated into the GUI. These modifications would need to be remade each time a new release of the code was received. Second, if each code is a subroutine of the GUI, it is difficult to send calculations to a remote machine to take advantage of faster platforms. Finally, each code would no longer work as a stand-alone product. The user would be forced to use the GUI to be able to access the code. Many of these codes can be used for purposes outside of *TADS*, and it is advantageous to retain access to these unused features.

A second approach is to leave each code as a stand-alone module, and either modify the I/O of the code to conform to some standard, or write

conversion modules into the input generators and post-processors for each code. Since the grid and solution files are the only link between one program and another, it is simpler to modify the I/O than to write special conversion routines. *TADS* follows this approach. The disadvantage to the *TADS* approach is that there are many files created during an analysis, and the directory can become cluttered. Although the clutter is unfortunate, these files provide a built-in restart capability for the analysis.

3.2.1 File Naming Convention

The files created or used by *TADS* use the casename.extension file name convention adopted from *ADPAC*. The user specifies a case name for the problem, and each file needed by *TADS* assigns a unique extension to it. This way, multiple airfoils could be run in the same directory. There is also much less confusion about which files were created by *TADS*. Some programs, notably the grid generators and quasi 3-D solvers expect files with specific names for input and output. These files do not follow the convention adopted for *TADS*. This is not a serious problem unless multiple runs of the same program must be made in the same directory. Multiple runs would require multiple files with the same name, resulting in overwritten data or confusion about the contents of files. While it would be possible to write scripts to rename or symbolically link files to the expected names, it is clearer and simpler to create subdirectories to contain these files. *TADS* creates a subdirectory for each blade-to-blade section to be analyzed. Within the subdirectory, some files do not conform to the naming convention, but confusion is avoided because the subdirectories themselves are named descriptively.

3.2.2 Data Standards

All files used by *TADS* are either ASCII text, or binary files written with the SDB library. SDB is a library of I/O routines which create platform independent binary data. On each platform, an SDB library is available to perform the necessary conversions. Using SDB, any platform can read binary data created by any other platform. Supported platforms include Cray, Silicon Graphics, IBM RS/6000, Sun, etc. The binary data structure of SDB is equivalent to reading and writing binary data in C on a Silicon Graphics workstation. SDB is documented in Ref.[20]. All *TADS* files are platform in-

dependent, so any program task can be performed on any supported machine without loss of generality.

Most of the binary files used by *TADS* are geometry or flow data files. All geometry or flow data files are written in *PLOT3D* format using SDB. Specifically, all files are 3-D, whole, multiple grid files, in accordance with the definitions in Ref. [19], pp 162-165.

3.2.3 Coordinate Systems

While *PLOT3D* files are Cartesian, many of the modules within *TADS* use cylindrical polar coordinates. Most *TADS* modules read the Cartesian coordinates and convert immediately to cylindrical polar for the internal calculations. All output files are converted back to Cartesian for output.

In the conversion between cylindrical polar and Cartesian coordinates, there are two common orientations: place θ=0 along the Y axis, or place θ=0 along the Z axis. The standard orientation in *TADS* places the R axis in cylindrical coordinates along the Z axis in Cartesian coordinates when θ=0. This is, in effect, a right handed system in which (X,θ,R) corresponds to (X,Y,Z). Some *TADS* modules, notably *TIGGC3D* and *ADPAC*, operate with a left handed coordinate system. Since only two dimensions are used, it is relatively unimportant except that the Cartesian orientation of a *TIGGC3D* grid is in violation of the *TADS* standard. The *TIGGC3D* mesh is modified by the body force calculator, which then sets the θ distribution according to the *TADS* standard.

The standard coordinate system and orientation make it simple to graphically compare the input and output of the various codes. For example, the user can examine the difference between the axisymmetric average stream surface computed from the blade-to-blade solver and the distribution set according to the mean camber line. It is also possible to verify that the mean camber line lies properly in the original airfoil description. Most of the modules would perform equally well with input files in another orientation, but verification would be more difficult. The coordinate system standard was adopted so that the geometric information used in each step of the analysis could be compared graphically without a coordinate transformation.

3.2.4 Shared Routines and Data

There are many routines which are shared between *TADS* modules. There are also many modules which need the same data structures (common blocks, etc) as other *TADS* modules. These routines and include files are saved in a separate subdirectory which is accessible by all *TADS* modules. This was done to eliminate duplicate (and possibly conflicting) copies of subroutines and include files. The common routines are bound into a library which is linked into each of the *TADS* modules. The include files are made available to the *TADS* modules through symbolic links. Each module has a makefile, to build the executable from the source code. Each makefile has a dependencies section which causes routines to be recompiled if an include file has been updated. The dependencies section insures that all object code will be up to date before an executable is made. These practices dramatically reduce the possibility of data errors in the codes. Each module uses the same data structures, and only one copy of each routine or include file exists.

3.3 Input Requirements

The *TADS* system requires four things as input: a case name, a Cartesian description of the airfoil, a description of the meridional flow path, and aerodynamic data. The airfoil is input as a 3-D surface in two parameters. One parameter wraps clockwise around the airfoil to form a closed surface, and the other runs with the span of the airfoil. The meridional flow path is defined by two lines in the (X, R) plane. The aerodynamic data contains tables of information at the leading and trailing edges. These tables consist of radial profiles of total temperature and pressure, static pressure, and Mach number components. This file also contains the ratio of specific heats, the number of blades, and the tangency points of the airfoil. The tangency points are those points in the airfoil description which denote where the leading and trailing edges join the pressure and suction surfaces. The User's Manual provides details on the contents and organization of the input files. All other information needed by *TADS* has either a default value which can be reset in an input panel, or is generated by another part of the analysis.

Chapter 4

Development of Program Modules

The *TADS* system is comprised of many independent modules which are linked together by the GUI. This chapter details the development of each module, in the order they are normally encountered in an airfoil analysis. Many of these modules were developed specifically for the *TADS* system, while others were provided. The user is referred to existing documentation for the provided programs for additional details.

4.1 *INTIGG*

INTIGG is an input generator for *TIGGC3D*. *INTIGG* takes its input from the casename.tdsaxi file and from the airfoil description and flow path files. The casename.tdsaxi file is created by the GUI, and contains the user choices entered in the *TIGGC3D* input panel. Included in this information are the grid size, indices of the leading and trailing edge, grid extents as a fraction of the axial chord, and whether or not to apply Carter's deviation angle rule. The Carter's rule trigger is ignored by *INTIGG* but is used by another program module.

INTIGG requires an axisymmetric representation of the airfoil, which consists of the shape of the leading and trailing edges in the meridional plane. The meridional projection of the leading and trailing edges is computed simply by locating the minimum and maximum axial extents of the

airfoil description on each defining slice. If the machine is a centrifugal or radial device, then the appropriate radius is found instead.

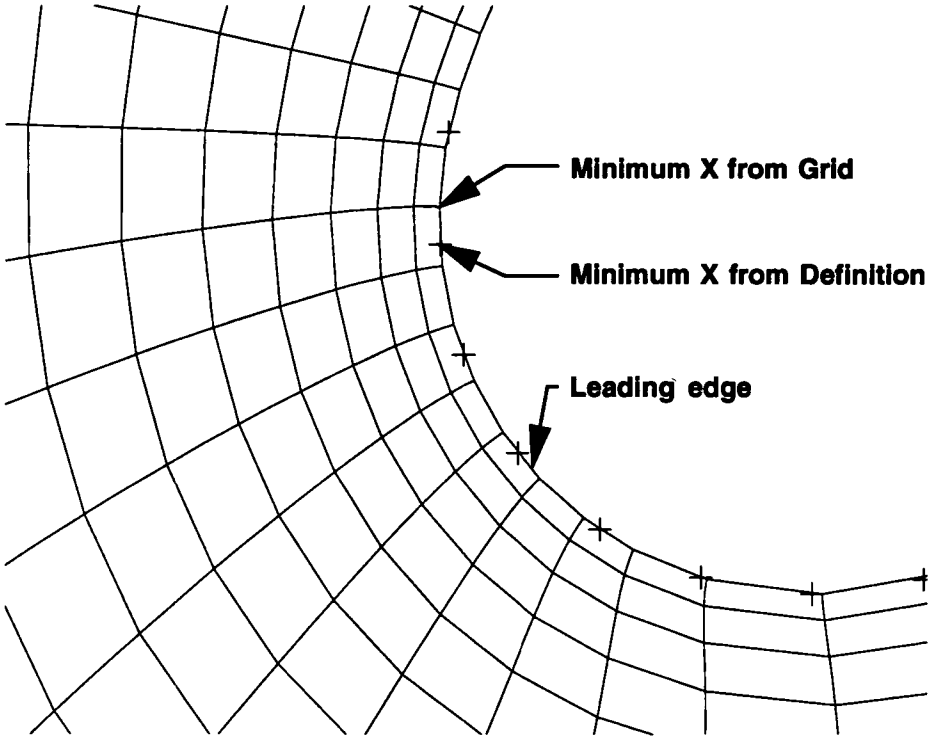
It should be noted that this procedure may not yield the same result as taking the minimum and maximum values from a grid generated on the same surface, Figure 4.1. The true extrema could be yet a third set of values. There is no requirement that the airfoil definition explicitly define the minimum or maximum axial extent of the airfoil, so small errors are introduced by using the the largest and smallest values to represent the meridional projection of the leading and trailing edges.

From the standpoint of the throughflow analysis, the error introduced is probably inconsequential. However, from a numerical standpoint, a number of potential problems arise. In the *TADS* system, there are many representations of the airfoil: the definition, the airfoil slices on the meridional streamlines, the blade-to-blade grids, the meridional projection in the throughflow grid, etc. Data is often transferred between the various representations by interpolation. Because the endpoints of the domain are different in each representation, interpolation errors are possible at the endpoints. This is of some consequence, since the largest flow gradients are frequently at the leading edge. *TADS* modules minimize the error introduced by interpolating along grid lines where possible, and by using a normalized airfoil chord when necessary. This essentially says that the leading edge in one representation is equal to the leading edge in another representation, regardless of variations in the (X, Y, Z) data which describes it.

INTIGG also requires the intersection points between the leading edge and the flow path, and the trailing edge and the flow path. Again, the airfoil description does not necessarily conform to the flow path; the description may not even span the entire flow path. Consequently, *INTIGG* finds the intersection points between the airfoil and the flow path by locating the intersection of splines through the given data, Figure 4.2. The upstream and downstream boundary locations of the grid are then computed using the hub axial chord, and the user specified fractional extent.

TIGGC3D treats the throughflow grid as three blocks: upstream of the airfoil, within the airfoil row, and downstream of the airfoil. *INTIGG* defaults to equal axial spacing within each of the three blocks. The spanwise spacing is determined by a user defined trigger which indicates whether a viscous or inviscid throughflow analysis is to be performed. The default is an inviscid analysis, and *INTIGG* prescribes uniform spacing in the spanwise direction.

Representation of Geometric Features on an Airfoil



- **Minimum X values from airfoil definition and grid are different**
- **Actual leading edge location may not exist in either description**

Figure 4.1: The various interpretations of geometric features must be carefully accounted for in the program modules.

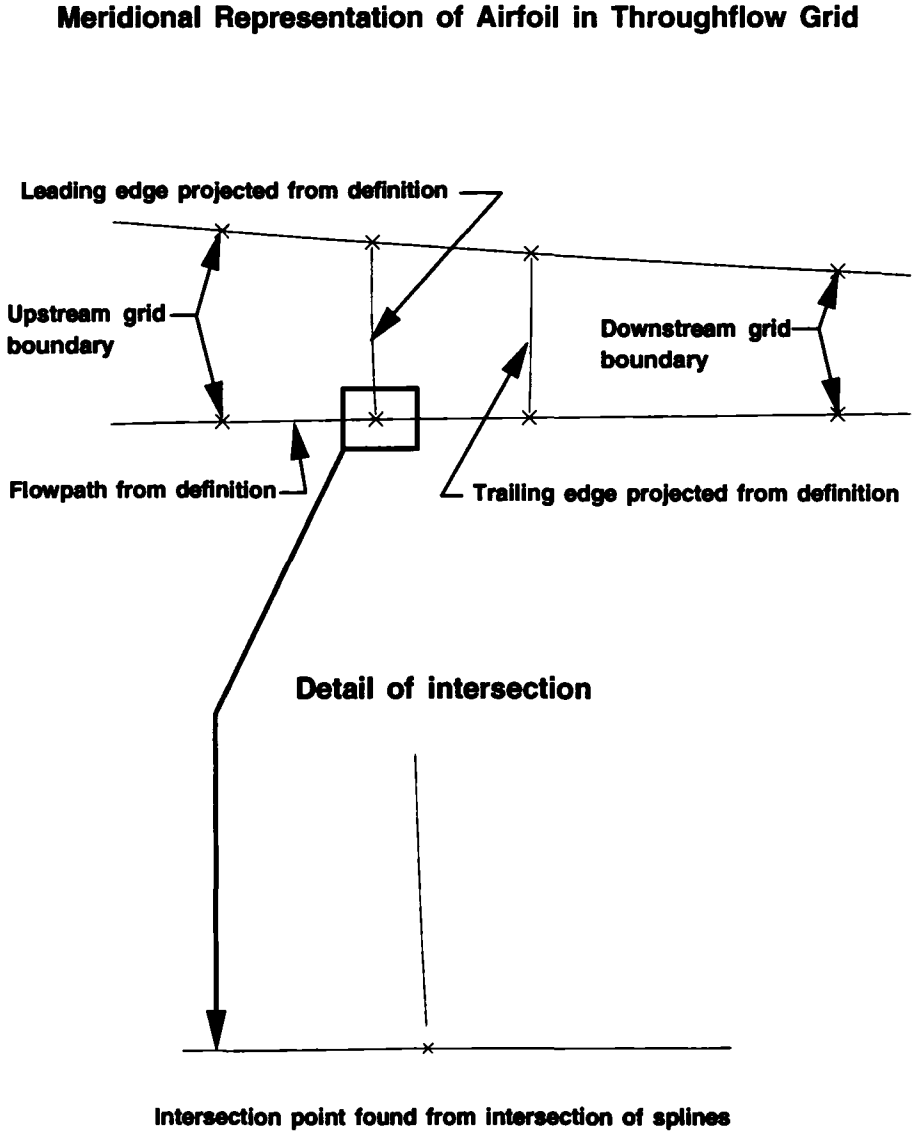


Figure 4.2: The grid extents and airfoil projection are computed from the definitional surfaces.

TIGGC3D writes out the final grid as a single block.

4.2 *TIGGC3D*

TIGGC3D is a 2-D/3-D grid generator for turbomachinery applications. It is a multiple block H-type grid generator with algebraic and some elliptic capabilities. *TIGGC3D* was originally designed to model multi-row core/bypass flows, and the input structure reflects this heritage. The *TADS* system uses *TIGGC3D*, version 5.2, as an 2-D axisymmetric grid generator for a single block algebraic grid. This capability is found in a related code *TIGGERC*, and is documented in Ref. [12]. *TIGGERC* was merged with *TIGGC3D* by NASA to reduce the code maintenance burden and to provide more capability in a single code. *TIGGC3D* is the only module aside from the GUI itself which uses graphics in the *TADS* system. *TIGGC3D* is also the only graphical module in *TADS* which does not use the Motif library under X-Windows.

The graphics in *TIGGC3D* use the Forms Library, Ref. [14] which, in turn, is programmed in Silicon Graphics GL. There also is an X-Windows version of the Forms library called XForms, or the Forms Library for X Ref. [22]. A *TIGGC3D* executable can be made with either Forms or XForms, but only the Forms executable has the intended look and feel.

Unfortunately, some of the drawing routines are programmed directly in GL. This is a limitation to porting *TIGGC3D* to other platforms which do not support the SGI GL graphics library. IBM offers a GL graphics board on its RS6000 systems, but the IBM implementation is not fully compatible with the SGI implementation. While the *TIGGC3D* executable can be made on an IBM workstation with a GL board, the graphics do not perform properly on the IBM.

TIGGC3D has a batch mode option, which does not call the graphics routines. This option is particularly useful on IBM RS/6000 systems where an executable can be made, but the graphics are not functional.

Other than the graphics related issues discussed above, the *TIGGC3D* code is used as received from NASA Lewis. Other versions of the code can be substituted, if necessary, without modification.

4.3 *ADPAC* Input Generation

The *ADPAC* throughflow analysis requires four files as input: a grid, a boundary condition file, a body force file, and an input file.

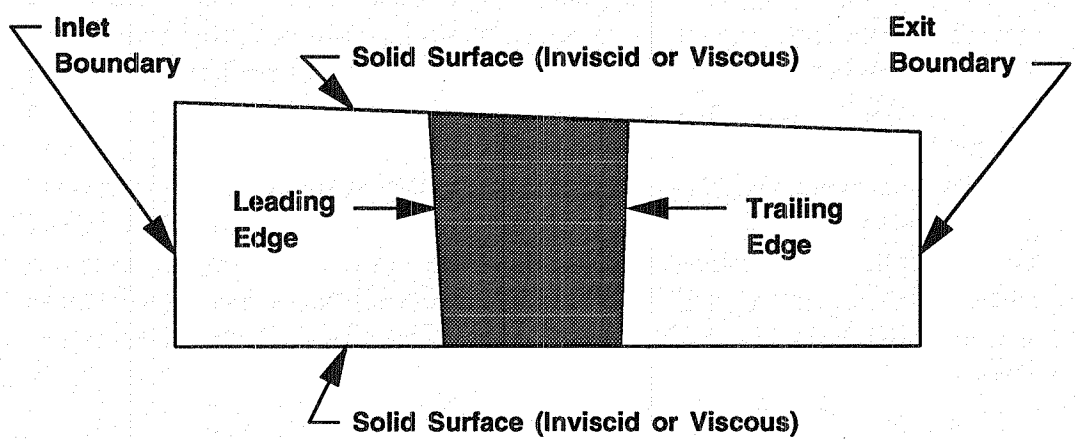
The input file is created by the GUI based on user choices in an input panel, or default values. The input file consists of execution control parameters and reference conditions. All *ADPAC* input parameters are described in Ref. [7]. Using the default parameters normally results in a successful throughflow analysis. However, the CFL number, number of time steps, and body force under-relaxation parameters are particularly useful for difficult cases.

The grid file is created by *TIGGC3D*, and must conform to the *ADPAC* naming convention, casename.mesh. If the batch version of *TIGGC3D* is used, the casename is set by default, but in the interactive mode, the user must type in the proper name when prompted.

The program *ADPACBC* prepares the boundary condition file for *ADPAC*. *ADPACBC* uses the axisymmetric grid, the user-supplied aerodynamic data, and the flow path description as input. *ADPAC* requires reference quantities which are used for non-dimensionalization. These are prescribed as the hub values of total pressure, total temperature and Mach number specified in the aerodynamic data file. For a throughflow calculation of a single airfoil, the *ADPAC* boundary conditions are depicted in Figure 4.3.

The implementation of the 1-D boundary condition extrapolation required careful attention to geometric issues. For example, the user specifies radial profiles of total pressure, total temperature and Mach number components at the leading edge. These profiles are accompanied by the appropriate radii. *ADPACBC* extrapolates the data from the leading edge (as defined by the aerodynamic data) to the upstream boundary of the grid. It is not correct to ratio the areas from the grid and the aerodynamic data file to enforce the conservation of mass. Because there is no requirement for the user data to span the flow path at the leading and trailing edges, the resulting areas may not be correct. This problem was solved by computing the normalized distribution of the points on the radial profile based on areas. This normalized distribution is then applied to the leading edge and the inlet boundary as defined by the grid. The ratio of areas is performed using only areas based on the grid, ensuring self-consistency. The exit static pressure is computed using similar techniques.

Specification of *ADPAC* Boundary Conditions



- **User specifies aerodynamic data at the leading and trailing edges as radial profiles**
- **ADPACBC extrapolates the data to the inlet and exit boundaries**
- **Extrapolation is according to 1-D gas dynamics, conservation of mass and angular momentum**

Figure 4.3: The *ADPAC* boundary conditions are set based on user supplied aerodynamic quantities and geometric considerations.

4.4 *BODYF*

BODYF creates the body force file for *ADPAC* and applies the mean stream surface shape to the axisymmetric grid. The input files for *BODYF* are the axisymmetric grid, the aerodynamic data file, the airfoil definition, and the mean stream surface file from *MEANSL* if available. *BODYF* is unique among the *TADS* program modules in that it expects to both read and write the axisymmetric grid file. There are no other program modules which modify a file read as input.

BODYF has two possible modes of operation: one is to create a mean stream surface from the mean camber line and possibly Carter's deviation angle rule, and the other is to interpolate a mean stream surface determined by *MEANSL* onto the axisymmetric grid. In either case, the blockage is computed and written to the body force file.

The blockage is defined at each grid cell center as the fraction of the total pitch open to flow. Except in the bladed region, the blockage is 1.0. In the blade region, the blockage is computed from the θ values on the pressure and suction surface at a given X and R. The difference between θ values is subtracted from the pitch, and normalized by the pitch to arrive at the blockage value.

4.4.1 Airfoil Thickness Determination

The airfoil description and the axisymmetric grid may have slightly different locations for the leading and trailing edges. To avoid interpolation difficulties between the different airfoil representations, a new procedure was developed. Figure 4.4 shows an axisymmetric grid and the blade geometry description projected on the axisymmetric plane. Both grids are defined in two parameters, where the indices i and j run in the axial and radial directions respectively. To determine the blade thickness values for the axisymmetric grid it necessary to interpolate the circumferential coordinate, θ, from blade geometry description.

The first step is to define a reference line which joins the leading and trailing edge points on the j=constant curves in the axisymmetric grid. Next, the radial differences between the reference line and the j=constant curve at each i station are computed. This radial difference is then splined versus the fractional distance from the leading edge (distance=0.0 at the leading edge

point and 1.0 at the trailing edge point) using a cubic spline. The next step is to define the j=constant curves in the axisymmetric grid on the projection of the blade geometry in the axisymmetric plane. Again, radial differences are computed from the same reference line used in the axisymmetric grid. This time though, they are calculated along i=constant curves at each j station. At each station, the fractional distance from the leading edge point is used to lookup the radial difference from the spline formulated for the axisymmetric grid. A difference of the radial differences is then calculated. A parameter is formulated along the i=constant curves which is the linear distance between ordered points. The blade coordinates (X and θ) are splined versus this length parameter and the length parameter is splined versus the difference of the radial differences. Where the difference of the radial differences is zero, the j=constant curves in axisymmetric grid intersect the blade geometry. Using this fact, the length parameter is easily determined from the spline of the differences versus the length parameter. The corresponding blade coordinates are looked up from their respective splines versus the length parameter. The final step is to formulate a spline of θ versus the fractional distance from the leading edge. This spline is then used to interpolate θ onto the axisymmetric grid. For generality, the procedure has also been coded to handle radial turbomachinery using a similar technique.

4.4.2 Mean Stream Surface Determination

The mean stream surface between airfoils is approximated by the mean camber line, in the absence of a computed stream surface from *MEANSL*. Originally, the mean camber line was approximated by the average of the θ values on the airfoil surface used for determining blade thickness. An improved procedure was later incorporated which computed the mean camber line as the locus of the centers of circles which are tangent to both the pressure and suction surface. The difference between these descriptions can be significant, especially near the leading and trailing edges, Figure 4.5. Of particular importance is the fact that the mean camber lined defined by a circumferential average passes through the minimum X point, and not through the true leading edge. The result is that the leading edge metal angle is distorted, especially at high setting angles, leading to incidence problems in the throughflow analysis.

The new procedure finds circles which are tangent to both surfaces at a

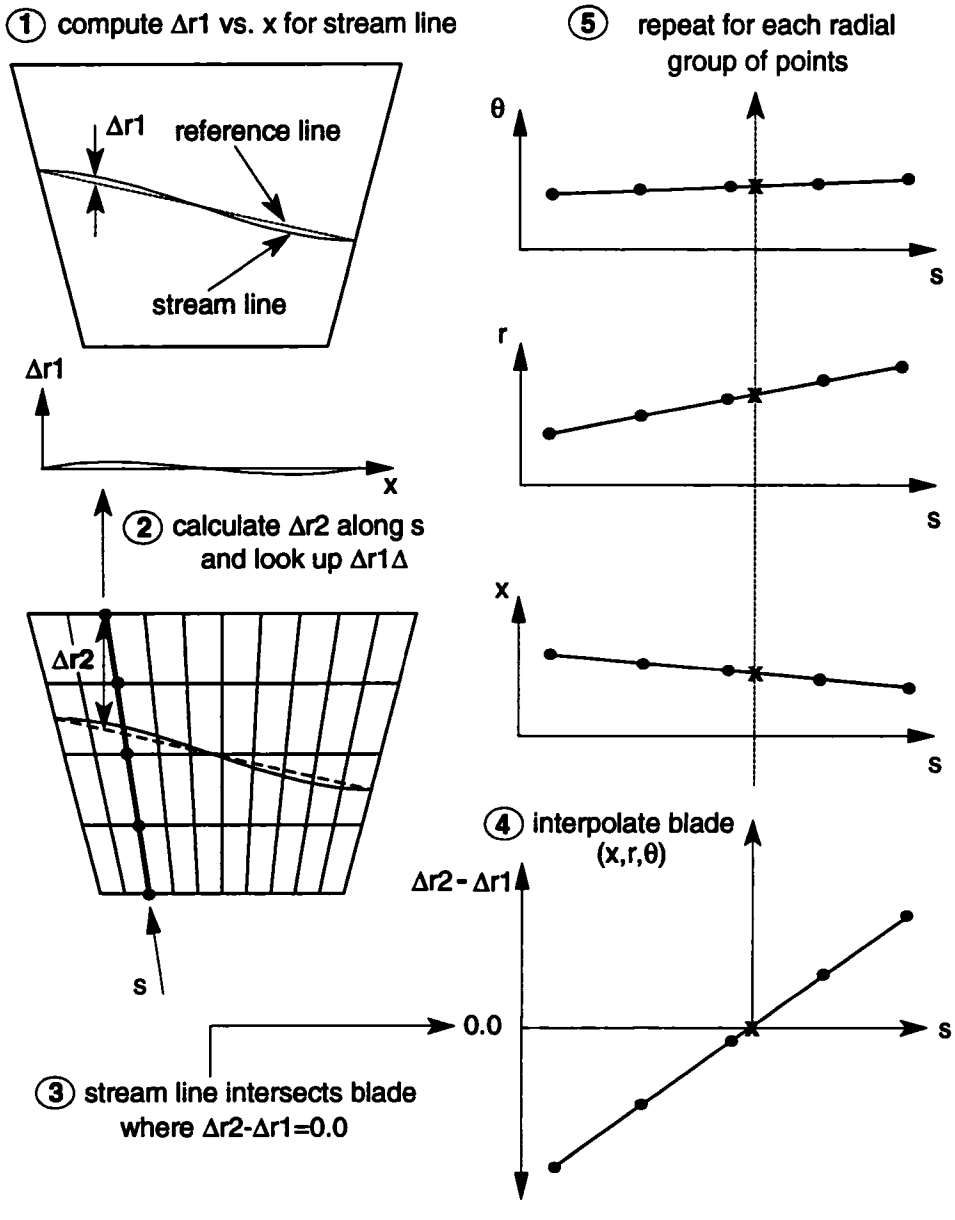


Figure 4.4: The airfoil thickness is determined by an interpolation procedure which handles differences in airfoil descriptions.

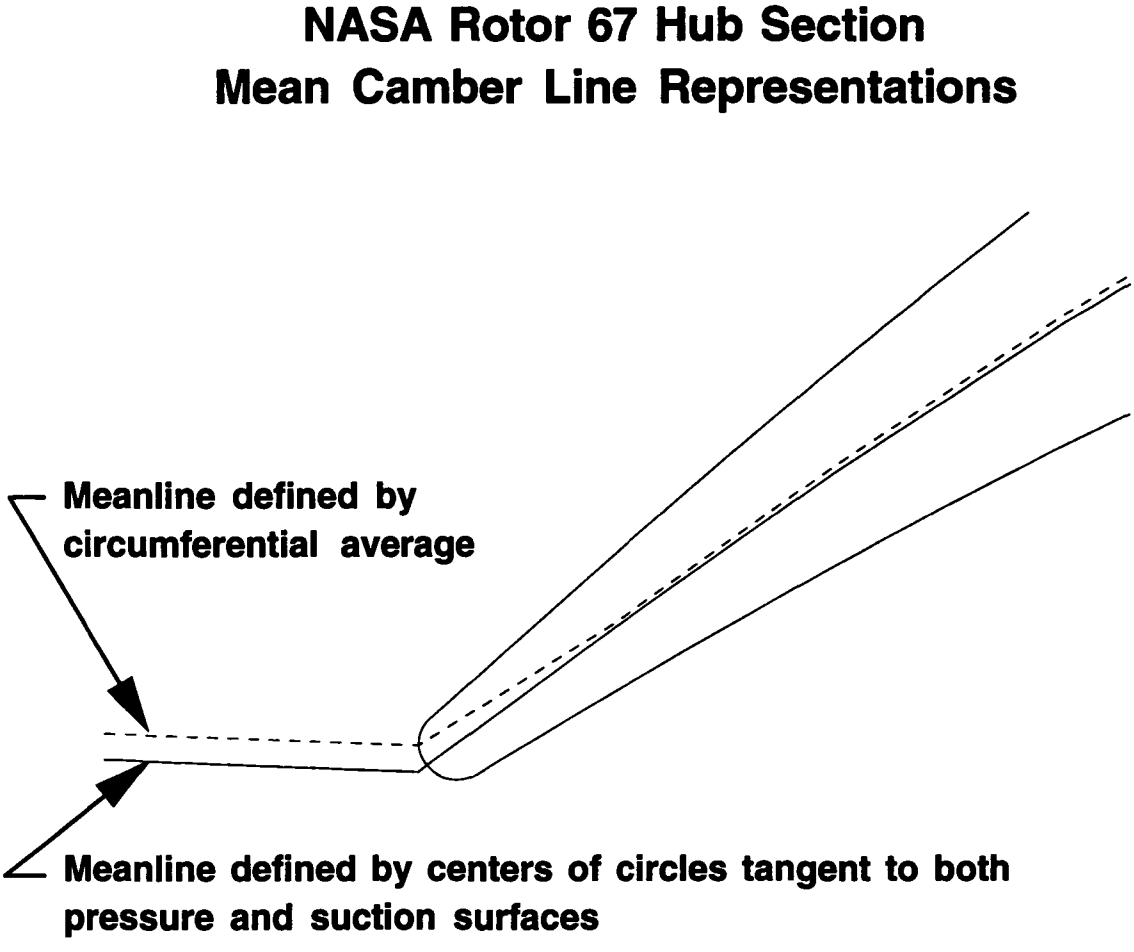


Figure 4.5: The procedure for determining the airfoil mean camber line strongly affects the incidence angle.

number of axial locations. The airfoil is considered to be made of three parts: the body, and the leading and trailing edges. The beginning and end of the body of the airfoil is determined from the tangency points. Only the body of the airfoil is used to determine the mean camber line. The leading and trailing edge angles are extrapolated from the spline of the mean camber line through the body of the airfoil. Using this procedure, a good representation of the mean camber line can be found, even for airfoils with non-circular leading and trailing edges.

4.4.3 Carter's Rule

Carter's deviation angle rule is often used in the design of compressor blades to account for the deviation of the mean stream surface from the mean camber line. Accounting for deviation with Carter's rule leads to more realistic throughflow solutions.

Carter's deviation-angle rule is a correlation which relates the deviation angle to the airfoil camber, solidity, the blade-chord angle (the angle between the blade chord line and the axial direction), and an experimentally derived factor. The details of Carter's rule are presented in Ref. [10].

Carter's rule specifies the deviation at the trailing edge, but does not specify the growth of the deviation along the airfoil chord. In the current work, the distribution is patterned after the method used in other design systems. Namely, the growth of deviation is specified as a parabola starting value at the trailing edge. This distribution is smooth and grows most strongly at the trailing edge, as is observed in experimental airfoil data.

4.4.4 Mean Stream Surface from *MEANSL*

The mean stream surface description found by *MEANSL* is defined only along the meridional streamlines from the blade-to-blade analyses. This description must be interpolated onto the full axisymmetric grid, which normally has more points in the radial direction. The interpolation is one-dimensional because the points in the *MEANSL* description of the mean stream surface are aligned with the radial grid lines in the axisymmetric grid. The interpolation assumes that the hub and shroud adhere to the same flow path. A linear interpolation is performed along the radial grid lines, using radius as the common parameter between the two representations.

4.5 *ADPAC*

ADPAC is a general multi-block 3-D Euler/Navier-Stokes solver capable of operating in either Cartesian or cylindrical-polar coordinates, Ref. [8]. *ADPAC* employs an explicit four stage Runge-Kutta algorithm to solve the finite volume representation of the governing equations, and uses a variety of convergence acceleration techniques, such as multigrid and implicit residual smoothing. While the existing *ADPAC* code could solve axisymmetric problems, it did not incorporate the blockage or body forces required for a throughflow analysis.

4.5.1 Body Force Implementation

At this point, some explanation of the various approaches to body forces is in order. The idea of using body force terms to simulate the presence of bodies in a flowfield is not new, nor is it unique to *TADS*. Recently, two main types of body force models have been employed in CFD codes.

A review of the literature shows that most previous authors add a force term to each momentum equation to account for the force exerted by the airfoil on the fluid. Frequently, these force terms are computed as pressure differences between the pressure and suction sides of an airfoil projected onto an element of area in each coordinate direction. Additionally, a blockage term is computed based on geometric quantities and is applied to the continuity equation. Any physical force could be modeled by these body force terms, simply by computing the magnitude and direction of the force.

In 1985, J. Adamczyk of NASA Lewis proposed a method of modeling the presence of neighboring blade rows in turbomachinery calculations with what he termed an "average-passage" representation, Ref [1]. In the Adamczyk scheme, the body force terms have a less physical interpretation. They are computed as the difference between an axisymmetric solution, and the axisymmetric average of a 3-D solution. A source term is computed for each conserved quantity and for pressure. A blockage term is also computed to account for the presence of the body in the flow. The source terms are not computed as forces acting on the faces of the control volume, but are accumulated as flux differences at each grid cell. In this procedure, the source terms automatically account for deviation and other phenomena which are not direct results of the pressure difference across the airfoil. However, this

procedure requires a full 3-D solution to compute the body force terms.

The present work follows a similar project in which researchers at NASA Lewis employed *VIADAC* as a throughflow solver, Ref. [11]. *VIADAC* and *VSTAGE* are two codes which use the Adamczyk body force approach. In *VIADAC*, the body forces are computed from stacked blade-to-blade solutions by the accumulation procedure outlined above. The original intent was to employ Adamczyk style body forces in an *ADPAC*-based throughflow analysis. While *ADPAC* does not have the full average passage algorithm, the coding already existed to create and use Adamczyk-style body force files. It was hoped that simply verifying the existing code would provide a suitable throughflow analysis. After further study, it was concluded that the original blockage/body force term implementation in the *ADPAC* code required some reformulation in order to be consistent with the design system strategy.

The original blockage/body force implementation in the *ADPAC* code was based on the scheme developed for the *VSTAGE* and *VIADAC* codes. This approach results in a coupled blockage/body force representation which did not permit accurate solutions for cases involving blockage alone without *a priori* knowledge of the flowfield. Consequently, it was not possible to impose a geometric blockage (such as the global effects on channel flow due to an internal strut) in the axisymmetric flow unless the resulting axisymmetric flow is already known. This is contrary to the design system philosophy, and resulted in the reformulation of the blockage representation.

A simple 2-D derivation of the revised *ADPAC* blockage term implementation is given below. Starting with the continuity equation in Cartesian coordinates modified for blockage represented by the term λ:

$$\frac{\partial \rho\lambda}{\partial t} + \frac{\partial \rho u\lambda}{\partial x} + \frac{\partial \rho v\lambda}{\partial y} = 0 \tag{4.1}$$

Next, taking the x momentum equation in nonconservation form we have:

$$\rho\frac{\partial u}{\partial t} + \rho u\frac{\partial u}{\partial x} + \frac{\partial p}{\partial x} + \rho v\frac{\partial u}{\partial y} = 0 \tag{4.2}$$

If we multiply the continuity equation by u, and add to λ times the x momentum equation, collect terms, and recast in conservation form, the result is

$$\frac{\partial \rho u \lambda}{\partial t} + \frac{\partial (\rho u^2 + p)\lambda}{\partial x} + \frac{\partial \rho u v \lambda}{\partial y} = p \frac{\partial \lambda}{\partial x} \tag{4.3}$$

Similarly, the y momentum equation becomes

$$\frac{\partial \rho v \lambda}{\partial t} + \frac{\partial \rho u v \lambda}{\partial x} + \frac{\partial (\rho v^2 + p)\lambda}{\partial y} = p \frac{\partial \lambda}{\partial y} \tag{4.4}$$

Finally, the energy equation is

$$\frac{\partial \rho e \lambda}{\partial t} + \frac{\partial u(\rho e + p)\lambda}{\partial x} + \frac{\partial v(\rho e + p)\lambda}{\partial y} = 0 \tag{4.5}$$

It is clear that the addition of the blockage term results in a source term which must be added to the solution scheme in order to properly account for the effects of geometric blockage.

The reformulated analysis utilizes a three-dimensional blade definition in the form of a mean camber surface (which must be accurately represented in the two-dimensional mesh) and a specified blockage (thickness) distribution over the bladed region. The body force utilized in the circumferential momentum equation is updated iteratively during the *ADPAC* time marching solution using a simple under relaxation procedure such that, at convergence, the resulting predicted relative flow stream surface is tangent to the local blade camber surface over the entire blade. The corresponding axial and radial momentum equation body force terms and energy equation source term are also updated consistently based on the components of the local blade surface unit normal vector. This implies that the body forces thus represent the idealized pressure forces imparted by the airfoil on the mean flow. The overall procedure is based on the analytical technique described by Damle, Dang, and Reddy [5]. It is relatively easy to upgrade the analysis to include more sophisticated body force models including the effects of local loss [5].

The *ADPAC* multigrid and grid sequencing capabilities were modified to incorporate the new throughflow analysis technique, providing a nearly threefold improvement in the convergence rate.

The final *ADPAC* code retains the Adamczyk capability, but also offers the reformulated approach. Both approaches use the same body force file format, but different meaning is attached to the variables. In addition to the source terms associated with the momentum and energy terms, there is also a

pressure "body force" term in the Adamczyk approach which is unnecessary in the reformulated approach. The *ADPAC* User's Manual, Ref. [7], explains the operation of these features.

4.5.2 Verification of Blockage Model

A sample application representing 2-D inviscid planar flow in a channel is presented in Figure 4.6. The channel has a linear variation in cross sectional area due to converging sidewalls. It follows that the blockage term λ should also have a linear variation from inlet to exit in the duct. In this example, λ was set to 1.0 at the duct inlet and 0.7 at the duct exit. Since the flow is inviscid and 2-D, the solution is essentially 1-D and can be determined based on area change and inlet Mach number alone. Due to the coupling of blockage and body forces in the *VSTAGE* and *VIADAC* codes, this type of flow cannot be accurately represented by specifying the geometric blockage *alone*. However, the predicted Mach number contours presented in Figure 4.7 based on the revised *ADPAC* formulation accurately reproduce the effects of the linear area variation with blockage specification only.

4.5.3 Verification of Body Force Formulation

Two test cases have been run to verify the body force terms: an annular twisting channel (S-duct) and NASA Rotor 67.

The S-duct, Figure 4.8, was chosen for its simplicity. It is an annular sector which has been twisted into a partial helix. A $49x9x9$ grid was generated for an Euler calculation. The duct has constant width, so no blockage is encountered. The solution was run as a static geometry (no rotation), and the pressure body force term was omitted from the calculation. The body forces were computed using a full 3-D solution from the *ADPAC-APES* (Average Passage) code, and used in an axisymmetric run of *ADPAC*. The *ADPAC* solution converged easily. Figure 4.9 shows a comparison of the resulting *ADPAC* solution and the axisymmetric average of the 3-D solution. Clearly, the body force terms are working as hoped.

NASA Rotor 67 provides a much more meaningful and difficult test of the body force formulation. An existing three-dimensional mesh was selected and altered to describe the airfoil in the mean stream surface/blockage format defined above. Computational results were collected from a 3-D solution

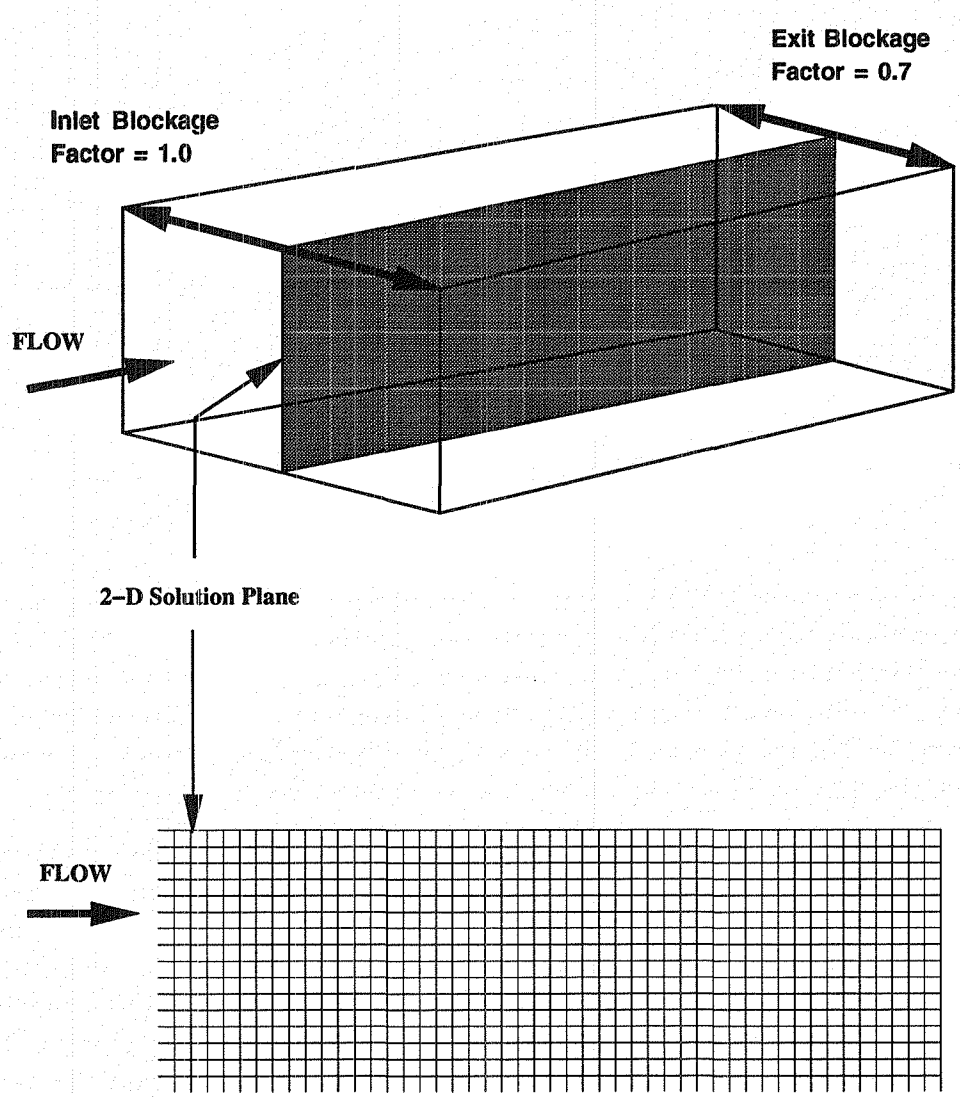


Figure 4.6: Simple channel flow with linear variation in cross sectional area results in a linear variation of the blockage term λ.

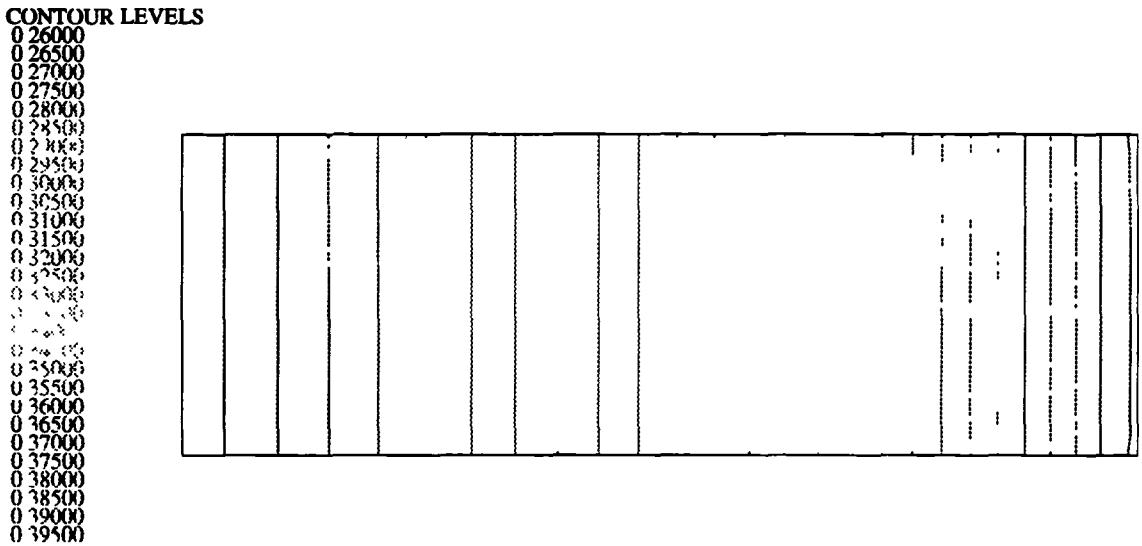
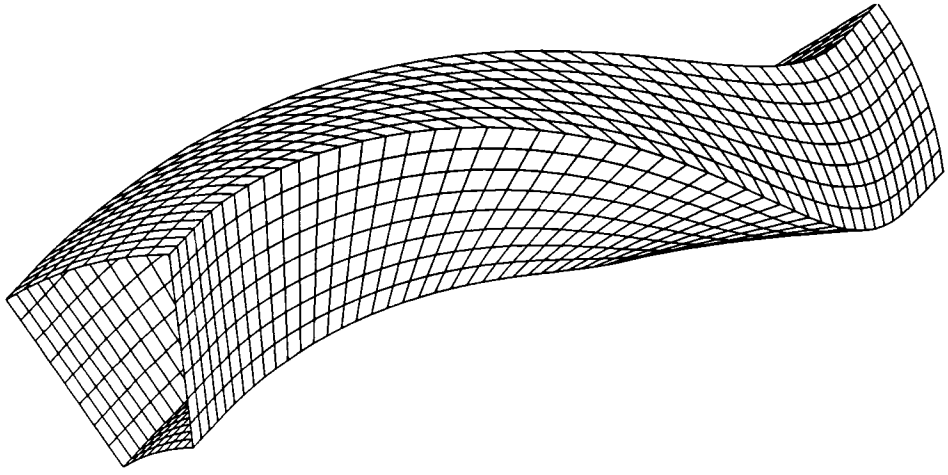


Figure 4.7: Predicted Mach number contours for simple channel flow with linear area variation using revised *ADPAC* formulation.

based on the original (3-D) mesh, an axisymmetric solution based on the apparent body forces computed from the 3-D solution, and the new throughflow analysis based on the mean camber surface mesh. It should be noted that the 3-D solution and the axisymmetric analysis with body forces computed from the 3-D solution result in, by default, identical axisymmetric flowfield representations, Therefore, only the axisymmetric solution is presented.

The axisymmetric representation of the mesh used for this comparison is given in Figure 4.10. For the axisymmetric solution utilizing body forces derived from the 3-D solution, the mesh can have any variation in the circumferential direction as only the meridional portion of the grid is used during the numerical solution. However, for the new throughflow analysis capability, the mesh must conform to the mean blade surface in the vicinity of the embedded blade row. The mesh surface is used to approximate the mean blade surface to properly update the body forces for the momentum and energy equations. In this initial set of calculations, the body forces for the new throughflow analysis were updated using an *ad hoc* under relaxation procedure defined by:

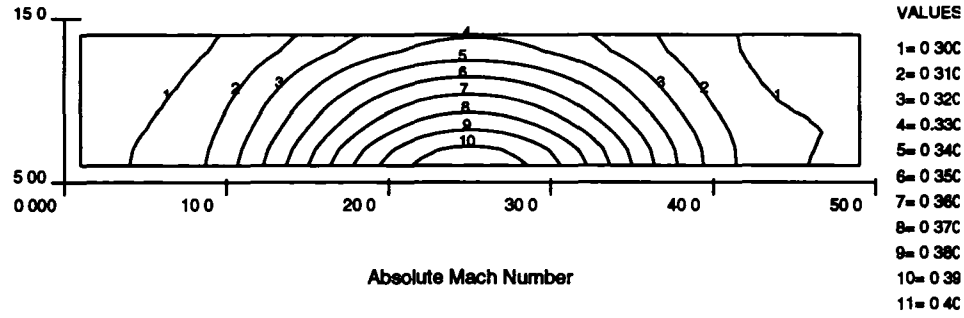
S–Duct Geometry



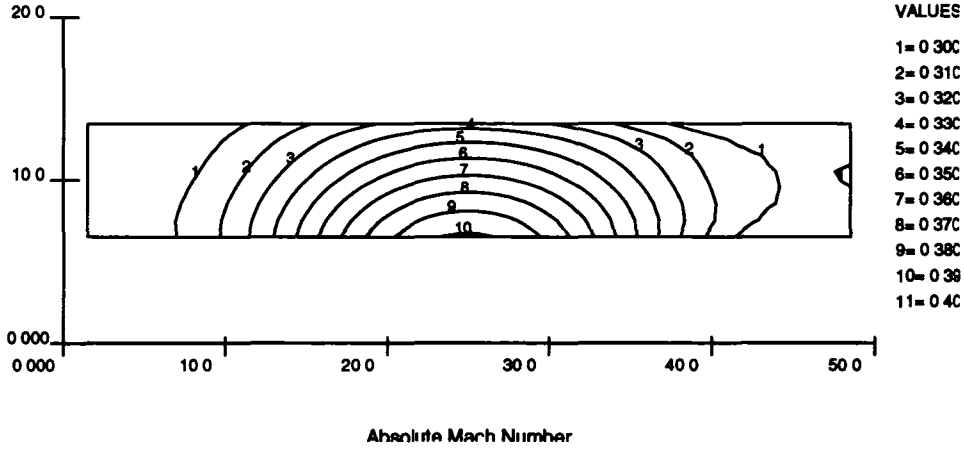
S–Duct is an annular channel with twisting. The inlet and exit are parallel to the machine axis so no body forces are present near the boundaries. The width is constant, so there is no blockage.

Figure 4.8: S-Duct geometry is a partial helix constructed from an annular sector.

Body Force Implementation in ADPAC, S–Duct



ADPAC Axisymmetric Solution with Body Forces



ADPAC–APES Axisymmetric Average of 3–D Solution

Figure 4.9: The axisymmetric solution with body forces and the axisymmetric average of the full 3-D solution are in good agreement.

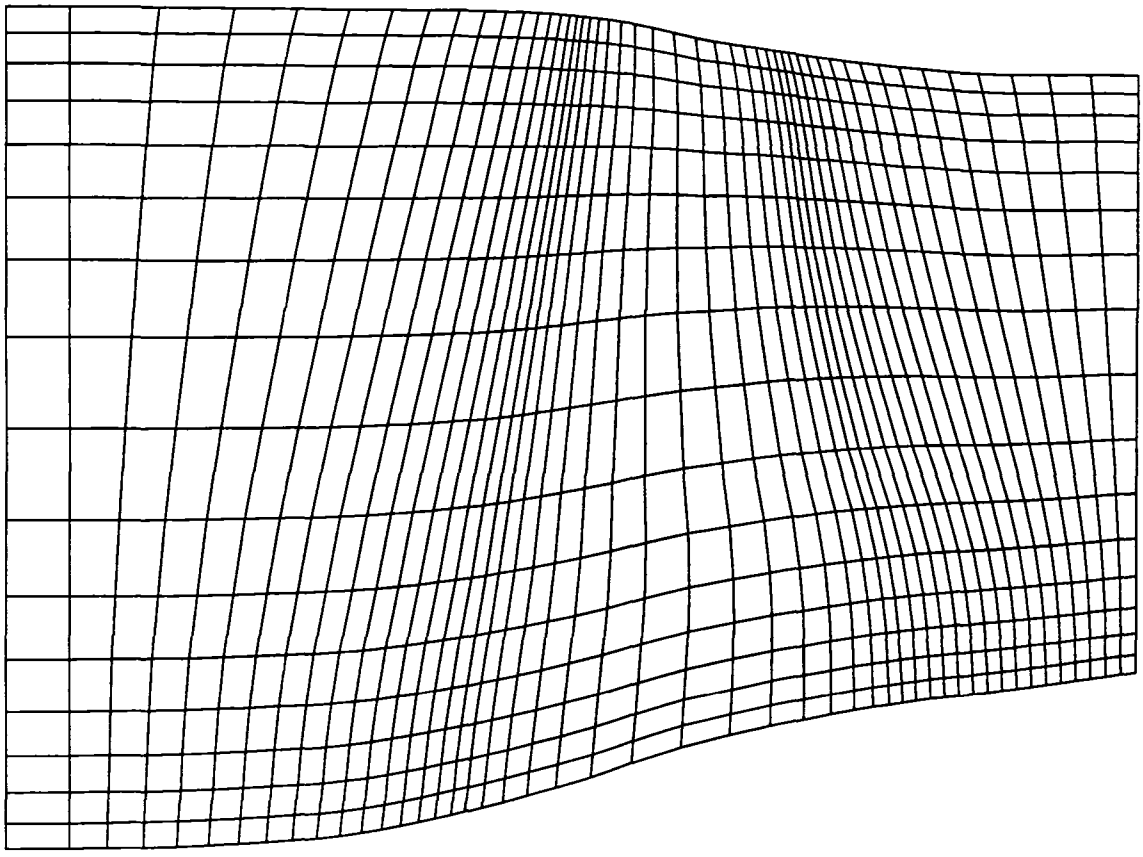


Figure 4.10: Axisymmetric Mesh System for NASA Rotor 67 Test Case.

$$B_\theta^{n+1} = B_\theta^n + \sigma(V_\theta^{blade} - V_\theta^{actual}) \tag{4.6}$$

where B_θ^n and B_θ^{n+1} represent the previous and updated circumferential momentum body forces, respectively, V_θ^{blade} is the apparent circumferential velocity required for flow tangency at the mean blade surface, V_θ^{actual} is the actual circumferential velocity from the flow solution, and σ is the under relaxation coefficient (0.5, in this case) used to update the body force. The body forces were updated at every iteration of the time marching solution.

The convergence history for the new throughflow analysis is given in Figure 4.11. Solution convergence was naturally slowed by the constant manipulation of the body force terms, but convergence is ultimately achieved after

approximately 400 iterations.

Figure 4.12 shows the predicted absolute total pressure contours using body forces from three different sources. The top plot shows the contours with body forces derived from the 3-D solution imposed on the axisymmetric solution. The middle plot shows the corresponding contours from the new throughflow analysis using the iterative body force calculation. The mean stream surface to which the flow was forced to be tangent was derived from the 3-D solution. Finally, the bottom plot shows the total pressure contours from the new throughflow analysis with the mean stream surface derived from the mean camber line of the airfoil and Carter's deviation angle rule. In general, the predictions compare well qualitatively, but show some discrepancy quantitatively. The top plot shows a smeared shock near the trailing edge because this solution is equivalent to an axisymmetric average of a 3-D solution. Since the shock is not aligned with the circumferential direction, the average tends to diffuse the shock. The center and bottom plots show a sharp shock at the trailing edge because the shock is axisymmetric, a consequence of the axisymmetric analysis. The bottom plot also shows a total pressure gradient at the leading edge. This indicates that the mean camber line is not actually the mean stream surface. Between the various solutions, the mass flow agrees within 2% of the axisymmetric average from the 3-D solution.

4.6 Streamline Finder and Airfoil Slicer

The blade-to-blade analysis is performed along streamlines in the meridional plane as found by the throughflow analysis. This requires that the meridional streamlines be located in the throughflow solution, and that the airfoil be sliced along these streamlines. *TADS* uses two separate programs to accomplish this purpose: *RADSL* and *SLICER.*

4.6.1 *RADSL*

RADSL locates the streamlines in the throughflow solution according to a distribution specified by the user in a GUI input panel. The user specified distribution is a normalized distribution which is applied at either the leading or trailing edge. The user selects whether the distribution is applied based

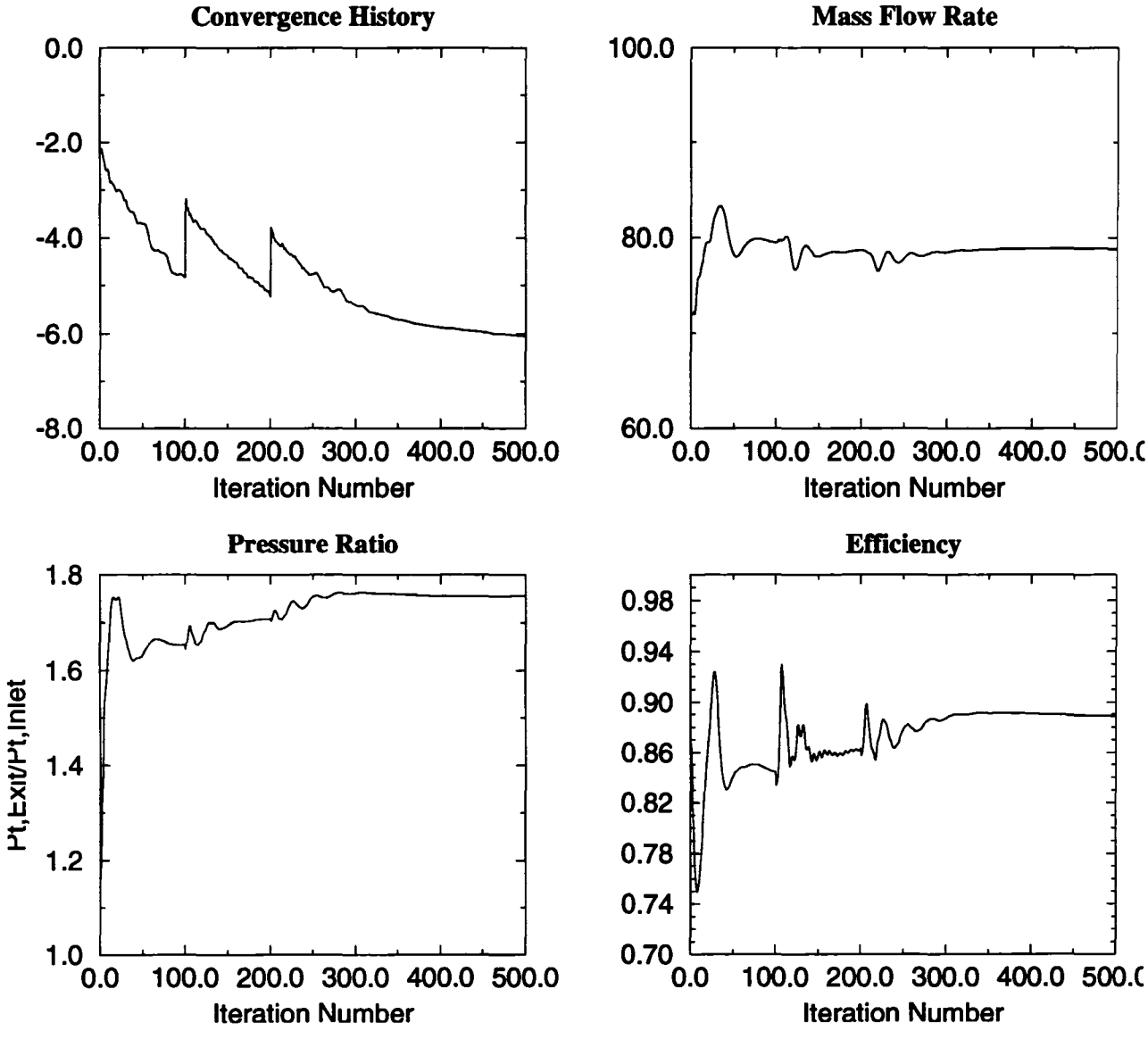


Figure 4.11: Convergence history for *ADPAC* based throughflow analysis applied to NASA Rotor 67.

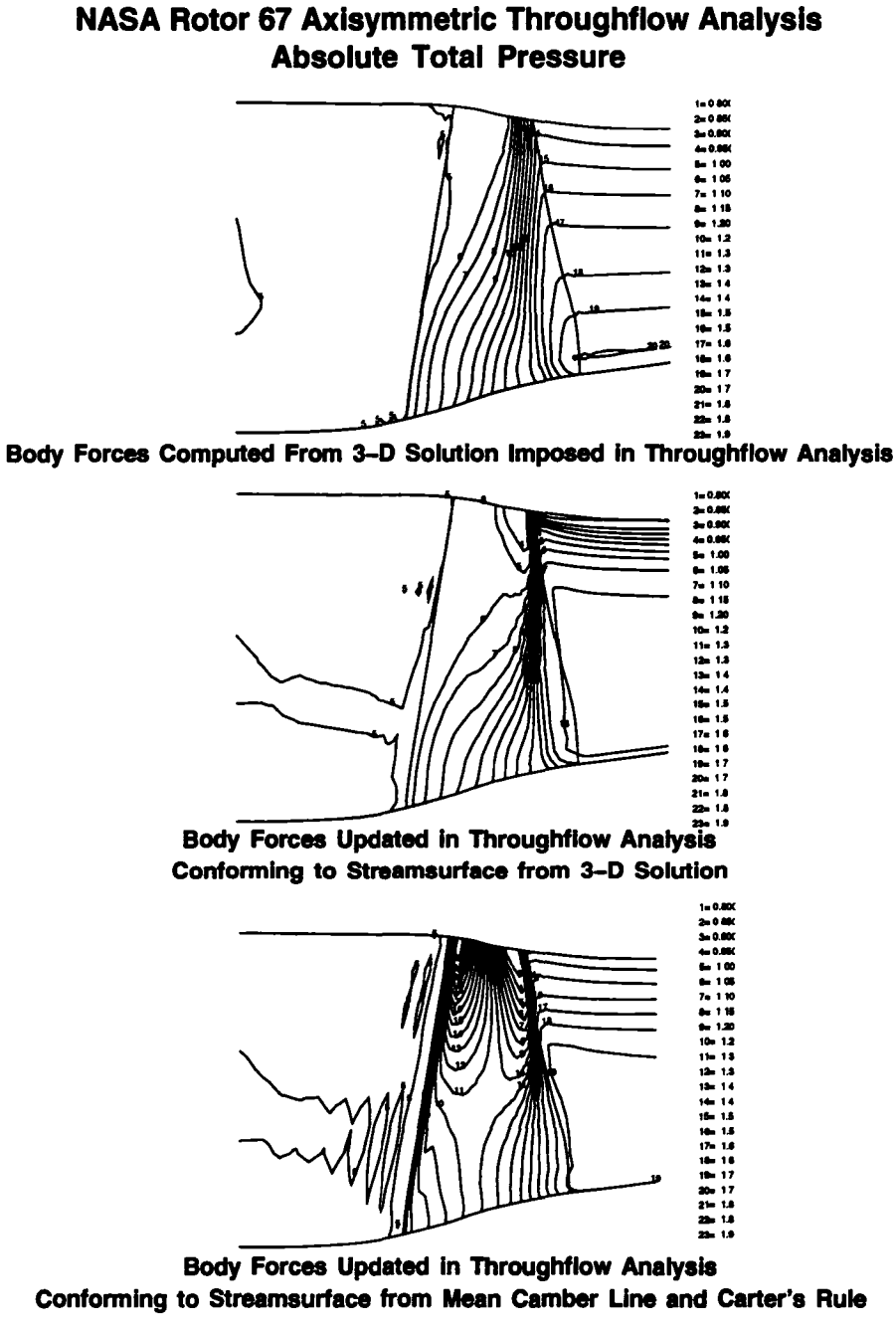


Figure 4.12: Predicted axisymmetric total pressure contours for NASA Rotor 67 based on *ADPAC* axisymmetric analysis with body forces from different sources.

on percent mass, percent span or percent area. The user selects the number of streamlines and the percentages where streamlines will be located.

For example, if five equally spaced streamlines are to be placed at the leading edge on a percent area basis, the procedure is as follows. The normalized mass flow is computed from hub to shroud at each axial grid station in the throughflow solution. At the leading edge, the values of mass flow are found, corresponding to the five locations: 0%, 25%, 50%, 75%, and 100% area. These streamlines are then traced through the entire domain. It should be noted that the chosen area distribution is applied only at the leading edge: elsewhere in the flowfield, the streamlines may not correspond to that particular area distribution. The percent span option functions similarly. If the percent mass option is chosen, then the the distribution is held throughout the flowfield.

There is one additional option: the user can find slices based purely on geometry, ignoring the flow solution. This option is triggered by selecting "Everywhere" as the location at which to hold the specified distribution. In this release, the only available distribution function is percent area. This option is useful in cases where the throughflow solution is suspect, or where there is some reason to want the blade-to-blade solutions along a constant area slice instead of along a streamline. The GUI input panel defaults to five equal slices at constant percent mass, with the streamlines anchored at the leading edge.

In all cases, the first and last streamlines are assigned to the hub and shroud as defined in the throughflow grid. User input which conflicts with this standard is ignored by *RADSL*

Finally, *RADSL* interpolates the throughflow solution onto the streamlines. The output file is a *PLOT3D* flow file whose dimensions are the number of axial points in the throughflow grid, and the number of streamlines. This information may be used by the blade-to-blade analysis to set boundary conditions. Because the blade-to-blade analysis acquires its boundary conditions directly from the throughflow solution, the throughflow calculation is normally run in Euler mode. It is not clear how to set the total pressure, temperature and flow angle on the hub and shroud, when the velocities are zero on viscous surfaces. The current version of *TADS* expects the throughflow analysis to be run as an Euler calculation.

4.6.2 *SLICER*

SLICER uses the original airfoil description and the streamlines found by *RADSL* to find the airfoil cross-sections to be used in the blade-to-blade analysis. *SLICER* also reads in the aerodynamic information file and interpolates flow conditions from the radial profiles onto the streamlines at the leading and trailing edges. This information may be used instead of the *PLOT3D* file interpolated from the throughflow calculation to set boundary conditions in the blade-to-blade analysis.

The process of slicing the airfoil along the streamlines involves repeatedly finding the intersection of two splines. Along each spanwise line in the airfoil definition, the intersection with each streamline is computed. The resulting airfoil description has the same number of points around the airfoil as the original definition. This airfoil description is used as the airfoil definition by the blade-to-blade grid generator. One limitation on the *TADS* system is imposed here: the spline along the span of the airfoil uses the radius as parameter. This means that centrifugal and radial devices cannot be handled by *SLICER*.

4.7 *GRAPE*

The blade-to-blade analysis uses the *GRAPE* code to generate a grid conforming to each axisymmetric surface defined by the meridional streamlines. *GRAPE* was originally written by Reese Sorenson at the NASA Ames Research Center as a 2-D Cartesian grid generator, Ref. [16] and Ref. [17]. The code was subsequently modified for cascades of airfoils by R. Chima of NASA Lewis Research Center, Ref [4]. *TADS* uses *GRAPE* to generate C-type grids which are later used by *RVCQ3D*. A GUI input panel provides choices and defaults for the important input parameters. The user selects the grid size and adjusts various parameters to improve grid quality.

GRAPE remains a 2-D Cartesian grid generator. However, a cylinder can be mapped directly into a plane by "unrolling." This is equivalent to using the quantity $Rcyl * \theta$ in place of Y. where $Rcyl$ is the radius of the cylinder. *GRAPE* can also be used for arbitrary surfaces of revolution by projecting the arbitrary surface onto a cylinder. The radius of the cylinder is set to the mean radius of the streamline. Further, the meridional distance is

substituted for the X value in the grid so that the grid is along the streamline. *RVCQ3D* expects the grid in this format, and remaps it to the proper radius internally.

A number of modifications were made to *GRAPE* for use in *TADS*. The output routine was rewritten to produce platform independent binary files by incorporating the SDB library. Also, user experience led to changes in some of the *GRAPE* input parameters. These changes make it easier to specify a set of defaults which yield acceptable grids over a wide range of shapes.

In the original code, some of the input parameters were inter-related. This was a source of user confusion, and proper handling of inter-related variables would require dynamic linkages between fields in the GUI. This capability is not available in the current release of *TADS*. In most cases, new parameters were introduced in the input routine, replacing similar parameters in the original code. The original parameters are then computed from the new parameters, leaving the internal workings of *GRAPE* basically unchanged.

For example, *GRAPE* originally had parameters for the number of points around the leading edge and the spacing between grid points around the leading edge. To increase the point density around the leading edge, the user needed to decrease the spacing parameter, and also increase the number of points around the leading edge. To create suitable grids from default parameters, the revised code expects the user to specify the leading edge arc length and the number of points around the leading edge. The arc length of the leading edge region is computed internally by the GUI from the airfoil tangency points, which are specified in the casename.tdsaro file. The user specifies the number of points around the leading edge, and the spacing is computed by *GRAPE*. This change removes the inter-dependence between variables, and simplifies user input by computing a reasonable default value for the leading edge arc length. A similar approach was taken with the trailing edge parameters.

The *GRAPE* code also requires the user to specify the grid index of the trailing edge. In a C-grid, there are two grid points which define this point, one on the lower surface and one on the upper surface of the airfoil. Originally, *GRAPE* required the user to specify both. Since the upper surface trailing edge index can be computed from the grid size and the lower surface trailing edge index, the upper surface parameter was eliminated from the input. The input routine computes the upper surface trailing edge index, and passes the value to the rest of the *GRAPE* code.

In the *GRAPE* code, the leading edge point distribution is set by clustering points around a certain point on the airfoil surface. This point is specified as a fraction of the arc length around the airfoil, starting from the trailing edge. This parameter is named *dsra*, and has a default value of 0.5. The default value clearly inadequate for sharp airfoils with camber, because the cluster point will be located on the suction surface, rather than on the leading edge. However, it is difficult for the user to choose the proper value for *dsra*. The *GRAPE* input generation subroutine computes an appropriate value for this parameter from the airfoil geometry and the airfoil tangency points. The leading edge is taken to be at half the arc length between the leading edge tangency points. Figure 4.13 shows a comparison between grids generated using the the default value of *dsra* and the value computed by the GUI for the hub section of NASA Rotor 67.

Finally, the original *GRAPE* code expected to receive the location of the upstream and downstream grid boundaries, specified in inches. These quantities are difficult for the user to specify, and different values should be specified for each meridional streamline to achieve suitable grid quality. Some other blade-to-blade grid generators locate the boundaries as a fraction of the airfoil axial chord or the pitch between airfoils. These parameters are an improvement, but user intervention is still required. For a compressor fan, for example, specifying the boundaries as a constant fraction of axial chord results in grids with too much space upstream of the leading edge at the hub, and too little space upstream of the leading edge at the tip. Conversely, specifying the boundaries as a fraction of the airfoil pitch results in grids with too little space at the hub, and too much space at the tip.

For the purposes of *TADS*, the boundaries are specified as a fraction of a distance. This distance is defined as the average of the axial chord and the airfoil pitch at each meridional streamline. In the cases tested, this has produced acceptable grids with minimal user effort. Two new parameters were introduced to *GRAPE*: *xupfrc* is the fractional distance of the upstream boundary, and *xdnfrc* is the fractional distance of the downstream boundary. Default values have been set for these parameters, but these may need to be adjusted depending on the shape of the airfoil (e.g. compressor blades normally require a smaller upstream fraction than turbine vanes). In *GRAPE*, the original parameters *xleft* and *xright* are computed from the new parameters and passed to the rest of the code.

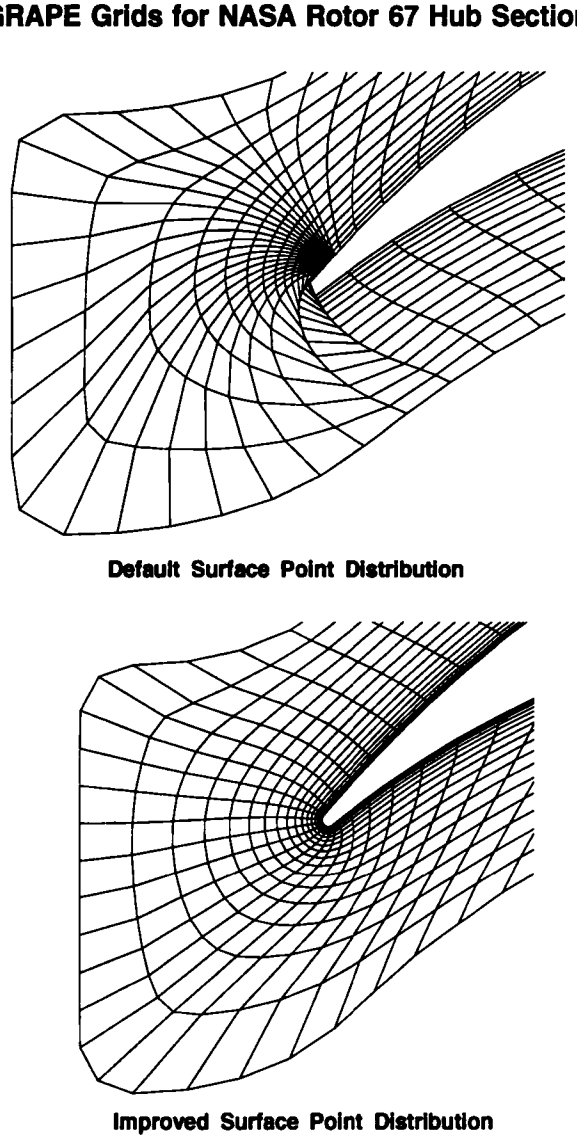


Figure 4.13: Comparison of airfoil surface point distributions in the *GRAPE* code.

4.8 *RVCQ3D*

RVCQ3D is an Euler/Navier-Stokes analysis capable of analyzing the quasi 3-D blade-to-blade flow in turbomachines, Ref. [2], and Ref. [3]. The input to *RVCQ3D* is specified in a GUI panel. *RVCQ3D* uses C-type grids generated by the *GRAPE* code. The input grid is not along the streamline, but is along a cylinder with radius corresponding to the mean streamline radius as described above. *RVCQ3D* also reads a table of values describing the radius and stream tube height distribution along the streamline.

The I/O routines in *RVCQ3D* were modified to utilize the SDB library in conformance with the *TADS* standard. Also, a change was made in the way that *RVCQ3D* sets boundary conditions at the upstream boundary. *RVCQ3D* expects to receive aerodynamic information at the leading edge and it extrapolates to the upstream grid boundary. The procedure is similar to the way that *ADPACBC* extrapolates data for the throughflow analysis. Since the blade-to-blade flow conditions are interpolated directly from the throughflow calculation, there is no need for *RVCQ3D* to perform an extrapolation. These modifications are limited and could be easily made to future releases of *RVCQ3D*.

4.9 Locating the Mean Stream Surface

Once the blade-to-blade analysis is completed, the last task is to determine the mean hub-to-tip stream surface between the airfoils. This task has two components: first the individual blade-to-blade solutions must be restacked into a 3-D representation, then the axisymmetric average of the solution must be computed, and the mean stream surface integrated from the averaged velocities.

4.9.1 *RESTACK*

RESTACK assembles the various blade-to-blade grids and solutions into *PLOT3D* X and Q files. This is a rather simple program: the only complication is in the conversion of data from the blade-to-blade representation to a true 3-D representation.

The blade-to-blade solutions are not computed on a true (X, Y, Z) representation of the data: the two dimensions are $(M, R \times \theta)$. These coordinates reflect what the flow actually "sees" along a streamline. Additionally, the velocities output by the throughflow analysis are (V_m, V_θ) The meridional coordinates and velocities must be converted to their 3-D cylindrical polar equivalents, and then converted to Cartesian coordinates for output. The streamline file written by *RADSL* provides the data needed to transform meridional coordinates back to 3-D cylindrical polar coordinates. The meridional velocity is converted to V_x and V_r by multiplying the meridional velocity by the unit vector tangent to the streamline. *RESTACK* is subject to alteration if other blade-to-blade analyses are incorporated into *TADS*.

RESTACK is programmed to expect data in the form written by *RVCQ3D*. In particular, *RVCQ3D* normalizes the aerodynamic quantities using a reference total temperature and pressure. For uniform upstream conditions, these reference quantities are normally set to 1.0, but radial profiles can be accounted for by setting different references on each streamline. *TADS* takes advantage of this capability. The hub streamline references are set to 1.0, and the other streamlines are set proportional to it according to the upstream profiles. No additional work is required to renormalize the flow on each slice to a consistent reference quantity when creating a 3-D file. The 3-D files created from *RVCQ3D* solutions are naturally self-consistent. Some other blade-to-blade solvers normalize the flow by setting the upstream total pressure and temperature to 1.0 internally. These solutions would have to be renormalized to a consistent reference before restacking.

4.9.2 *MEANSL*

MEANSL finds the shape of the mean hub-to-tip stream surface between adjacent airfoils starting with *PLOT3D* X and Q files. To perform this calculation, the grid and flow data are converted to cylindrical polar coordinates. The averaging is performed in the θ direction at axial locations chosen from the throughflow grid. The result is an axisymmetric averaged flow solution on a 2-D grid: one dimension is the number of points in the axial direction, the other dimension is the number of meridional streamlines.

The averaging procedure minimizes the dependency on the type or quality of the grid. *MEANSL* does the averaging as an accumulation of flow along a line, and not as an accumulation through 2-D faces. By formulating the

average along a line, the dependence upon neighboring slices is removed.

For each desired axial location along a streamline, two sweeps of the grid are performed: the first finds all of the intersections with the grid lines which wrap around the airfoil (contours), and the the second finds all of the intersections with the lines emanating from the airfoil (normals). The intersections are then sorted by θ, in the passage between adjacent airfoils. The axisymmetric averages are then computed by accumulating the fluxes along the sorted line.

This averaging procedure has a number of advantages. The procedure does not expect any particular grid topology, simplifying the job of adding different blade-to-blade analyses. The accumulated fluxes are comprised of as much data as possible because every intersection between the grid and the line of interest is used. Therefore, boundary layers or other flow features are resolved as well in the accumulation of fluxes as they are in the solution. This would be of particular benefit for blade-to-blade analyses with adaptive gridding.

The axisymmetric average data is used to determine the shape of the mean stream surface between the airfoils. The averaged velocities are, by definition, tangent to the mean stream surface. An integration is performed along each meridional streamline to find the shape of the mean blade-to-blade stream surface from the averaged velocities. The tangent to the mean stream surface is formed as the angle between the circumferential velocity and the meridional velocity. By integrating the angle with respect to the meridional distance along the streamline, a mean stream surface is determined. The output of *MEANSL* is a *PLOT3D* X file containing an axisymmetric grid, warped into the shape of the mean stream surface. This shape would be interpolated onto the full throughflow grid by *BODYF* to apply this stream surface shape in the throughflow analysis.

Chapter 5

Development of GUI

The Graphical User Interface (GUI) for the *TADS* system controls the operation of the program modules. It organizes the work flow into logical pieces, and provides a simple way to select or modify program input parameters.

5.1 Panel Overview

The GUI consists of a number of interactive panels with push buttons, pull-down menus, text fields, etc. These panels allow the user to select which programs to execute, create input sets for the chosen modules, and configure remote hosts on which modules can be executed. The GUI is written using the Motif widget library under X-Windows. Motif and X-windows are highly portable, having become a de-facto standard among workstation and supercomputer vendors.

5.1.1 Main Panel

A main panel controls the operation of all other panels within the GUI and all program module execution, Figure 5.1. There are three groups of buttons on the main panel: the group on the left is the "program mode selector", the buttons on the right are the "component group controls", and the buttons on the bottom are the "action buttons." The program mode selector determines the appearance of the main panel, and the behavior of the component group controls. The component group controls allow the user to make choices

regarding each functional task in the analysis. The action buttons allow the user to define remote hosts, open a UNIX shell, or exit the GUI.

There are five modes of operation available in the program mode selector. The selected mode determines how the GUI will respond when program modules are selected. The first mode, labeled "Edit Programs," causes the component modules to change appearance from push buttons to pull-down menus, Figure 5.2. The pull-down menus allow the user to select a program module to perform each task (e.g. *TIGGERC* or Batch *TIGGERC* can be chosen for the axisymmetric grid generator). At present, most component modules have only one working choice, but the capability was added so that users could easily incorporate their favorite grid generators and flow solvers into the *TADS* system. The program modes labeled "Edit Data," "Edit/Run," and "Run" cause the component modules to appear as either push buttons or toggle buttons. These modes control input creation and program execution of the component modules. In the "Edit/Run" and "Run" modes, a small green button labeled "Run" is enabled at the bottom of the component group controls as seen in Figure 5.1. The user selects which modules are to be run using toggle buttons to the right of each component. When all of the desired modules have been selected, the user selects the "Run" button to start the execution process. In the "Edit/Run" mode, the input panel for each selected module is brought up starting at the top of the component groups and working down. After the user finishes with the input panel, the program module is run. The program modules are run sequentially until all selected modules have been completed. In the "Run" mode, no input panels are brought up, the selected modules are simply run starting at the top and continuing down the component group.

In the "Edit Data" mode, the user selects push buttons which bring up the appropriate input panels, Figure 5.3. The input data is created and saved for that module only, and no execution is performed. The user may select these panels in any order. One strategy for running the GUI is to use the "Edit" mode to define all of the input parameters needed for each program module, and then the "Run" mode is used to execute the entire analysis. This keeps the user from having to wait for programs to finish before setting up the next program module.

Another strategy is to use the "Edit/Run" mode to perform the analysis piecemeal. It is frequently convenient to select only the modules associated with the throughflow analysis to be sure that an acceptable solution has

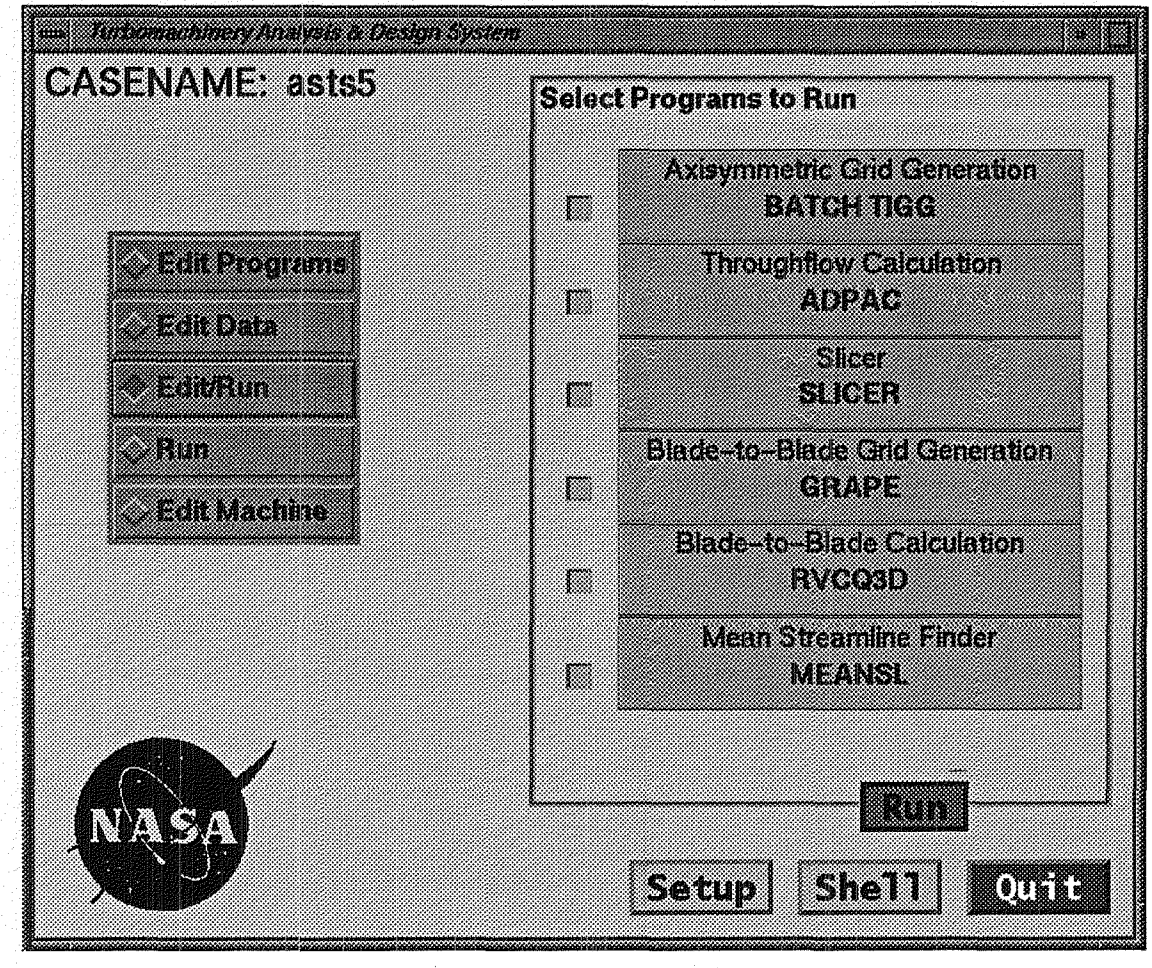


Figure 5.1: The Main panel of the GUI controls the complete analysis. The "Edit/Run" mode is shown here.

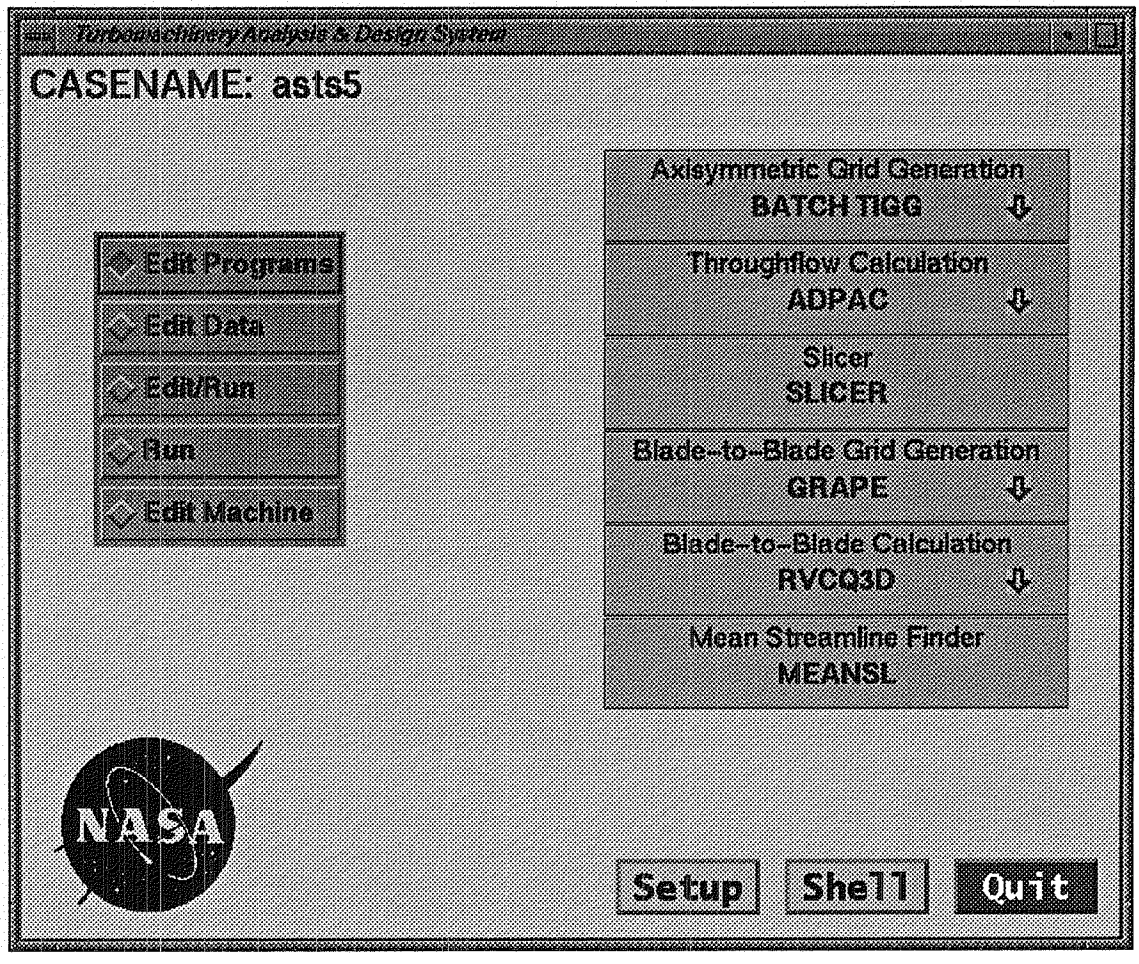


Figure 5.2: In the "Edit Programs" mode, the user selects program modules from a pull-down menu for each component of the analysis.

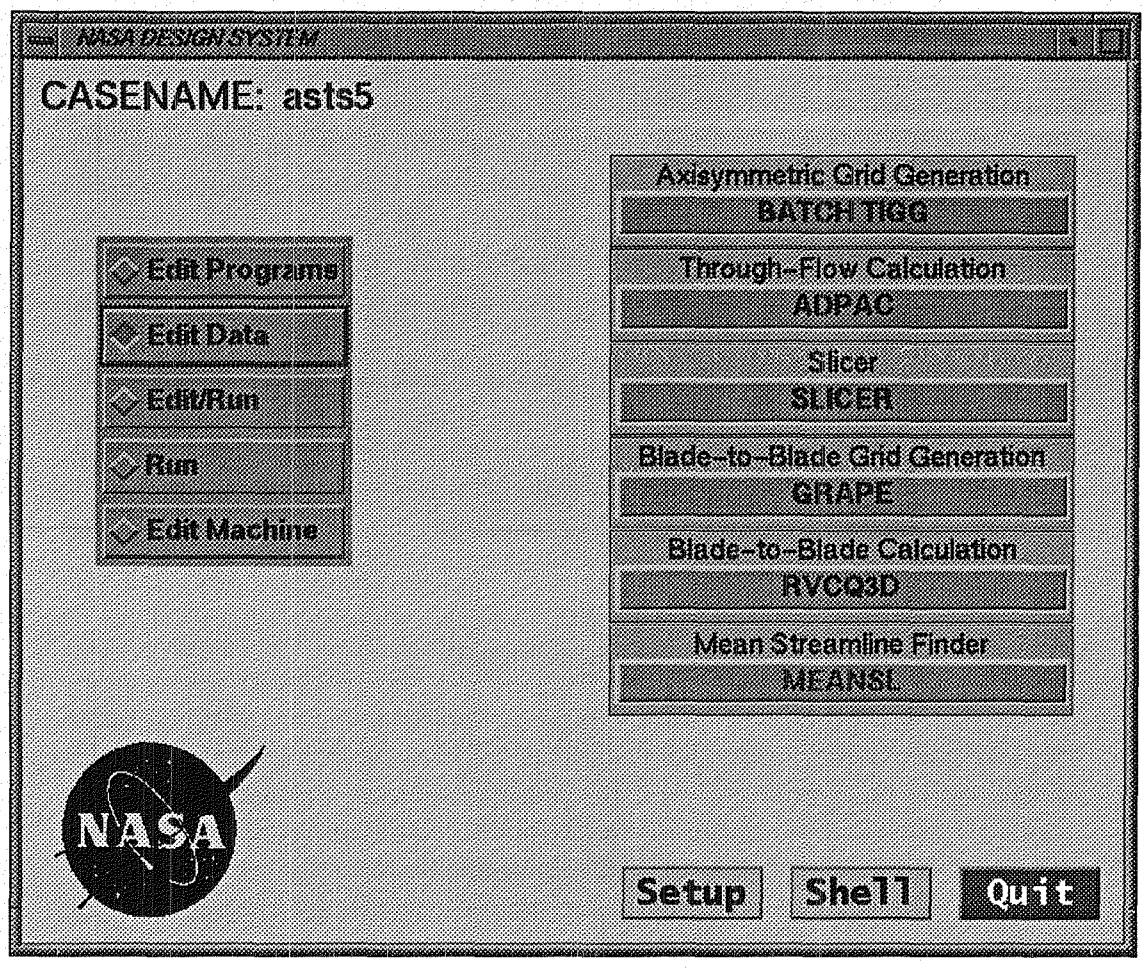


Figure 5.3: Input data panels for the program modules can be accessed from the main panel in Edit/Data mode.

been obtained before attempting to run the airfoil slicer and blade-to-blade modules. The remaining modules can be executed as a second step. The advantage of this strategy is that later modules will not have to be rerun because of errors in an early module. Because of its flexibility, the "Edit/Run" mode is the most common approach to controlling an analysis.

The final mode of operation in the main panel is labeled "Edit Machines." This panel is shown in Figure 5.4. This mode allows the user to select which host is to perform the calculations for each program module. It is often advantageous to run the longer running portions of the analysis (e.g. the throughflow and blade-to-blade flow solvers) on a remote machine to take advantage of faster processors. This option is only functional if hosts other than the local machine have been configured in the remote host setup panel. At present, all slices in the blade-to-blade analysis must be run on the same host.

In addition to the main panel, a status panel is created whenever the GUI is executed. This panel gives information about the function of certain buttons, and indicates when a program module is being executed. It displays the name of the module, the host on which it is being run, and the pathname to the current working directory. This panel is for display only, and no user input is accepted in this panel.

5.1.2 Remote Host Setup Panel

The action button labeled "Setup" opens a display panel for defining remote hosts, Figure 5.5. All modules within the GUI can be executed either on the local host or on a remote host. The remote hosts must be configured so that the GUI can call the appropriate executables in the appropriate directories. The text block labeled "Hosts" lists the available hosts for execution. Only hosts on this list can be accessed for remote execution. The radio button group labeled "Type" specifies the vendor and machine type for each host. At present, the panel has choices for Silicon Graphics and IBM RS/6000 workstations. There are two possible SGI choices to differentiate between the SGI R4000 chip and the R8000 chip. The SGI Power Challenge selection uses executables which have been optimized to run on the R8000 chip. The text boxes at the bottom right of the panel specify the paths to the executables and to the working directory for the highlighted host. Each host can have different paths for both executables and working directories. This was

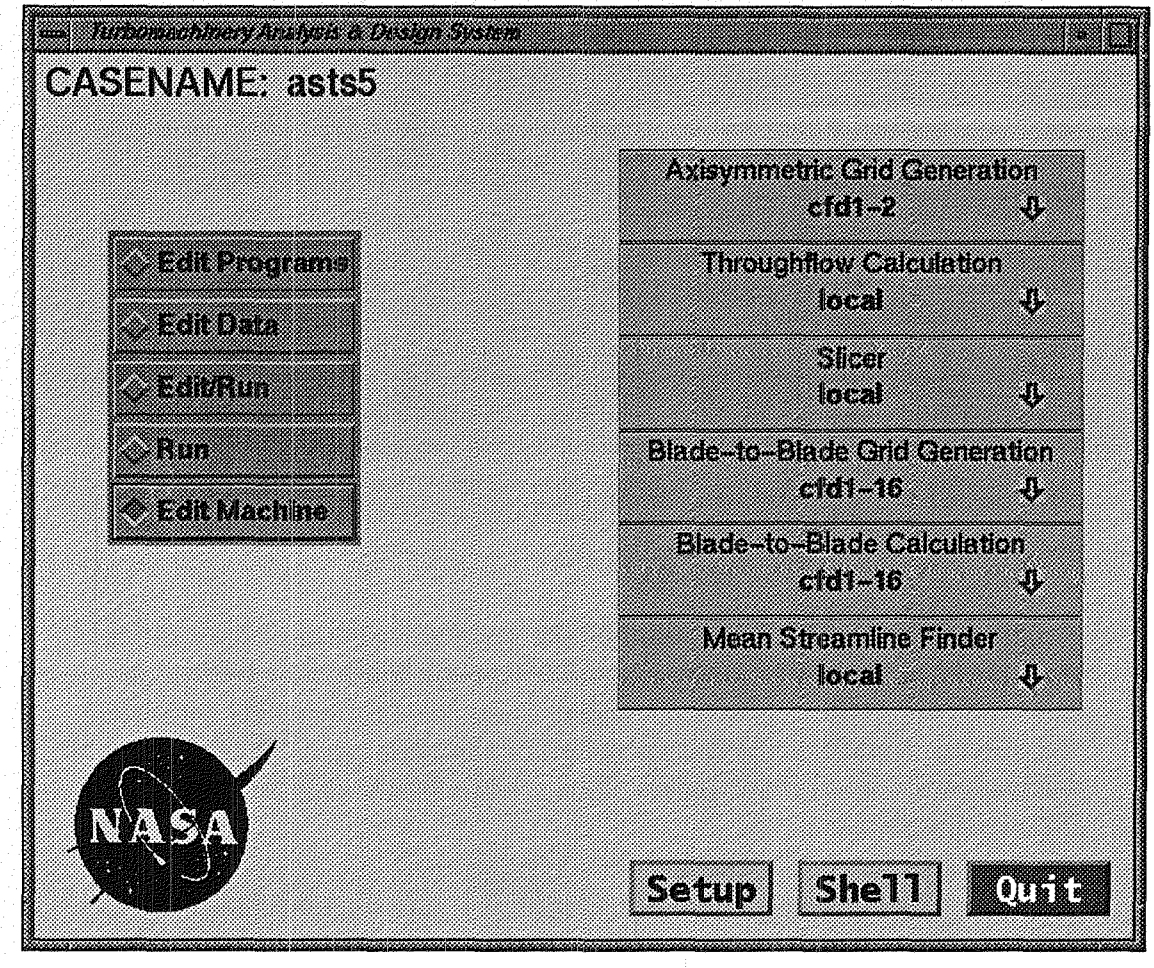


Figure 5.4: In the "Edit Machines" mode, the user selects a host processor for each program module.

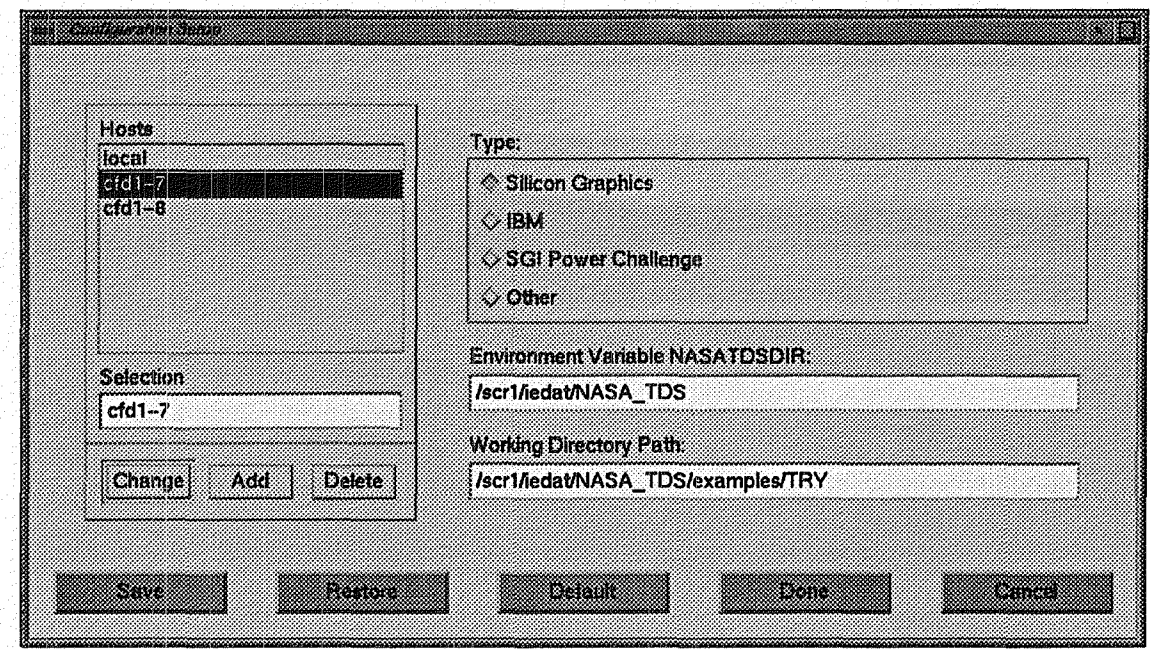


Figure 5.5: Program modules can be run on remote hosts configured using the Setup Panel.

designed to work with NFS mounted file systems which may have different pathnames to the same directories on different machines. The buttons at the bottom of the screen are action buttons which handle the saving and restoring of data, and allow the user to return to the main panel. A similar set of buttons exists in all input panels. The specific function of these buttons is discussed in Section

5.1.3 Input Panels

Most of the panels in the GUI are for creating input files for program modules. These input panels are similar in form and function, but some control multiple executions of the same program. Specifically, the panels associated with the blade-to-blade analysis have additional features to deal with the fact that the program modules they control must be run once per streamline. These panels, called "slice-dependent" panels, are discussed in the next section. Examples of simple input panels are the *TIGGC3D* input panel and

the *ADPAC* input panel, shown in Figure 5.6. The *GRAPE* and *RVCQ3D* input panels are slice-dependent panels.

An input panel is essentially a container widget which holds other widgets corresponding to input variables in the program modules. Action buttons at the bottom of the panel control the saving of data and closing the panel. The control widgets are most often editable text fields, but can also be pull-down menus or toggle buttons. The control widgets are laid out in a row-column matrix with labels indicating their significance.

Each input parameter has a separate controlling widget and label. Provision has been made to include a brief description of the highlighted input parameter on the screen as a reminder of its function. This reminder appears at the top of the screen, adjacent to the casename, and it changes with the input focus. This provision has not been fully implemented, but it is available in all input panels. All that is required is to add a text string for each variable in the GUI panel code.

In the case of text fields, provision has also been made for input data checking for valid types and ranges. For example, an integer field will not accept fractional entries or character data. Also, the entered value must lie within an acceptable range, or the entry is not accepted. An error dialog widget indicates the proper data range. In the input panel source code, a range of acceptable data is not required, and defaults to all inputs. The values typed into a text field are checked and accepted whenever the input focus changes.

Focus changes when the enter key, tab key or mouse input is received by the GUI. This does not mean that the data has been saved permanently, or that it will be written to an input file, but merely that it is part of the current data set. Data saving and input file creation are accomplished through the action buttons. The point here is that the user can create and view a complete input set before committing to the changes. Provision is made to abandon all changes made since the last save through the action buttons.

For variables with few options, pull-down menus and toggle buttons are employed. Toggle buttons are used in cases where the variable is either "yes" or "no," "true" or "false." Examples of this are triggers to generate a restart file, run viscous or inviscid, etc. The actual input variable may be an integer, but in each case, the input parameter controls an either/or choice.

Variables with limited options are well suited to the pull-down menu.

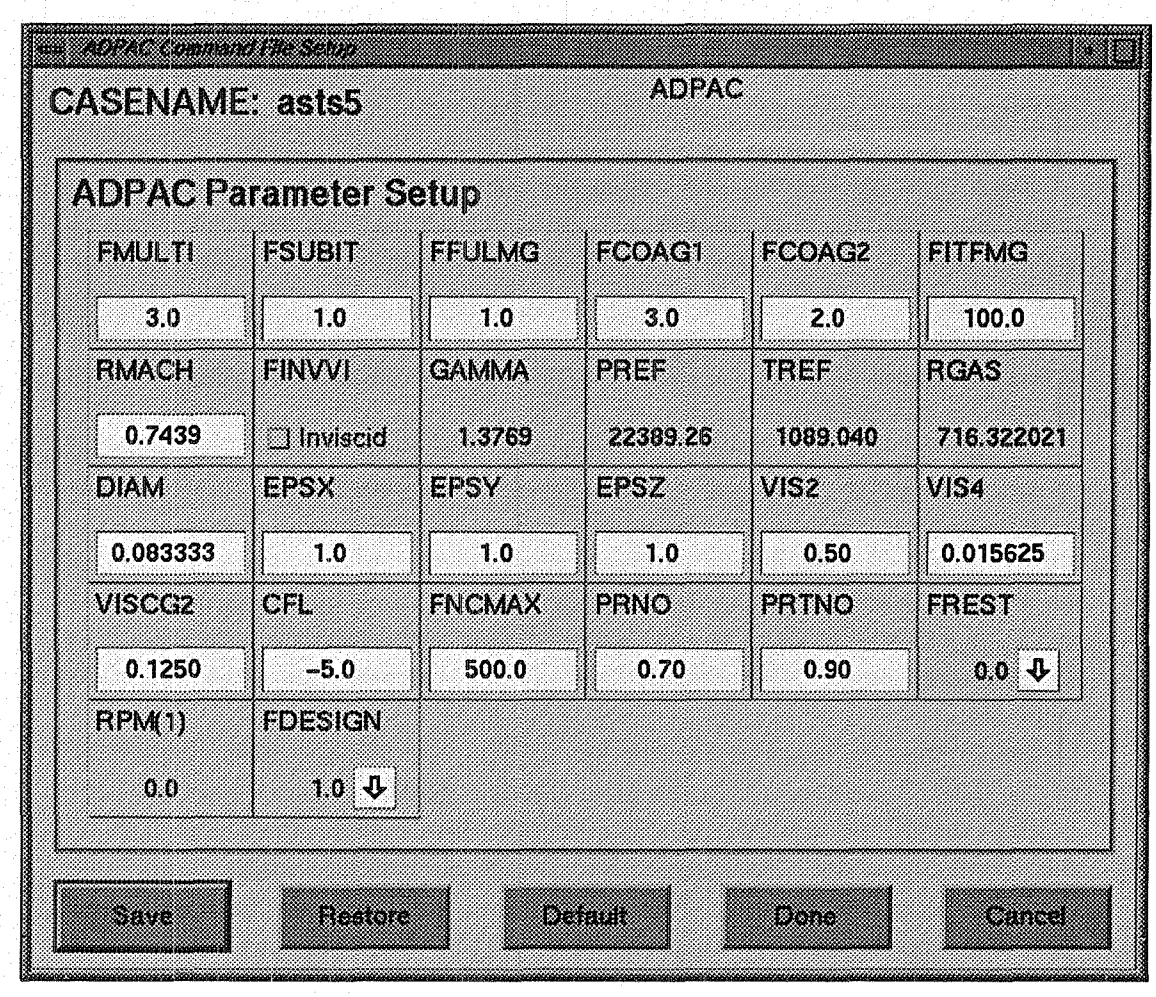


Figure 5.6: The *ADPAC* input panel is an example of a simple input panel.

Pull down menus display the values of the available choices, and a brief description of each choice. For example, the *RVCQ3D* input panel uses a pull-down menu to select the type of upstream boundary condition to be employed: subsonic flow holding inlet flow angle, supersonic flow, or subsonic flow holding circumferential velocity component. The description fields are especially helpful for variables which are rarely changed.

Each input panel has a default dataset which is part of the initialization code. This default data is the most basic default: other defaults are used when available. Some of the input panels have database files associated with them which keep track of previous user choices for a particular case. Other input panels use the input files created in previous runs of the same case. When available, data from these files are loaded into the input panel and form the initial data set. The idea is to minimize user input requirements by using the results of a previous run as the initial data set for the current run.

Some input variables in one program must be consistent with input to other programs. For example, the grid size for the blade-to-blade solver is set when generating the blade-to-blade grid. Therefore, the user is prevented from changing the grid size in the blade-to-blade solver input panel: the value input to the grid generator panel is displayed, but can't be edited. When a text box can be edited, the background of the box is white. When a text box is for display only, the background is the same as the background color of the container widget.

Where possible, the input parameters are grouped as they appear in the program module documentation, or in sample batch input files. This may be a drawback for inexperienced users, especially in cases where the organization of the input files is poorly conceived. For the user who is used to running the programs outside the GUI, it is beneficial to group them in the customary order.

5.1.4 Slice-Dependent Panels

There are a number of additional features and complications associated with slice-dependent panels. Figure 5.7 is an example of a slice-dependent panel. The most important aspect of slice-dependent panels is understanding how data is used and saved between slices. In simple input panels, there is no ambiguity; values are set and used in the normal manner. However, in dealing with slice-dependent panels, there are some variables which are the same for

all slices, and some which vary from slice to slice. For example, the number of blades on the wheel is a constant along the span of an airfoil, but the axial position of the inflow boundary may vary between meridional slices. It is important to know when a variable is set for all slices, and when it is set for only the current slice.

In slice-dependent panels, there is an additional widget in the upper right-hand corner which indicates which slice is being edited. This widget is a pull-down menu with an entry for each slice plus an entry for "All Slices." When "All Slices" is selected, the variables which are changed in the panel are set as constants for all slices. When data is saved, it is saved for all slices, any individual slice modifications are lost. A warning panel is displayed to alert the user, and a confirmation is required before data is overstored. In any event, only editable variables are propagated for all slices; parameters which are not editable are set internally for each slice.

Variables which are set individually for each slice are not editable in the "All Slices" view. When an individual slice is selected, only the variables which can vary among the slices are editable. When data is saved from the individual slice view, only the data for the current slice is affected. There are some variables which are frequently constant for all slices, but are sometimes slice-dependent. There is a provision for treating a single variable as either constant or variable among the slices, but most of these have been converted to slice-dependent variables to remove the confusion surrounding their use.

The slice-dependent panels make use of a relational database which is maintained for each slice dependent panel. The database files are random access binary files, containing the values of all parameters for all slices. The database files follow the naming convention casename.program_name.db (e.g. rotor67.grape.db). When data is saved, it is written to the database file, and when data is restored, it is re-read from the database file. When a slice-dependent panel is exited, new input files for the program module are created for each slice. Simple input panels do not employ a database, but rather work directly with existing input files, when available.

The recommended procedure for setting data in slice-dependent panels is to set the values for "All Slices" first. After saving the "All Slices" data, then select the individual slice panels which require modification. Save each of these panels and return to the main panel.

As before, not all parameters can be set by the user. Some are computed from known data (such as the number of blades and the airfoil pitch), and

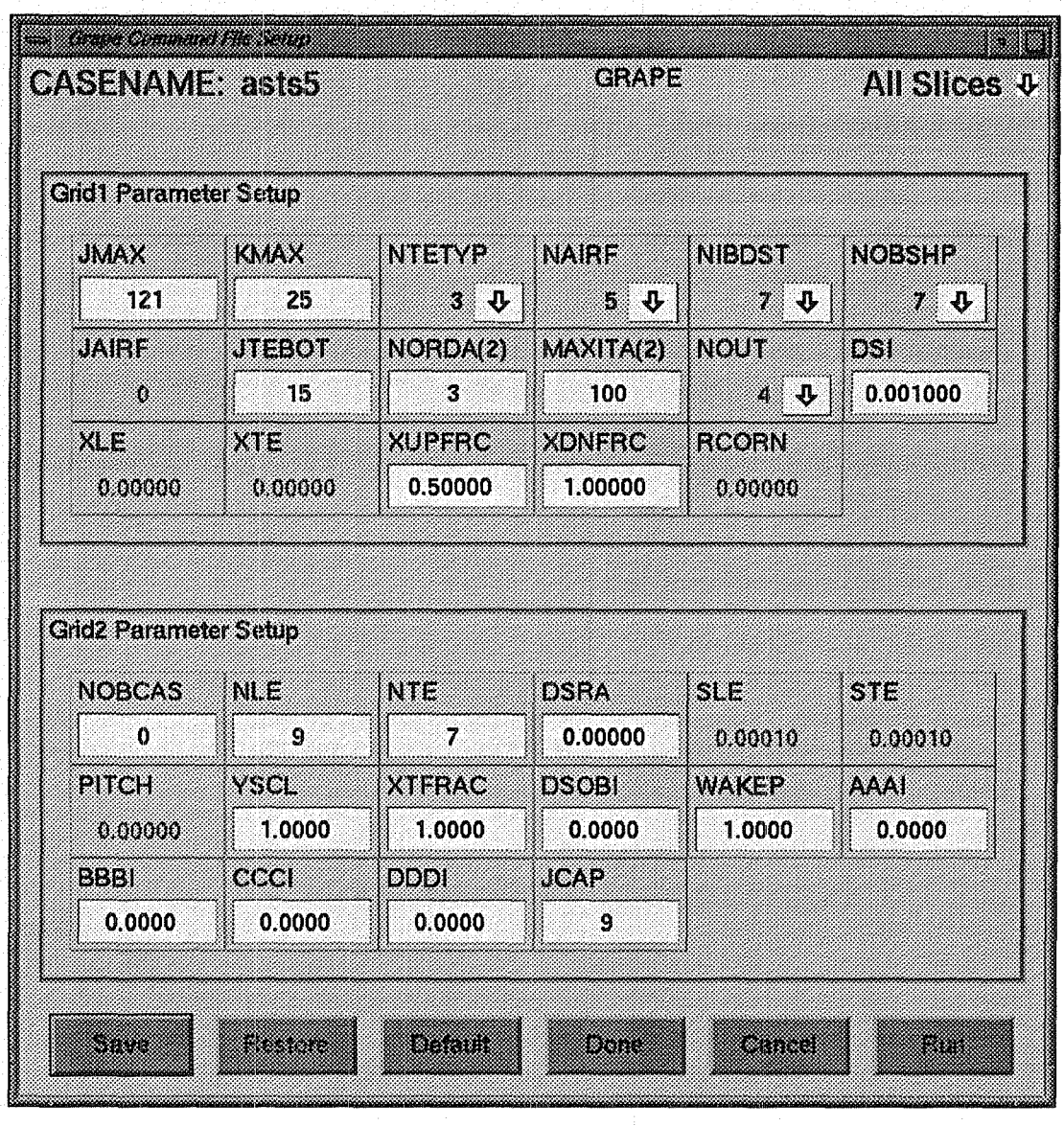


Figure 5.7: The *GRAPE* input panel is an example of a slice-dependent panel.

some are set in other panels and may not be modified (such as grid sizes, etc).

Another feature is provided in slice-dependent panels which is not available on simple panels. When viewing the input panel under the "Edit Data" mode selected in the main panel, an additional action button is displayed. This button, labeled "Run," allows the user to execute the program module for a single slice instead of for all slices. This is particularly useful when the user is unsure of the parameters chosen for the blade-to-blade grid generator or flow solver. Instead of waiting for all slices to run before discovering an input error, the user can execute a single slice and check the results before executing the other slices. This is also useful, for checking the sensitivity of an analysis to a particular parameter (such as incidence angle). A single slice can be run repeatedly without running any other slices. To avoid confusion, the "Run" button is de-activated when "All Slices" is selected from the pull-down menu. The button is activated only when the user is viewing the data for a single slice.

5.1.5 Action Buttons

All of the input panels in *TADS* have a row of action buttons located across the bottom of the panel. Generally, these action buttons control file creation and modification. Some buttons also initiate program execution. Generally, these buttons behave as described in Table 5.1. The few exceptions are documented in the User's Manual.

5.2 Programming Philosophy

The programming philosophy used in creating a GUI can make the difference between an intuitive, easily maintained interface, and a confusing interface built on tangled code. Recognizing the importance of standardizing the look and feel, the structure of routines, and the exchange of data between programs, the *TADS* system follows an object-oriented approach.

Conceptually, an object oriented approach means that the program modules are designed around the function they perform instead of the data on which they operate. Most codes are built around the data. This means that each routine is specific for the job it performs. In this model, it is often dif-

Table 5.1: Action buttons on standardized input panels control file creation, modification and restoration.

Save	Overstore current panel data to a file if changes have been made. If no changes have been made, then no action is taken.
Restore	Restore current panel data from a file. Any changes not saved prior to a **restore** are lost. This action button is only active if the input file exists (from a previous **save**).
Default	Reset current panel data to default values. These defaults are setup specifically for *TADS*. This means they are not necessarily the same as the defaults stated in the formal documentation of the individual component modules. Any changes not saved prior to a **default** are lost.
Done	**Save** current data and then exit current panel. In some instances, this action button will force the execution of secondary component programs such as preprocessors. Also, a message will appear in the message panel indicating any programs being executed.
Cancel	Exit current panel without saving current changes. If a **save** has been done prior to **cancel** secondary programs will be executed (if appropriate) as described above for **done** If changes have been made to the data without a **save** being done, the user will be so informed and given the option to return to the current panel.

ficult to re-use code because the data structure is embedded in the routine. A slightly different problem requires all new code. Under the object-oriented approach, the routines are written around the function they perform. Code re-use is planned from the start. The GUI is programmed in C, which is not an object-oriented language, but object oriented philosophy was adopted where possible.

An object oriented approach was used in generating the panels: each panel can be considered to be an instance of a model. That is, each panel is patterned after a model with changes only to the data to suit a particular use. The code interprets the data structure and creates appropriate objects for each input parameter.

To clarify the idea behind object-oriented programming, consider the following example. Suppose that two input panels are to be created. The first panel requires a pull-down menu for the first input item, and text fields for all others. The second panel requires a pull-down menu for the second and fourth items and text fields for all others. Traditional programming would write two separate routines to handle these cases. While much of the two routines would be common to both, custom coding would be used to handle the special cases. The traditional approach is data-oriented programming: routines are written specifically for the data that they handle. In the object-oriented approach, only one routine would be written, capable of handling each case. Each input item has associated data which indicates the desired type of widget. The code simply knows that each input item will require an object on the display panel. The type of object to be used is interpreted for each parameter. With the object oriented approach, the data structure is larger, but there is very little redundant code. A further benefit is realized in the object-oriented approach in that changes to the objects are automatically effective for all panels, minimizing code maintenance.

5.2.1 Panels as Objects

There are four model panels in *TADS*: the main panel, input panel, slice-dependent input panel, and the remote host setup panel. Each model panel has flexible data structures which are used in each panel of its type. A new instance of the structure is created for each panel, and the particular data is loaded into the structures, but the function and nature of each structure is the same in all panels. The data structures are comprised of many records,

one for each input parameter on the display. Included in the data structure is the parameter name, the value, the valid limits for the values, the type of widget to be displayed, and some information about initialization.

5.2.2 X-Windows/Motif Widget Implementation

The GUI is programmed with the Motif widget library running under X-Windows. As is customary with X-Windows/Motif applications, a resource file controls the colors, fonts, borders, and other aesthetic features of the individual windows and widgets. One weakness of the X-Windows system is that there is no standard way to refer to font names, and no guarantee that the fonts used by an application exist on a particular machine. In particular, SGI and IBM differ on the proper names for fonts. A separate resource file is provided for SGI and IBM implementations of the GUI. If other types of workstations are to be used, there may be some modification required to achieve a working set of fonts. One point of confusion is when the GUI is run on a remote machine with the panels displayed on a local machine. In X-Windows, the fonts are resolved on the local machine. That is, if the user is sitting at an SGI workstation, the SGI resource file should be used, even if the GUI is run as a remote process on an IBM workstation.

Most of the objects which appear in the GUI panels are conglomerations of Motif widgets. There are many instances where widgets were combined or customized, but the following four examples are most often used. The ability to enable or prevent editing of a parameter was required to prevent users from specifying contradictory input. Part of the data structure determines the conditions under which a particular parameter is editable. A special widget was made which contains both a text entry field and a label. The ability to group widgets was required in the airfoil slicer input panel, Figure 5.8. Pulldown menus were customized to cause the background color to change when the widget is enabled or disabled. In each case, the underlying routines for the screen objects are pure Motif widgets. Following the object oriented philosophy, new objects were created from existing objects to minimize coding and maximize the clarity of the main routines.

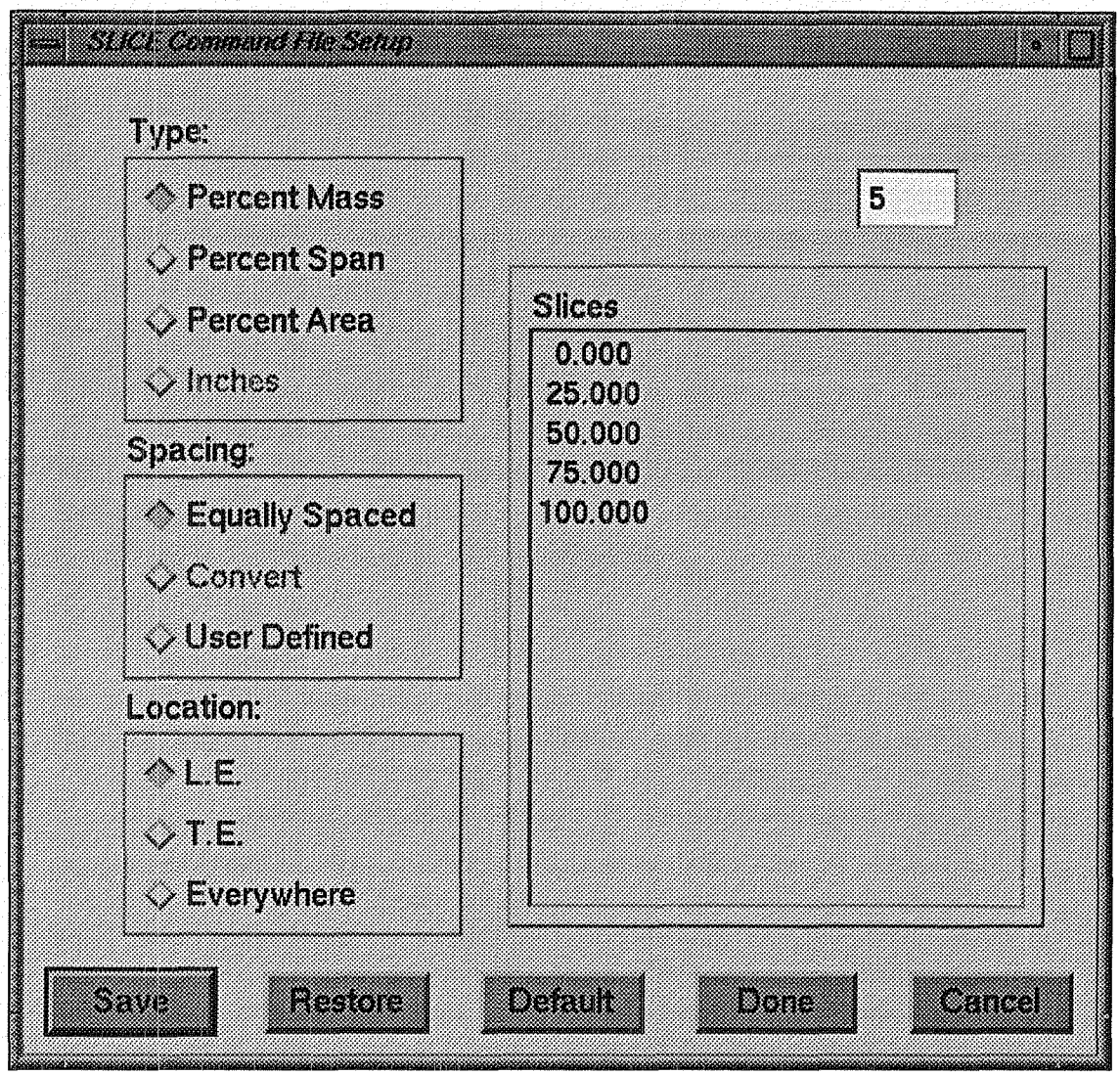


Figure 5.8: The Slicer panel of the GUI enables the user to control the location of the meridional streamlines for blade-to-blade analysis. Radio buttons are grouped and interconnected to insure consistent input.

5.2.3 Scope of Data

A common issue when coupling codes into an integrated system is that of the scope of data. The basic question is: "If a parameter is changed in one routine, do all other routines receive the changes?" Most parameters are strictly local. The advantage of local parameters is that there are no unintended side-effects. Often, two programs will have a variable of the same name with different meanings. Local variables keep the modules isolated.

Certain parameters have been identified within the GUI which have global scope. These parameters are available to all routines within the GUI. Among them are the number of airfoils, grid sizes, reference total pressure and temperature, and the wheel speed. The global parameters are listed in the routine globals.c. There are other parameters which are shared between routines, but are not global in scope. Most data sharing is accomplished through I/O in shared files. An example of this sharing is the axisymmetric grid. Many routines read the grid as input, and two routines write out the file. This type of data sharing is not truly global in that only routines which read the file receive updates to the data.

This means that it is a simple matter to generate a new panel of a given type. The changes consist of filling the data structure with the input parameters for the particular application, and adding a new stanza to some conditional blocks to show the new choice on parent menus. New stanzas must be added to the call-back block to show how the application is executed, and some parameter statements need to be added to a header file. Adding a new panel can be accomplished in about two hours if a suitable model exists.

Chapter 6

Modification of TADS

The *TADS* system is built on program modules with data transfer via files and flexible data structures. This architecture was adopted to minimize the effort required to extend or modify the system. The *TADS* system is divided into two parts: the GUI and the program modules. The program modules are loosely coupled to one another through files and are separate executables from the GUI. The GUI is more tightly coupled with data sharing through C structures. Object oriented programming concepts were employed to maximize modularity in the GUI. The program modules written specifically for the *TADS* system are modular, but the flow solvers and grid generators are used as received from the authors. Details about the program modules are found in the chapter "Analysis Coupling". The GUI calls the program modules via the C "system" function, which forks a new process as a child of the GUI process in the UNIX system.

6.1 Program Module Modifications

Program modules can be added to the *TADS* system, but some modification to the GUI and the module source code will be required. This section deals with the modifications required to the program module itself.

The required modifications to program modules are normally straightforward. The program module should perform I/O to named files following the casename.extension standard, should read and write mesh and flow data to *PLOT3D* style files using the SDB library, and should take all required input

from files, rather than from screen input. All I/O that does not use the SDB library should be ASCII text.

Of course, there are exceptions to the above rules. The blade-to-blade analyses are run in subdirectories of the main directory, and the file naming convention is relaxed in the subdirectories. Also, some programs are inherently interactive (e.g. *TIGGC3D*), and naturally require keyboard and mouse input.

Program modules with their own graphics or graphical interfaces are a special case. The ideal situation is for graphics in a program module to be programmed in X-Windows using the Motif widget library. These programs will be fully portable across all machine types supported by the GUI itself. Programs using strictly XForms graphics calls are also portable. Program modules with Silicon Graphics GL or other proprietary graphics library routines will generally limit the portability of the module. Obviously, portability is not an issue in homogeneous systems of workstations. Also, GL applications can be run on remote SGI machines so long as they are displayed on a local SGI machine.

Currently, there are very few places in the *TADS* system where the user can specify contradictory input between program modules. One objective of any extension of the system should be to prevent contradictions with existing data or programs. This could easily occur for program modules with their own graphical interfaces. For example, *TIGGC3D* has its own interface and takes most of its input from a file. When *TIGGC3D* is executed, the user must specify the name of the input file to load the data, and must also specify the name of the output grid. These names must be the ones that other program modules expect in the *TADS* system, or the other program modules will not find their input files. For example, the *ADPAC* flow solver expects the mesh to be in a file called casename.mesh. There is no simple means to enforce the *TADS* requirement for file names in *TIGGC3D*. This is a fairly minor point, but it illustrates how two uncoupled interfaces can lead to multiple specifications of the same parameters and contradictions between modules. If a new program module calls for interactive input of data which is already known to the GUI, a mechanism needs to be developed for the GUI to output the required information to a file, and for the program module to use the contents of that file as the default values in its interface. Otherwise, the user must be educated about the connections between the new module and existing modules in *TADS*.

6.2 Adding Program Modules to the GUI

A number of modifications need to be made to the GUI to add a program module. These consist of creating an input panel, adding the program module to the list in the main panel, creating subroutines to read and write the program module input files, and updating the global parameters.

6.2.1 Creating an Input Panel

The object oriented philosophy used in the GUI greatly simplifies the task of generating new a new input panel. The best procedure is to make a copy of a similar panel and modify it for the new application.

Since the blade-to-blade tasks are the most likely place for new modules to be added, the *RVCQ3D* input panel will be used as an example of how to create a new panel. The *RVCQ3D* input panel code is called rvcq3dgen.c in the gui subdirectory of the *TADS* system. In this file are many variables which start with the letters "rvc". A three letter abbreviation of the new application should be chosen to replace "rvc" in the variable and function names. This will insure that all new variable and function names are created, and that there will be no side effects between functions. There are many other variables in the code, but they are either global already, or are local to the *RVCQ3D* input panel code.

Action Buttons

For every panel there is a structure for the action buttons named BTNS_DATA. There is also a manifest constant (RVC_BTN_CNT in rvcq3dgen.c) which is defined to be the number of action buttons on the panel (6 for *RVCQ3D*). The BTNS_DATA structure defines the widget name and the placement of each action button. The specific form of this and all other data structures is found in the guilib subdirectory in a file called ltds.h. The actions of the buttons are defined in the function "rvc_inp_dec_pbCB". The BTNS_DATA structure and call back function generally do not require modification, except for changing the variable names as discussed above.

Input Panel Data Structures

There are two data structures which need to be tailored to the new module: GROUP_DATA and GROUP_PNTRS. These structures control the names, contents and behaviors of the individual parameter widgets on the input panel. The manifest constant "RVC_CNT" sets the number of input groups to be displayed on the input panel. The groups are arbitrary divisions of the input parameters, which are grouped and titled on the input panel. For *RVCQ3D* the groups correspond to the members of each input namelist. If the FORTRAN namelist style input is used in the program module, the input groups should be defined by the namelist members. Each group can have as many as 30 parameters associated with it, as defined by the MAX_CELLS constant in the file ltds.h. The constant "RVC_CNT" is defined in the file constants.h. A new constant needs to be defined for the new panel in the form of "RVC_CNT" (use the three letter abbreviation chosen above).

For each input group there are two sets of parameters encased in curly braces. The first set of parameters describes the characteristics of the group: the group title, namelist name (if applicable), position, size and margins, the number of input variables in the group, and the number of columns to be used by the widgets on the input panel. The second set of parameters is repeated for each input variable. The first three parameters are the variable name and two widget id parameters. The widget id parameters are set internally by the GUI and the user should initialize them to 0. The fourth parameter is a Boolean variable which determines whether the widget is active (editable) or not. This parameter may be reset internally, but the specified value is used initially.

The fifth parameter determines the behavior of the widget for slice-dependent input panels. A value of 0 means that the widget is active or inactive regardless of whether the panel is in "All Slices" mode, or is set to a specific slice. A value of 0 effectively means that the fourth parameter controls the behavior of the widget (used for slice-independent data). A value of 1 means that the widget is active in "All Slices" mode and inactive for any individual slice. Conversely, a value of 2 means that the widget is inactive in "All Slices" mode and active for any individual slice.

The sixth, seventh and eighth parameters are values of the input variable. The sixth parameter is a pointer to the current value of the input variable. The seventh parameter is the default value of the input variable. The eighth

parameter is used internally to determine whether or not the value has been changed on the input panel. This parameter should be initialized to the default value.

The ninth parameter is the number of decimal places to be displayed in the input panel. The number of decimal places is also used when generating the input file for the program module. The tenth and eleventh variables are pointers to the minimum and maximum acceptable values for the input parameter.

The twelfth parameter specifies the type of data range checking to be performed. A value of 0 means no data checking. A value of 1 means check a range between the minimum and maximum. A value of 2 means the input value must be greater than or equal to the minimum value. A value of 3 means the input value must be less than or equal to the maximum value.

The thirteenth parameter specifies the type of widget to be displayed on the input panel. A value of 0 means that a text box will be displayed. A value of 1 indicates a pulldown menu, and a value of 2 specifies a toggle button.

The GROUP_PNTRS data structure has a record for each input variable divided into groups like the GROUP_DATA structure. The parameters in the GROUP_PNTRS structure are the pointed-to locations of the pointers in the GROUP_DATA structure. The three parameters are the current value, minimum and maximum for the input variable. The current value is a placeholder for a variable which is set internally, and should be initialized to 0. The minimum and maximum values should be set to the valid limits of the parameter whenever possible. In the event that the range is unknown, the values should be set to 0, and the data checking parameter in GROUP_DATA (twelfth) should be set to 0.

The reason for the GROUP_PNTRS structure is that it provides a convenient mechanism for creating and using the database files associated with the slice-dependent input panels. The contents of these databases are read and written directly from the GROUP_PNTRS structure. The whole GROUP_DATA structure is not part of the database because some parameters, such as the widget id, have different values for each execution of the *TADS* system. If these were part of the database, then the widget id numbers would be corrupted on restart. Other parameters are constant and need not be part of the written database. The GROUP_PNTRS structure avoids unnecessary storage and corruption of internally generated values.

Implementing Callback Functions

Once the new panel has been created, variable names changed, and data structures specified appropriately, the next step is to add callback functions. Callback functions are the pieces of code which perform actions in response to various events. Examples of events are opening the input panel, quitting the input panel or pressing an action button. Without the callbacks, the input panel is not connected to the GUI or the program modules.

Most of the changes to the new input panel function code required to implement callbacks are accomplished by the variable name changing described above. The bulk of the effort is in writing the functions required by the callbacks. There is a function for reading data from an existing input file and recomputing special input parameters, and a function for writing new input files.

The file input function is called when the input panel is opened, and when the *TADS* system is initialized. The file input function obviously contains coding to read an input file for the program module. However, the values from an existing input file are not appropriate for some input parameters. In the case of the blade-to-blade flow analysis, the reference conditions, boundary conditions, and geometric information should be computed from values known in *TADS*, rather than used directly from an existing input file. Generally, if an input parameter can be computed, the computed value should be used rather than the read value. This eliminates the possibility of specifying conflicting data in the GUI. The computation of input parameters frequently requires reading other *TADS* files, and working with globally defined data (such as a grid size).

Frequently, the file input function is written in FORTRAN, while the GUI is written in C. C codes can call FORTRAN subroutines provided that two issues are resolved. First, all elements in FORTRAN argument lists are passed by reference, and not by value. Therefore, the C code must specify all arguments as pointers. For simplicity, current functions pass all arguments as float (real) values. If the actual argument is an integer, temporary variables are used inside the function, and assignments are made appropriately. It is not necessary to follow this strategy, but it simplifies the writing of the C statement to call the FORTRAN subroutine. Second, FORTRAN compilers use different naming conventions for modules, depending on the vendor. For example, the SGI compiler refers to subroutines by their name in lower

case post-pended with an underscore. The IBM compiler can be forced to to the same, with compiler options. Other vendors use different naming conventions, and that affects the way that the C code calls the FORTRAN routines. Some experimentation may be required before the various modules will link into an executable.

The file output routine contains coding to write an input file for the program module. If the program module uses namelist style input, the function "punch_namelist" can be used, following the model in rvcq3dgen.c. If not, then custom coding must be written and linked to the GUI. The above discussion about mixing C and FORTRAN applies here also.

Modifying the Main Panel

Changes must be made to the main panel source code main.c to add the program module to the appropriate component group. In the function "init_gui_input_panels" is a case block which determines which input panel is initialized for each component group. The new module should be added here under the appropriate case. Similar changes must be made to a case block in the function "dec_btnCB" which initializes the program module input data in the "Edit/Run" and "Run" modes. The function "runCB" contains a case block which initializes the input data and runs the appropriate program module. Again, the new module needs to be added, following the example of other modules. There will be multiple changes to this function because there are multiple events which cause the execution of a program module. Also, prototypes of the new functions need to be added to the header section of main.c.

6.2.2 Finishing the Installation

The *TADS* system must know where the executables can be found for each supported platform. The source code for the new program module should be placed in the modules subdirectory with the other modules. Also, symbolic links to the executables should be placed in the apl subdirectory. At present, executables are required for SGI R4000 and R8000 workstations, and IBM RS/6000 workstations.

This completes the addition of a new program module to an existing component group. Adding a new program module following an existing model can be accomplished in about a day by an experienced programmer.

6.3 Component Group Modifications

Adding a component group is a more complicated exercise, and may require new coding for which no model exists, depending on the function of the component. An example of a new component would be a blade shape generation code for the design system. The majority of the effort will be in modifying main.c to handle the new capability. If the new task fits in one place sequentially in the work flow, the changes will mostly involve expanding existing decision blocks. On the other hand, if the new module is callable in many places during the analysis sequence, then whole new decision structures will be required.

New interface routines may also be needed between the new component and existing components of the analysis. These routines should be placed with the program modules in the modules subdirectory. The common directory under modules is a valuable source of routines for reading and writing *TADS* files, and converting data between various coordinate systems.

6.4 Adding New Host Types for Remote Execution

Adding new host types is relatively straightforward. An example of this would be to add Cray computers to the list of supported execution platforms. This involves changing the configgen.c source code in the gui subdirectory. In the function "configuregen" is a case block which identifies the supported platforms ("Silicon Graphics", "IBM", etc.). The new host type should be added to this list, and the loop index should be increased to reflect the new choice. Also, the file config.h has an enumerated type "mach_types" which needs to be updated following the pattern of the case block modification. The maximum number of supported platforms is specified by the manifest constant "MAX_NO_MACHINES" in the file constants.h.

The program modules are executed via "system" function calls from the GUI. The "system" is used to invoke the UNIX shell script rsh_tds from the apl subdirectory. The shell script tests to see which machine type is required, and creates the appropriate execution statement. The test logic must be updated to show the new machine type. The machine types correspond to the enumerated type mentioned above. The script interprets the type of

input and output files required from the number of arguments received by the shell script. Some modification may be necessary to create the proper execution statement. The script then executes the statement on the local machine, or starts a remote shell to run on the specified host.

6.5 *Makemake*

Makemake is a UNIX shell script to create makefiles for *TADS* program modules. It is run in the source directory of a program module and creates a new makefile named Makefile.new.

Makemake offers many features for managing coupled codes. One difficulty in supporting multiple platforms is keeping the object files segregated in the source directory. *Makemake* applies different suffixes to the object files from each compiler to avoid problems with linking dissimilar objects.

Also, targets are provided in the makefiles for checking source codes into and out of the Revision Control System. RCS allows the evolution of a code to be tracked by managing different releases of each source code in a special subdirectory. Any previous release of a subroutine can be recalled so that older capabilities are always recoverable. A release numbering scheme enables incremental improvements to be distinguished from major new releases. All program modules written for *TADS* use RCS.

A dependencies section is generated in the makefiles so that if a file is updated, all objects dependent on that file will automatically be re-compiled when the next executable is made. Dependencies are identified in either the C or FORTRAN syntax. A reliable dependencies list greatly reduces the time (or uncertainty) involved with creating new executables.

The ability to create archive libraries of subroutines is also incorporated into makefiles created by *makemake*. These libraries are identified with the associated revision level of the code so that executables can be created easily for older releases.

Program modules written for the *TADS* system share include files between modules. In each source directory, a symbolic link is made to the include files in the common directory. To avoid entering the include files into multiple RCS directories, the symbolic links should be removed before running *Makemake*. A UNIX shell script rmlinks accomplishes this job. Similarly, the script linkinc restores the links.

Makemake requires a makefile template. The resulting makefile is effectively an edited version of the template. To create a different style of makefile, the user simply supplies a suitable template. *Makemake* and the associated tools and templates are found in the TOOLS subdirectory.

Chapter 7

Verification

The coupled throughflow and blade-to-blade analyses have been successfully applied to four cases which will be reviewed here: NASA Rotor 67, the fifth stator from an 8-stage core compressor (AST), the first rotor from the Purdue Low Speed Turbine Rig, and the vane from a turbine stage tested in a shock tunnel (VBI stage). These four cases represent vanes and blades from both compressors and turbines, and span the spectrum of turbomachinery flow conditions from incompressible to transonic. The purpose of these studies is to verify the operation of the *TADS* system. Euler results from the *ADPAC* throughflow solver are compared with the axisymmetric average of a full 3-D Euler *ADPAC* solution, demonstrating the performance of the body force and blockage implementation in the throughflow analysis. Euler solutions of individual blade-to-blade streamlines are compared with the corresponding results from the full 3-D Euler analysis, to verify the sharing of boundary condition information between the throughflow and blade-to-blade analyses. The sum of the mass flows from the blade-to-blade analyses are compared with the mass flow from the throughflow calculation and with the full 3-D calculation to verify the internal consistency of the coupled system.

7.1 NASA Rotor 67

NASA Rotor 67 is a transonic fan which has been studied extensively both experimentally and analytically. The highly loaded rotor was tested by Pierzga and Wood at NASA Lewis in 1985, Ref. [13]. Analytical researchers have had

difficulty matching the data from the experiments, leading to the conclusion that the reported "hot shape" of the airfoil was inadequate. Since then, a new "hot shape" for the rotor was generated from the cold coordinates using finite element methods at Allison, and subsequent analytical results were significantly better. This redefined "hot shape" was used in the current work.

Contour plots of absolute total pressure are shown for the throughflow and 3-D analyses in the section "Verification of Body Force Formulation." The 3-D and throughflow analyses have been rerun using finer grids, and those results are presented here.

The analysis was run for three full iterations: that is, the throughflow analysis and blade-to-blade solvers were run three times each, updating the meridional and blade-to-blade stream surfaces each iteration. Figure 7.1 shows the relative Mach number contours from the throughflow analysis at each iteration. As seen, the shock spreads down the span of the airfoil and a radial gradient forms downstream of the airfoil as iterations progress. The changes are smaller between the second and third iteration, indicating that the total system is converging. The large change between the first and second iteration is largely due to changes in the mean stream surface near the leading edge. The mass flow varies with iteration, and is closest to the mass flow from the full 3-D Euler solution after the third iteration. The pressure ratio drops and the efficiency rises with each iteration. The magnitude of the changes decreases between iterations.

Figure 7.2 shows the comparison of the relative Mach number contours between the third iteration through *TADS* and the axisymmetric average of the full 3-D Euler solution. The general trends are the same between solutions, but the details are different. The contours upstream and downstream of the rotor are in good agreement. In the bladed region, the differences are much larger. To some extent, these differences are expected because of the different solution procedures used. In the full 3-D solution, there is a shock structure, but the axisymmetric average de-emphasizes the shocks because the shocks are not aligned with the circumferential direction. On the other hand, the throughflow analysis is incapable of producing an oblique shock because the flow is assumed axisymmetric. This explains why the strong shock is present in the throughflow solution and not in the axisymmetric average. The presence of the shock accounts for most of the difference between the two solutions.

The throughflow solution is used primarily to provide the meridional

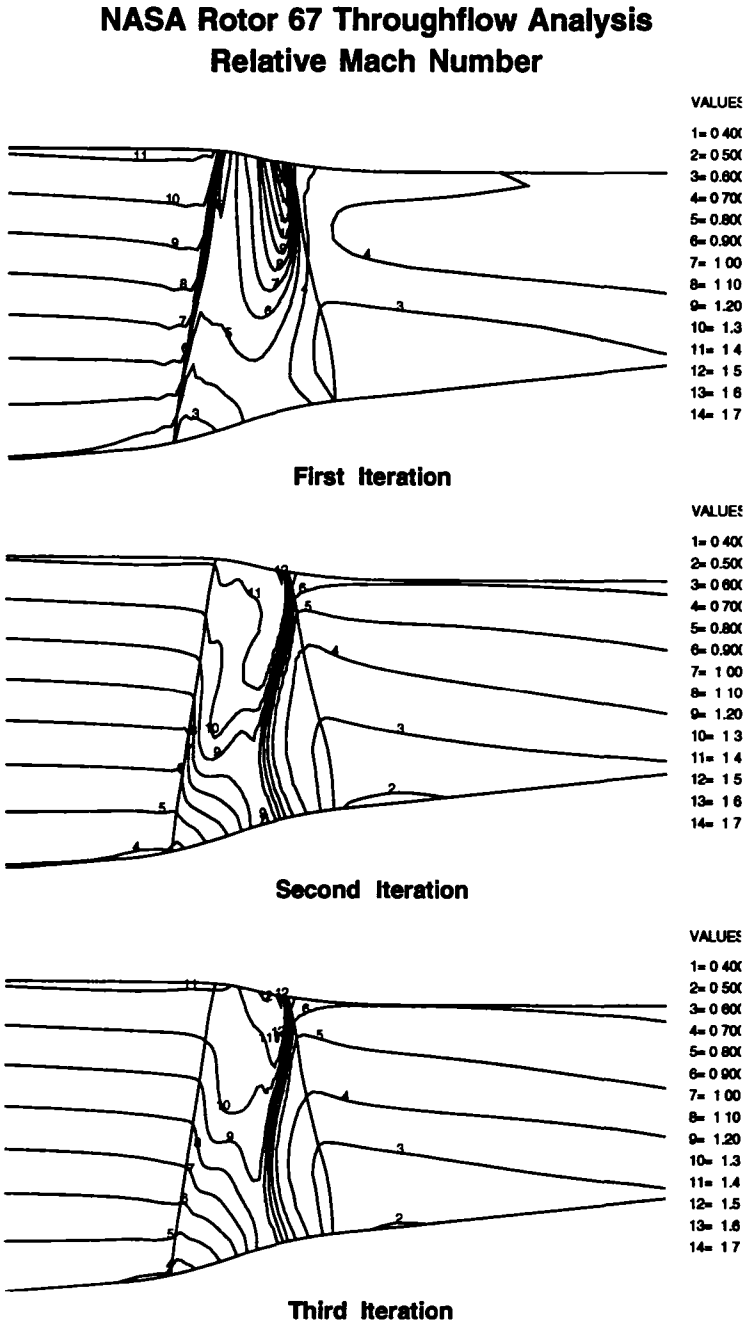


Figure 7.1: The relative Mach number contours show how the throughflow solution responded to changes in the mean stream surface between iterations.

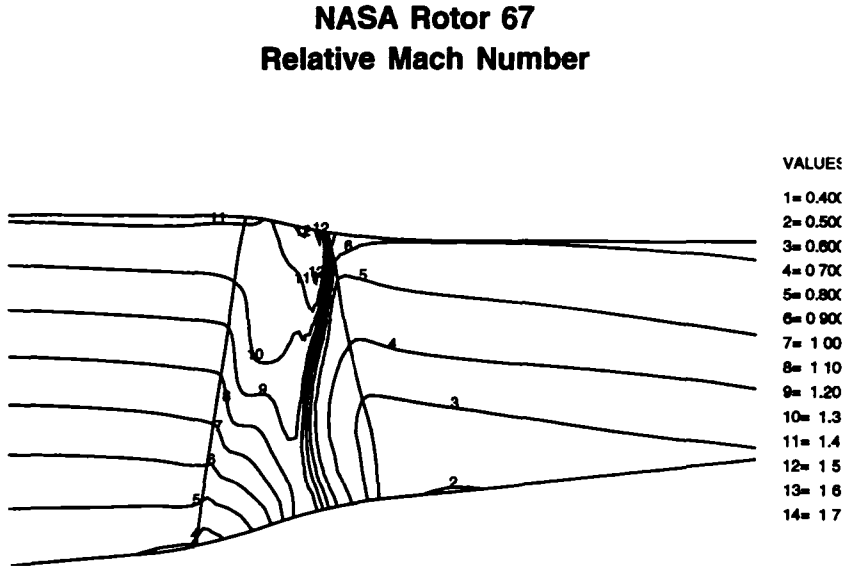


Figure 7.2: The relative Mach number contours from the third iteration and the axisymmetric average of the full 3-D solution are in good agreement outside of the bladed region. The presence of the normal shock in the throughflow analysis accounts for differences in the blade row.

streamline shapes and boundary conditions for the blade-to-blade analysis. If the upstream and downstream solutions are in good agreement, and the streamlines from the throughflow solution are close to the streamlines from the 3-D solution, then the differences between the solutions are not terribly important to the overall analysis. However, the shape and distribution of the streamlines have a first order effect on the blade-to-blade solutions. The rate of change of radius (dr/dx) and the rate of change of stream tube height (db/dx) appear in the source terms in the quasi-3D analysis. Small irregularities in the streamline shape or the stream tube height can cause large differences in the blade-to-blade solutions.

Figure 7.3 shows the meridional streamlines computed three ways: from the axisymmetric average of the full 3-D Euler analysis, from the third iteration of the coupled throughflow and blade-to-blade system, and from purely geometric considerations, saying that flow is directly proportional to area. As seen, the streamlines from the *TADS* solution have nearly the same shape as streamlines from the axisymmetric average. The radial locations of the streamlines are slightly different, indicating that there is more flow near the tip in the full 3-D Euler solution. This relates to the differences in the shock structure between the two solutions.

A second flow feature also affects the distribution of the streamlines in the meridional plane. In the blade-to-blade plane, there is a flow separation at the hub region of the rotor, Figure 7.4. The extent of the separation is influenced by two factors. First, the radial distribution of the streamlines sets the stream tube height in the blade-to-blade flow, which in turn, influences the diffusion near the trailing edge. Second, all of the results presented in this report are solutions of the Euler equations. Since the flow is inviscid, the separation seen in the solutions is largely a function of the artificial dissipation in the various codes. The artificial dissipation scheme in *RVCQ3D* produces more losses than the scheme in *ADPAC*. It turns out that the *RVCQ3D* solution is quite similar to the hub section of a full 3-D Navier-Stokes solution, because of the artificial dissipation in *RVCQ3D*. The grids used in the blade-to-blade analysis are clustered near the airfoil surface, which exacerbates the problems associated with artificial dissipation in *RVCQ3D*. However, less refined meshes resulted in poor solution quality near the airfoil surface due to lack of resolution.

Figure 7.5 shows the comparison of the midspan sections from the blade-to-blade analysis and the full 3-D Euler solution. The agreement between

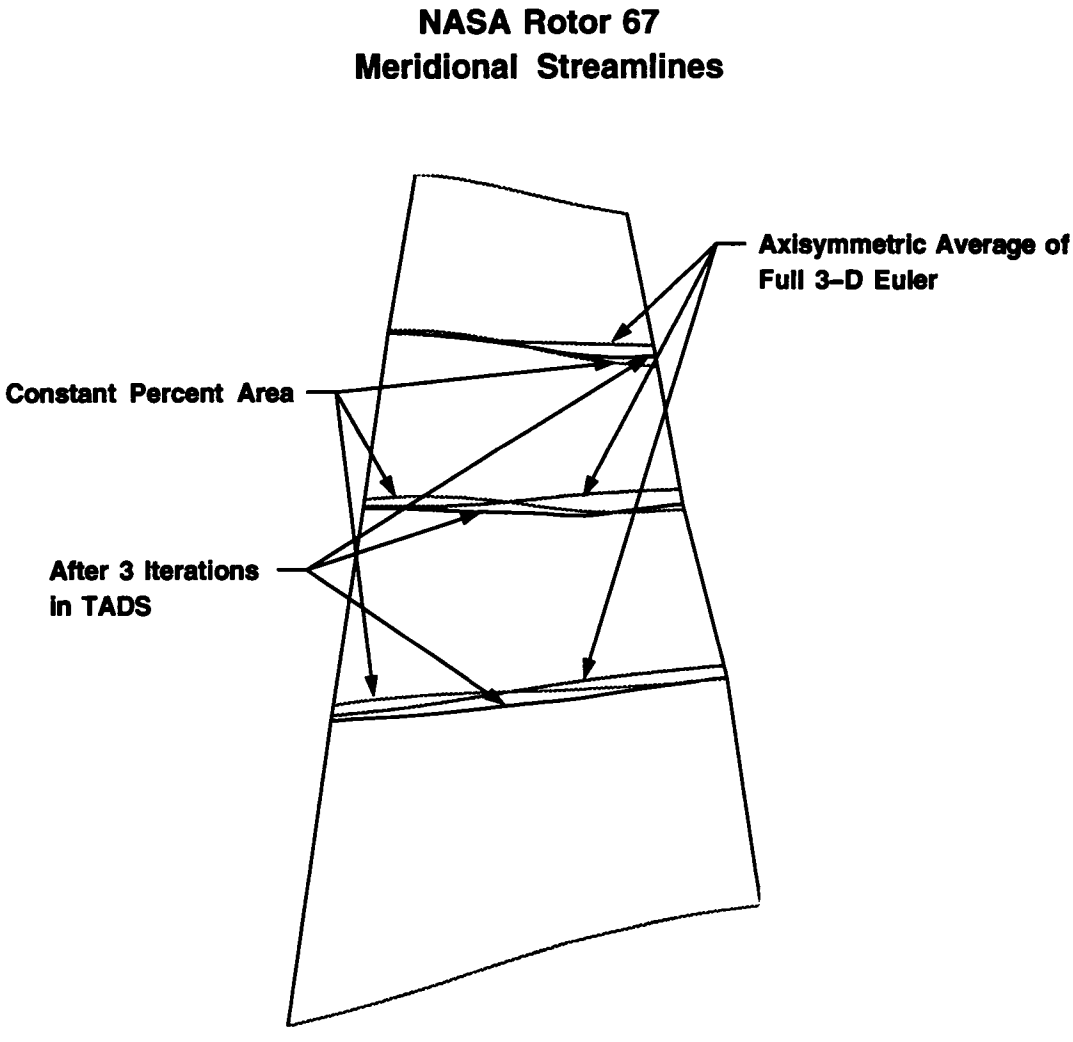


Figure 7.3: The meridional streamlines from *TADS* differ slightly from the full 3-D Euler streamlines because of differences in the shock structure between the two solutions.

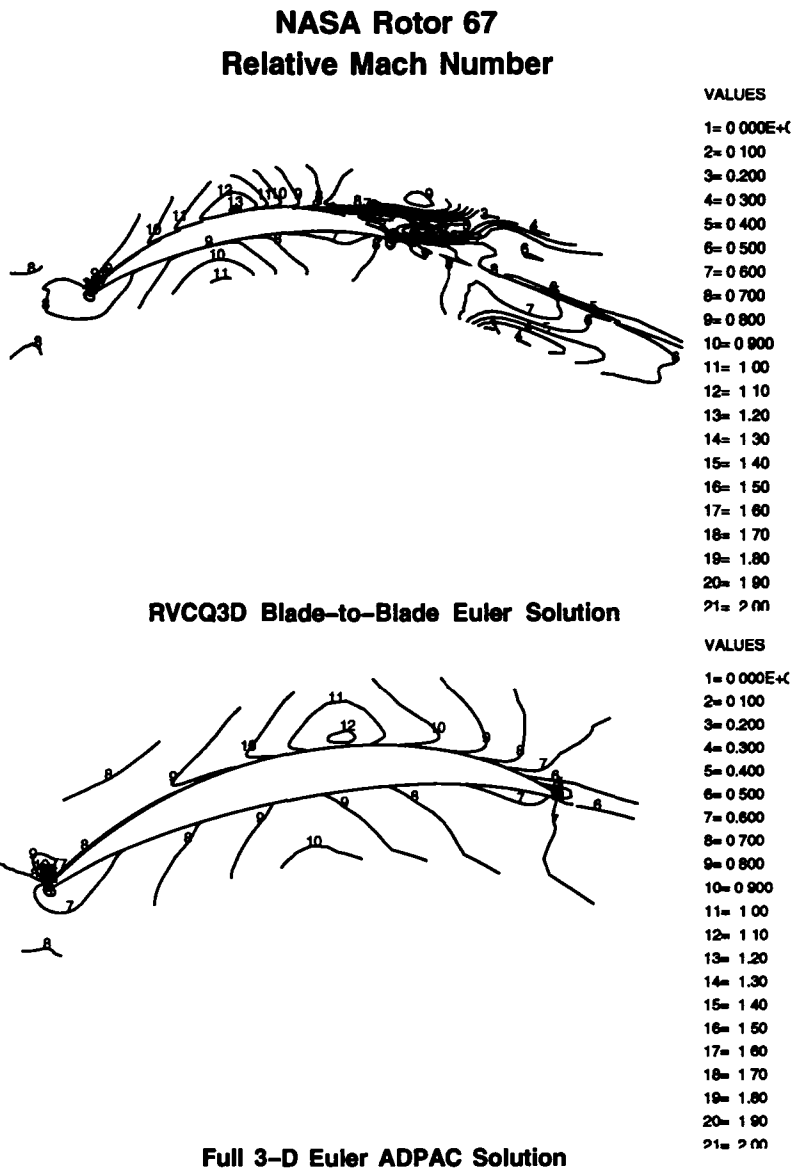


Figure 7.4: The relative Mach number contours at the hub section are similar, but significant differences arise because of the separation at the trailing edge in the *RVCQ3D* solution.

these solutions is not particularly good, for many of the reasons already discussed. The shape of the midspan streamline is different between the throughflow analysis and the full 3-D Euler analysis, Figure 7.3. In transonic flow, small differences in flow area can have a dramatic effect on the location and strength of shock waves. In fact, in the first iteration through *TADS*, it was necessary to use the streamline definition based purely on geometry in order to get the blade-to-blade analysis to converge on some streamlines. The mean blade-to-blade stream surface was based on the mean camber line and Carter's rule in the first iteration, because no blade-to-blade solution was available at that point. This stream surface was not correct, and resulted in inaccurate positions of the meridional streamlines found from the throughflow solution. The blade-to-blade analysis was not able to find a stable solution along some of these meridional streamlines.

Figure 7.6 shows the comparison of the tip sections from the blade-to-blade analysis and the full 3-D Euler solution. These solutions are in rather good agreement both qualitatively and quantitatively. Again, the larger wake in the *RVCQ3D* solution is the result of the higher dissipation near the blade surface resulting from the damping scheme in *RVCQ3D*. The tip solutions are less influenced by the streamline definition from the throughflow analysis because only the blockage is different between the solutions. The location of the hub and tip streamlines are fixed to the flow path definition. In light of this, it is expected that the hub and tip solutions would be in better agreement with the full 3-D solution than the interior streamlines.

Generally, the *TADS* solution of NASA Rotor 67 shows that the coupling of the program modules within the *TADS* system is correct. Boundary condition information is properly passed between the various codes, and the conversions between the non-dimensionalization schemes used in the codes are correct. Table 7.1 shows the comparison between successive iterations through *TADS* and the *ADPAC* 3-D Euler solution for Rotor 67. The agreement between the overall performance quantities in *TADS* and the 3-D Euler calculation is quite good. This is remarkable in that there are significant local differences between the various solutions, as discussed above.

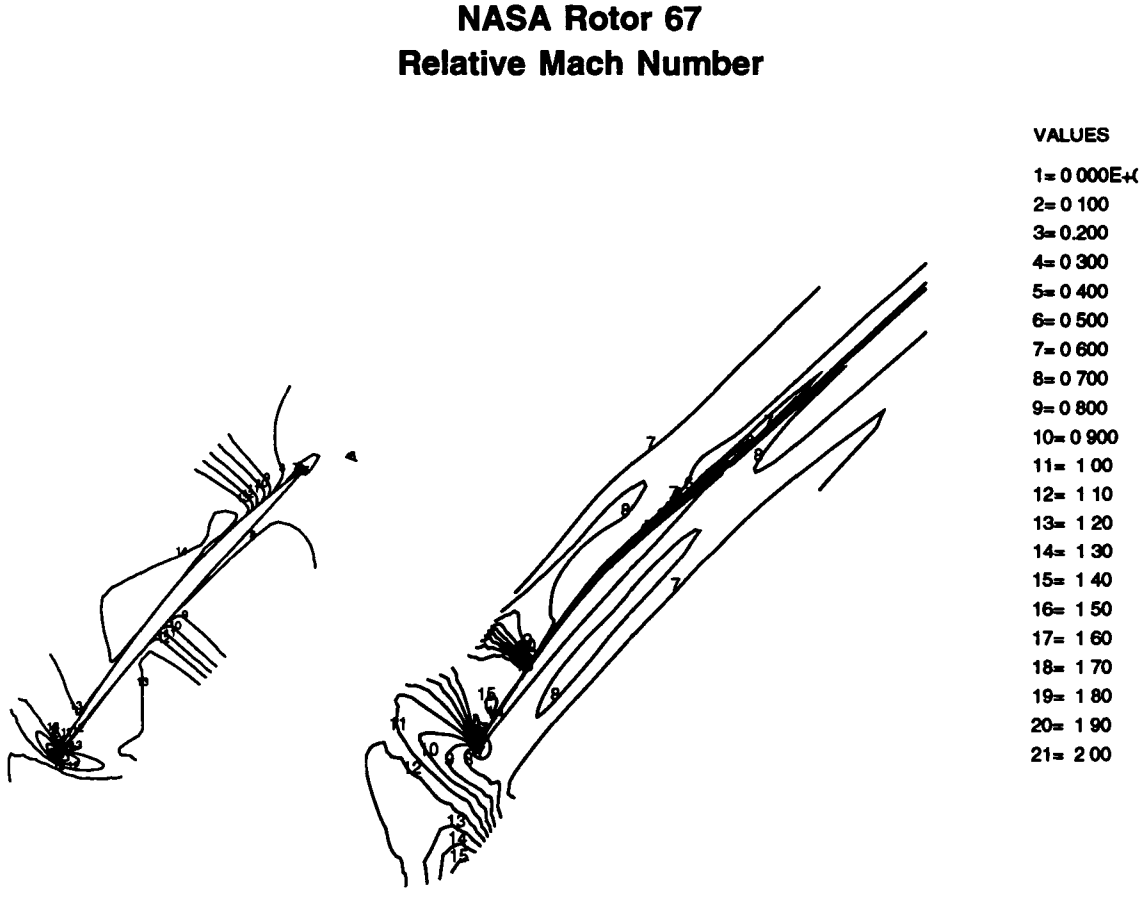


Figure 7.5: The relative Mach number contours at the midspan section are different because of differences in the meridional streamlines and stream tube heights between the solutions.

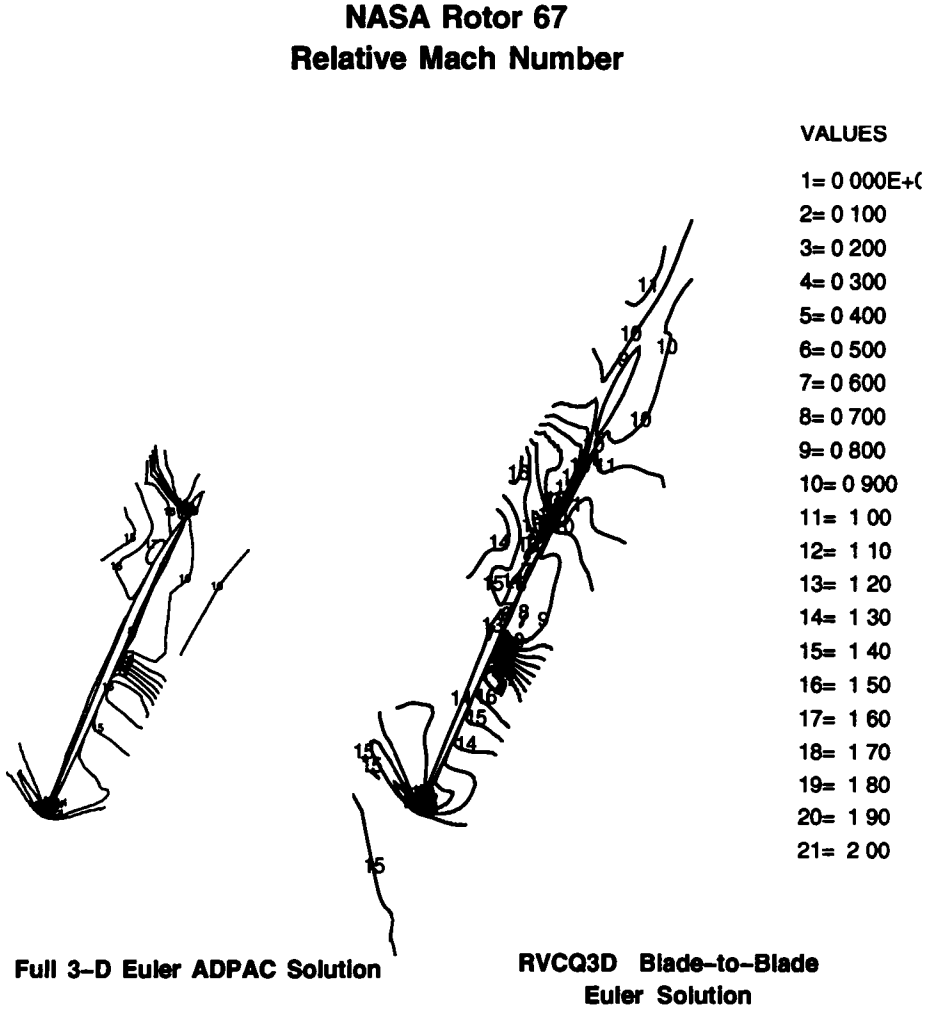


Figure 7.6: The relative Mach number contours at the tip section are in very good agreement.

Table 7.1: Comparison of *TADS* iterations with *ADPAC* 3-D Euler solution for NASA Rotor 67 shows good agreement.

	Flow (lbms/sec)	Pressure Ratio	Efficiency
TADS Iter. 1	77.57	1.781	87.8%
TADS Iter. 2	76.73	1.696	90.9%
TADS Iter. 3	77.83	1.692	92.2%
ADPAC 3-D Euler	78.52	1.695	92.6%

7.2 AST Compressor Stator 5

The fifth stator from the Allison candidate engine for the NASA Advanced Subsonic Technology (AST) program was also analyzed with the *TADS* system. The AST compressor is an eight stage high speed machine (19000 rpm) and is representative of current core compressor designs. The fifth stator was chosen because the flow is in the high subsonic range and the flowpath has significant contraction. The *TADS* analysis was performed for three full iterations through the throughflow and blade-to-blade analyses, and an *ADPAC* 3-D Euler calculation was run for comparison.

Figure 7.7 shows the Mach number contours from the throughflow analysis for each *TADS* iteration. As seen, there are differences between the first and second iteration but the second and third iteration are very similar. Unlike the Rotor 67 case, the first iteration, which uses the mean camber line and Carter's rule as the mean stream surface, is a good approximation to the converged solution.

Figure 7.8 shows the comparison between the converged throughflow analysis and the axisymmetric average of the 3-D Euler solution. The two solutions are in good agreement, both inside and outside the bladed region. There are no strong shock waves or large separated zones in either the blade-to-blade or throughflow solutions as there were in the Rotor 67 study. As discussed above, strong shocks tend to be misrepresented by the axisymmetric assumption in the throughflow analysis. Also, the losses caused by large flow separations are not modeled in the current analysis. The fact that losses computed in the blade-to-blade analysis are not communicated to the throughflow analysis leads to inconsistencies between the two analyses. In the absence of these factors, the coupled system performs well.

Figure 7.9 shows the meridional streamlines computed from the *TADS*

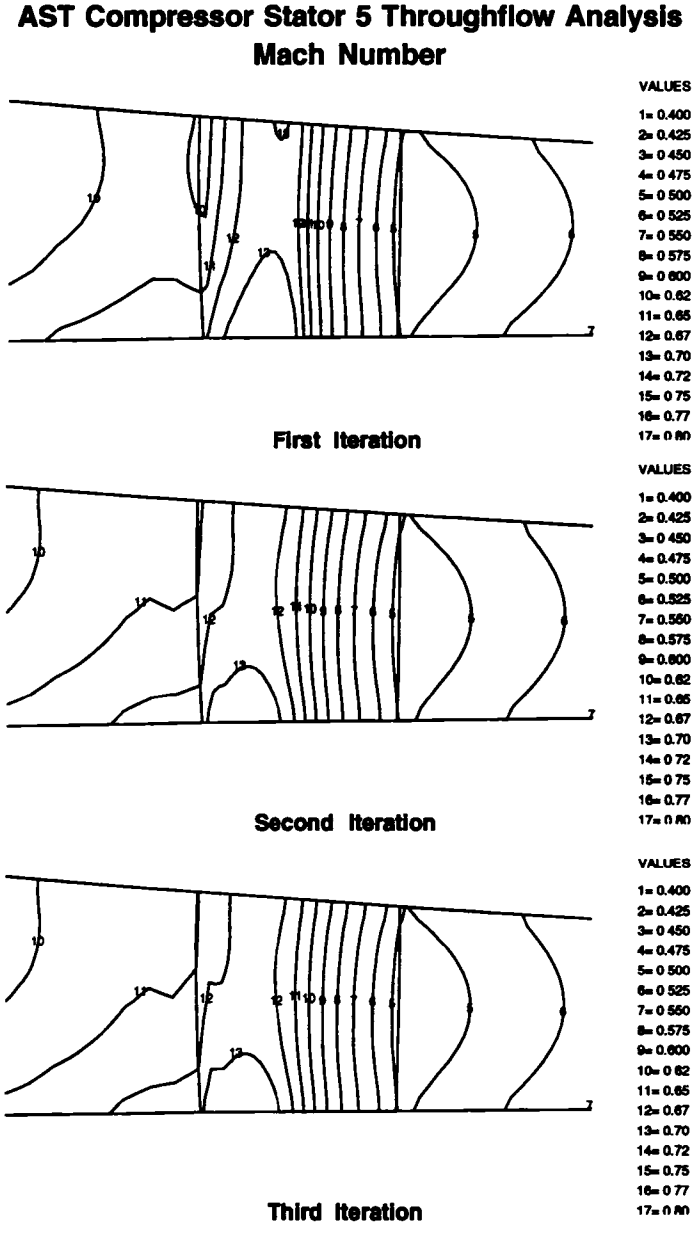


Figure 7.7: The Mach number contours show little difference between iterations, indicating that the initial stream surface (the mean camber line plus Carter's rule) is a good approximation to the converged solution.

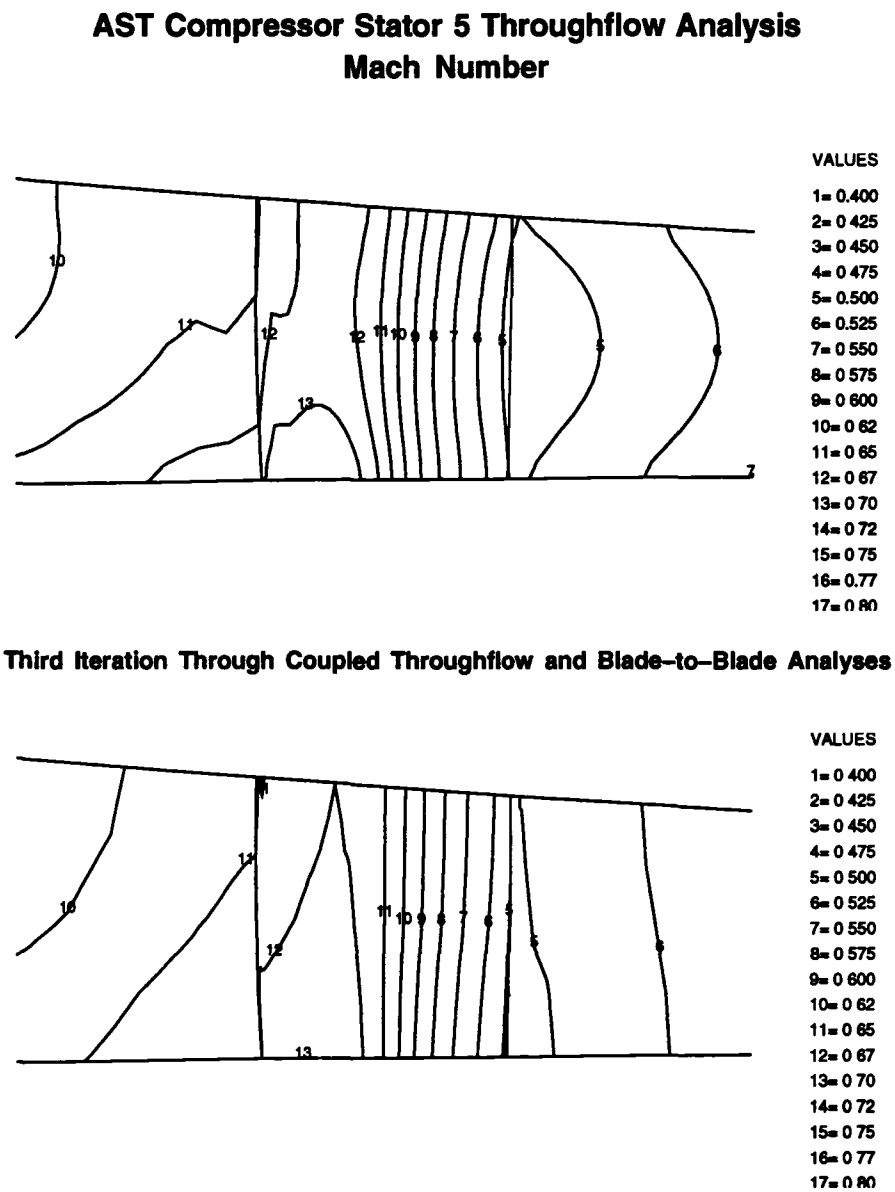


Figure 7.8: The Mach number contours from the converged *TADS* analysis and the axisymmetric average of the full 3-D solution are in good agreement both inside and outside the bladed region.

Table 7.2: The *TADS* iterations show good convergence, and reasonable agreement with the *ADPAC* 3-D Euler solution for the AST Compressor Stator 5.

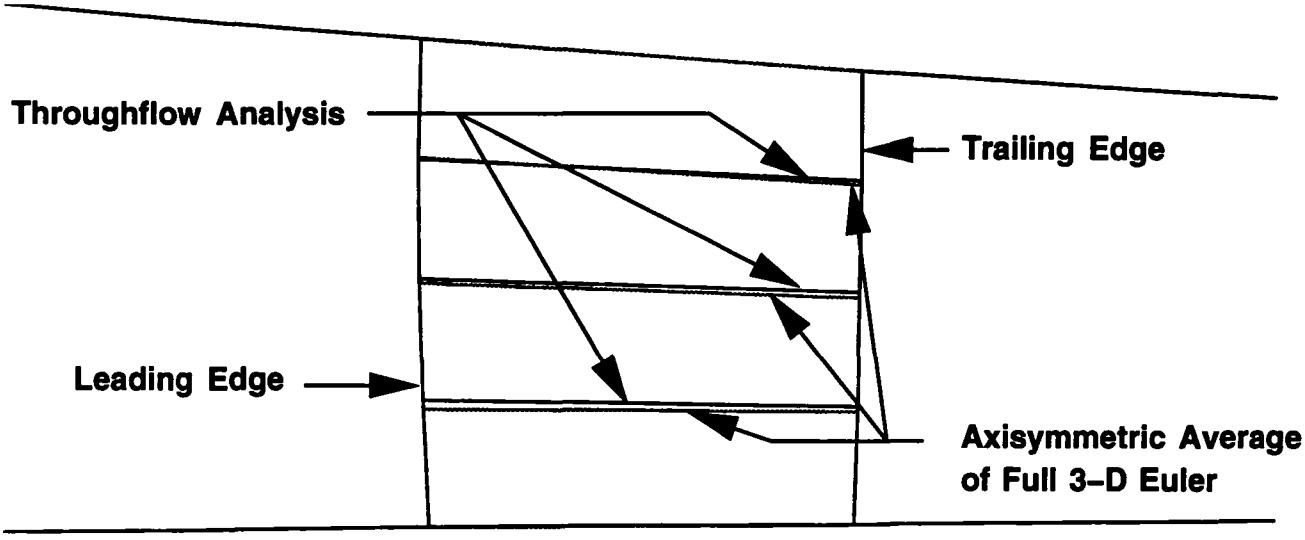
	Flow (lbms/sec)
TADS Iter. 1	68.42
TADS Iter. 2	69.19
TADS Iter. 3	69.18
ADPAC 3-D Euler	67.58

analysis and from the axisymmetric average of the 3-D Euler calculation. The *TADS* streamlines are from the first iteration, which is essentially the same as the converged solution. As seen, the streamlines are essentially parallel lines, and there is good agreement between the two analyses. Generally, the streamlines from the *TADS* analysis are at slightly higher radii than the streamlines from the 3-D solution. These differences have only a minor effect on the blade-to-blade solutions. The source terms in the quasi-3D analysis contain derivatives of the streamtube height and streamline shape. Because the throughflow streamlines are essentially parallel to the axisymmetric averaged streamlines, the derivatives are the roughly equal. Thus a quasi-3D analysis run with either streamline definition will produce the same result.

A comparison of the hub, midspan and tip Mach number contours between the two analyses are presented in Figures 7.10, 7.11 and 7.12, respectively. In all three cases, there is excellent agreement between the blade-to-blade analysis (*RVCQ3D* and the *ADPAC* 3-D Euler analysis. The minor differences in the streamline shapes are the cause of the small differences in Mach number levels between the solutions. In the hub and tip sections, the Mach numbers from the *TADS* analysis are slightly higher than from the 3-D solution.

Table 7.2 shows the mass flows after each iteration through *TADS* and from the *ADPAC* 3-D Euler solution for the AST fifth stator. The consistency between iterations indicates that the *TADS* analysis has converged, and that the first iteration is a good approximation to the converged solution. The mass flow from the converged *TADS* solution is within 2.5% of the full 3-D analysis, which is consistent with the blade-to-blade comparisons presented above. A small adjustment to the exit static pressure would eliminate this difference.

AST Compressor Stator 5
Meridional Streamlines



Meridional streamlines are computed two ways:

1. **Streamlines are computed from throughflow solution which used the mean camber line plus Carter's deviation angle rule as the mean stream surface.**

2. **Streamlines are computed from the axisymmetric average of a full 3–D Euler solution.**

Figure 7.9: The meridional streamlines between the two analyses are in good agreement.

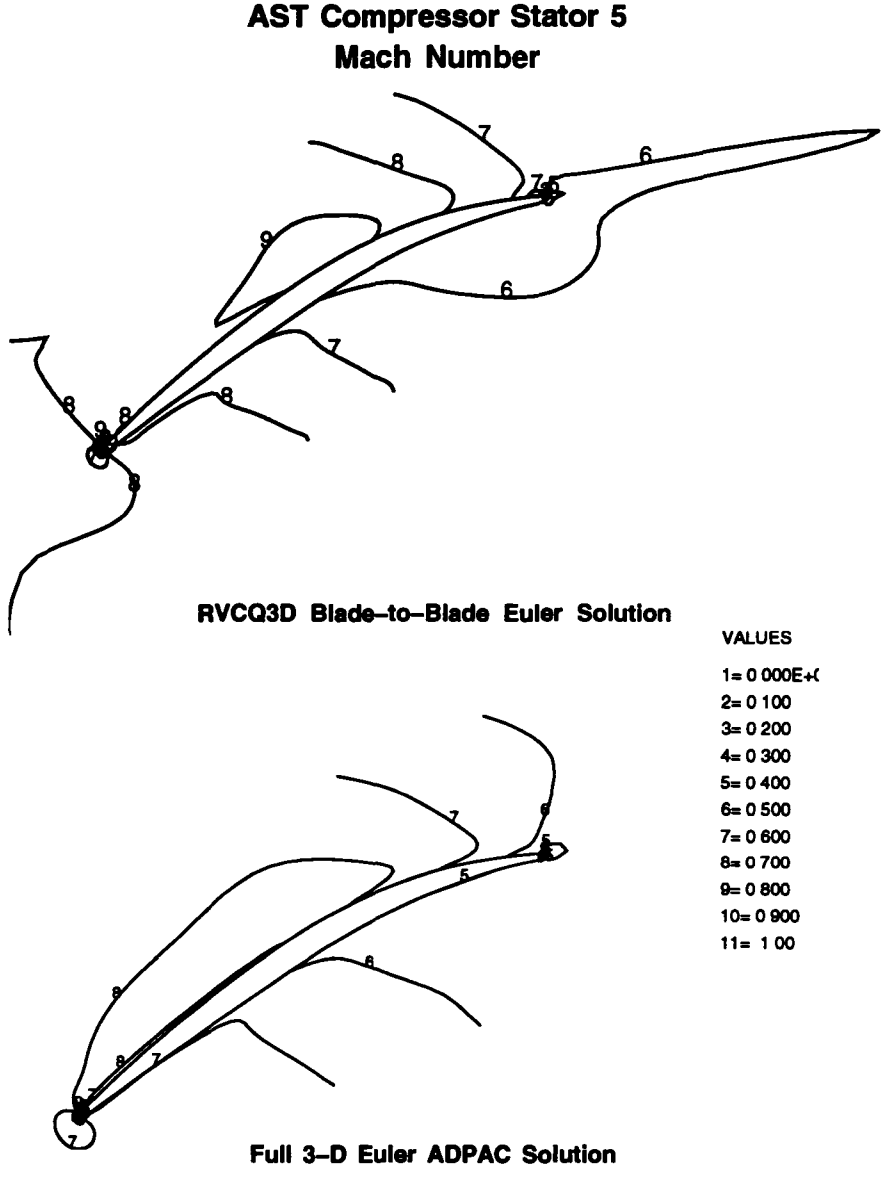


Figure 7.10: The Mach number contours at the hub section are in good agreement. The quasi-3D solution is at a slightly higher flow rate than the 3-D Euler section.

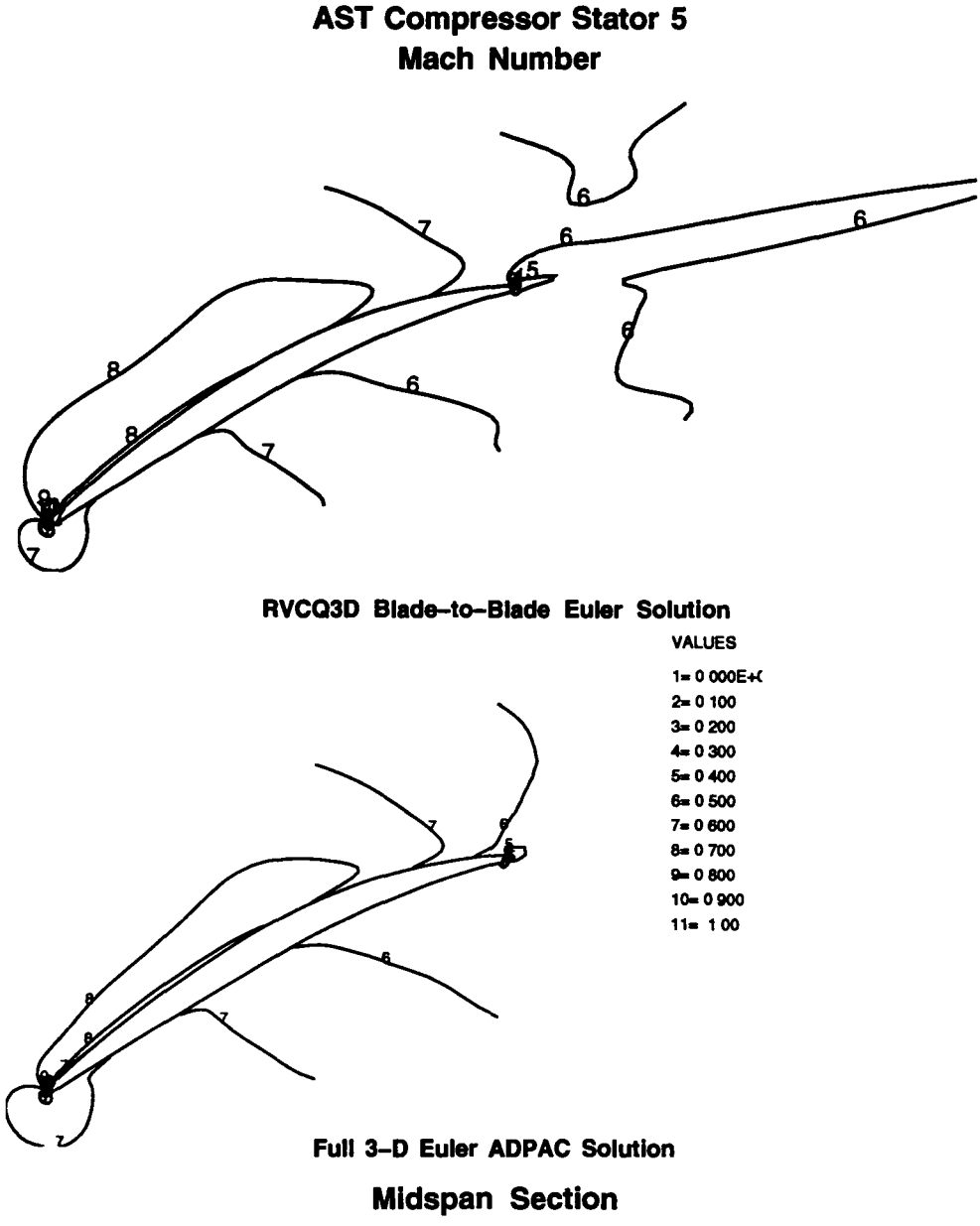


Figure 7.11: The Mach number contours at the midspan section are in excellent agreement.

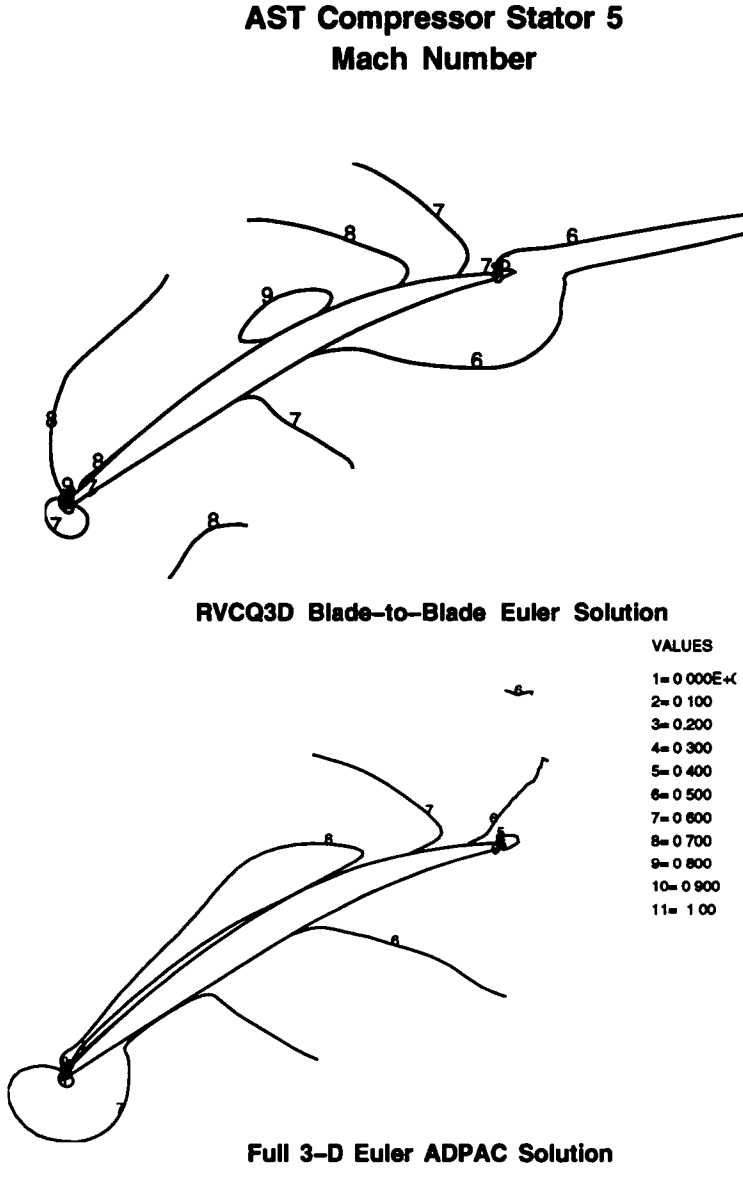


Figure 7.12: The Mach number contours at the tip section are in good agreement. The quasi-3D solution is at a slightly higher flow rate than the 3-D Euler section.

Table 7.3: The *TADS* iterations show good convergence for the first rotor of the Purdue Low Speed Turbine Rig.

	Flow (lbms/sec)	Pressure Ratio
TADS Iter. 1	5.447	.9361
TADS Iter. 2	6.657	.9326
TADS Iter. 3	6.765	.9329
TADS Iter. 4	6.774	.9329

The AST fifth stator calculations show that the *TADS* analysis is capable of accurately predicting the flow through a modern compressor stator. Carter's deviation angle rule performs well in the absence of shock waves and separated zones, yielding effectively the converged solution. In cases such as this, it is appropriate to run the *TADS* system for only one iteration.

7.3 Purdue Low Speed Turbine Rotor

The first rotor from the Purdue Low Speed Turbine Rig was chosen as a test case because of the high camber of the airfoil. The flow is basically incompressible, with a peak Mach number of around 0.3. The flowpath is annular, and the wheel spins at 2500 rpm.

The meridional Mach number contours from the throughflow analysis are shown for each iteration of the *TADS* system in Figure 7.13. The mean camber line was used as the initial mean stream surface because Carter's deviation angle rule is not applicable to turbine airfoils. As seen, the *TADS* system converges on the third iteration. Judging from the downstream Mach number distribution, the second iteration would be an acceptable stopping point for normal design work. Table 7.3 shows the mass flows and pressure ratios from each iteration.

Figure 7.14 shows the meridional streamlines after the first and second iterations through the *TADS* system. The third and fourth iteration are essentially replicas of the second iteration, and are not shown. The streamlines from the first iteration sag at the trailing edge, probably due to the fact that the flow does not follow the mean camber line near the trailing edge. Near the hub, the flow is being turned too much, the mass flow will be too high at the hub compared to the tip. Most likely, this is the cause of the small dip

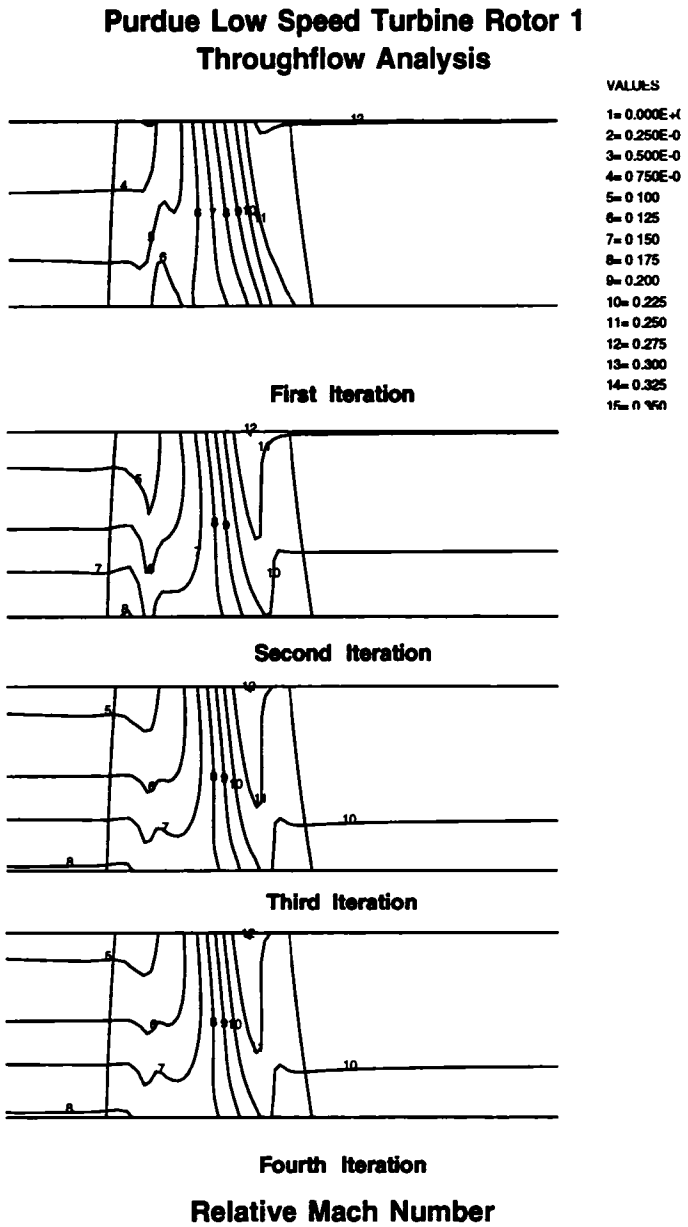


Figure 7.13: The relative Mach number contours from each iteration show that the *TADS* system is converged after three iterations.

in the streamlines near the trailing edge.

Figure 7.15 shows the blade-to-blade solutions for the hub, mean and tip sections of the Purdue Low Speed Turbine Rig first rotor. This turbine was designed to be two-dimensional: there is little radial migration of flow, and the loadings at each section are approximately the same. There is very little difference between the solutions for each section, indicating that the design intent was achieved.

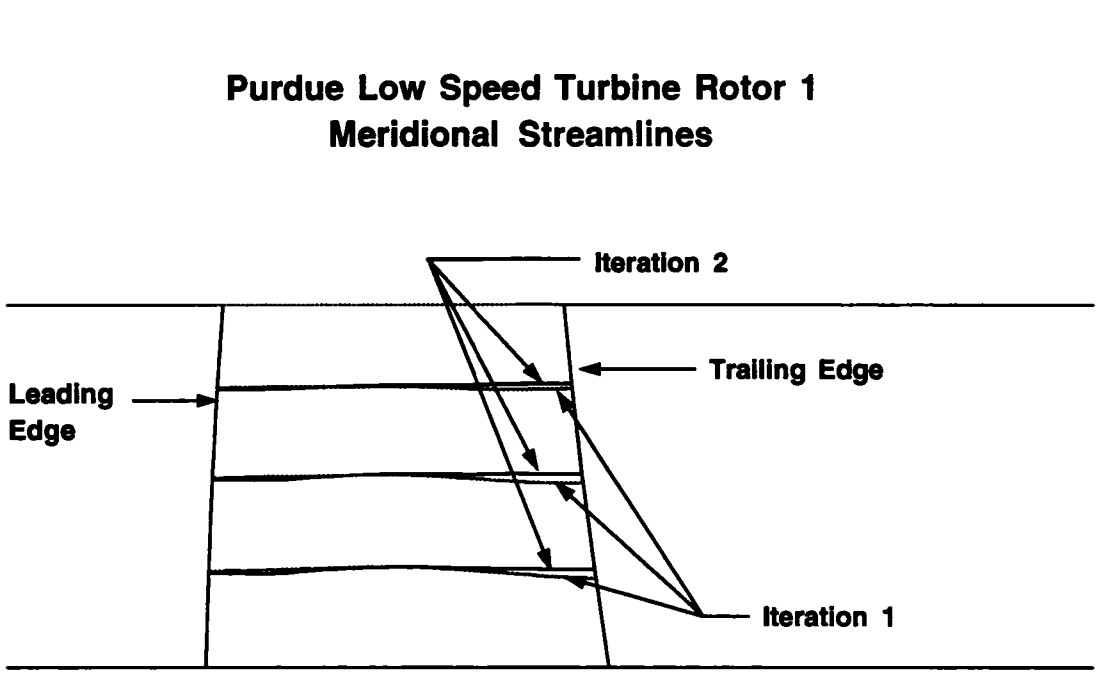
The *TADS* results show the expected behavior for the Purdue Low Speed Turbine Rig. This case has much greater blockage than the compressor cases presented above. The success of the analysis indicates that the blockage terms are performing as designed in the throughflow analysis.

7.4 VBI Turbine Vane

The fourth test case selected to verify the operation of the *TADS* system is the Vane-Blade Interaction (VBI) turbine vane. The VBI turbine is a single stage transonic turbine, which spins at 11,400 rpm in an annular flowpath. The steady and unsteady performance of the VBI turbine has been investigated at the Calspan Research Center by M. Dunn. Reference [6] documents the geometry, the experimental apparatus, and presents both experimental and analytical aerodynamic data for the VBI turbine. The VBI vane makes a good test case because of the significant airfoil thickness and the transonic flow.

The *TADS* system was run for four full iterations. Figure 7.16 shows the Mach number contours from the throughflow analysis after each iteration. The solution is converged in three iterations, but the first iteration is a reasonable approximation to the converged solution. The meridional streamlines found from the throughflow analysis after the first and fourth iterations are shown in Figure 7.17. The only difference in the streamlines between the first and fourth iterations is near the trailing edge. In turbine airfoils, however, the trailing edge is the critical area because the throats are typically set at the trailing edge. Changes in the stream tube height at the trailing edge can have a significant effect on the Mach number levels seen in the blade-to-blade solutions. In this case, the differences in the midspan solutions between the first and fourth iterations are minimal, Figure 7.18.

Table 7.4 shows the mass flows after each iteration through *TADS* and



Meridional streamlines are computed two ways:

Iteration 1. Streamlines are computed from the throughflow solution, which used the mean camber line as the mean stream surface

Iteration 2. Streamlines are computed from the throughflow solution, which used the mean stream surface calculated from the blade-to-blade solutions in Iteration 1.

Figure 7.14: The meridional streamlines computed from the throughflow solution are constant after two iterations.

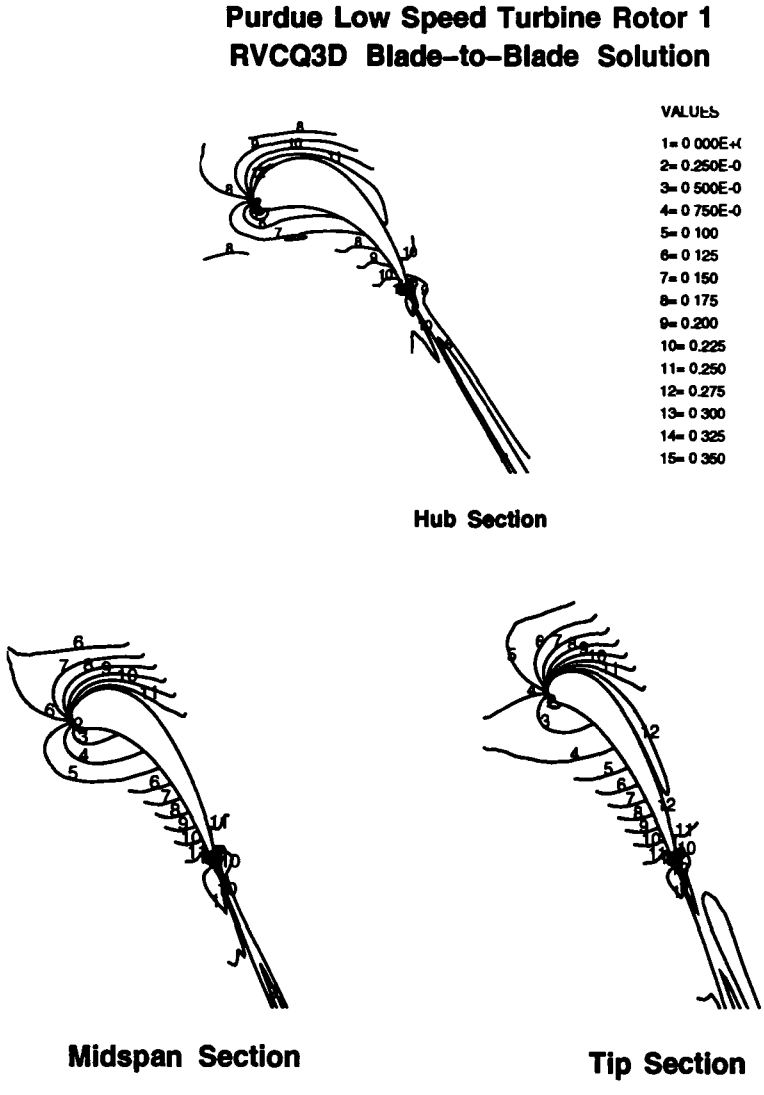


Figure 7.15: The relative Mach number contours from the blade-to-blade analysis show that the loading is essentially constant from hub to tip.

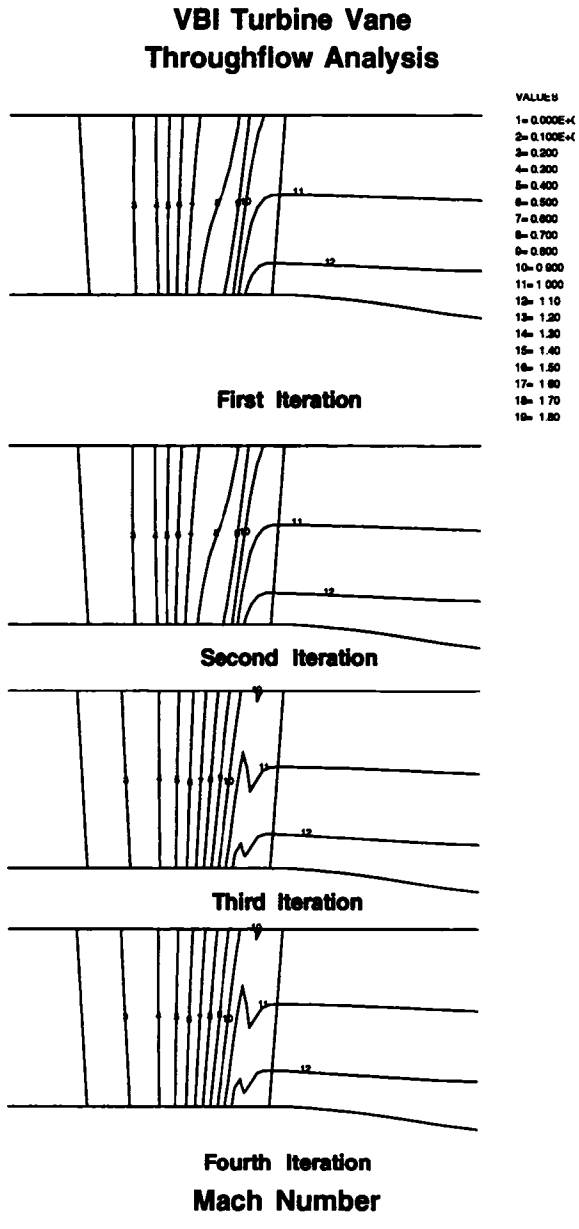
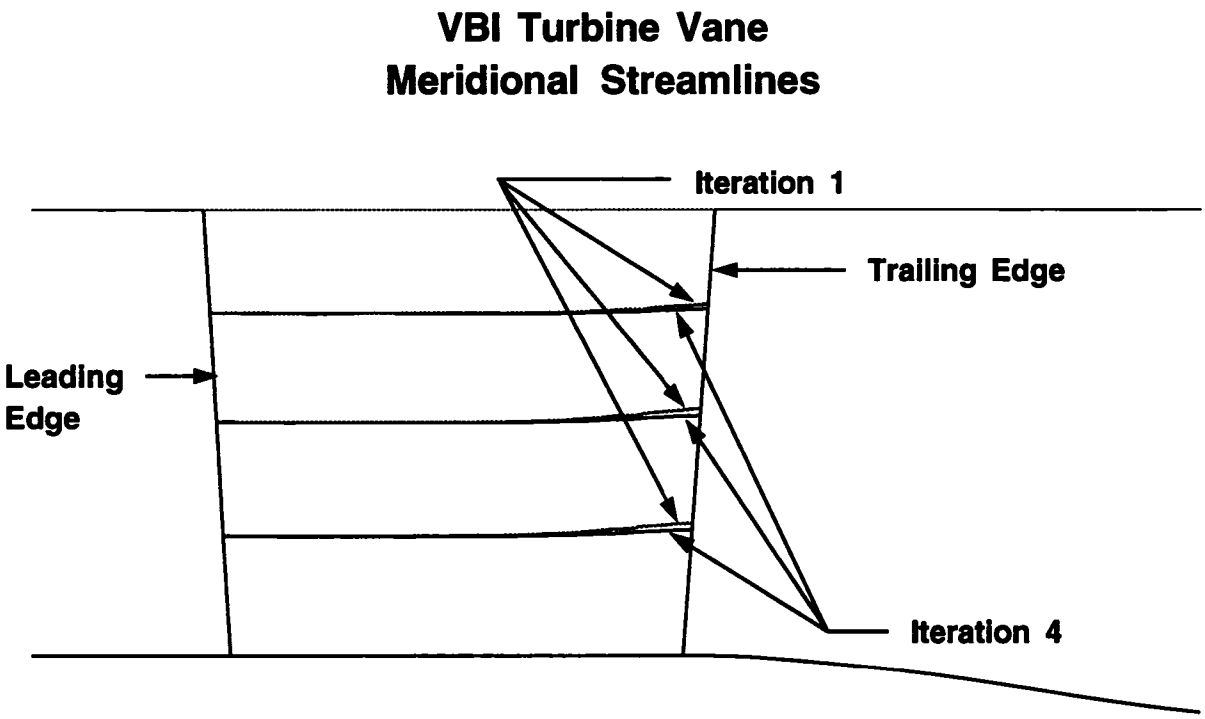


Figure 7.16: The meridional Mach number contours from each iteration of the throughflow analysis show that the *TADS* system is converged after three iterations.



Meridional streamlines are computed two ways:

Iteration 1. Streamlines are computed from the throughflow solution, which used the mean camber line as the mean stream surface

Iteration 4. Streamlines are computed from the throughflow solution, which used the mean stream surface calculated from the blade-to-blade solutions in Iteration 3.

Figure 7.17: The meridional streamlines from the first iteration are a good approximation to the final solution.

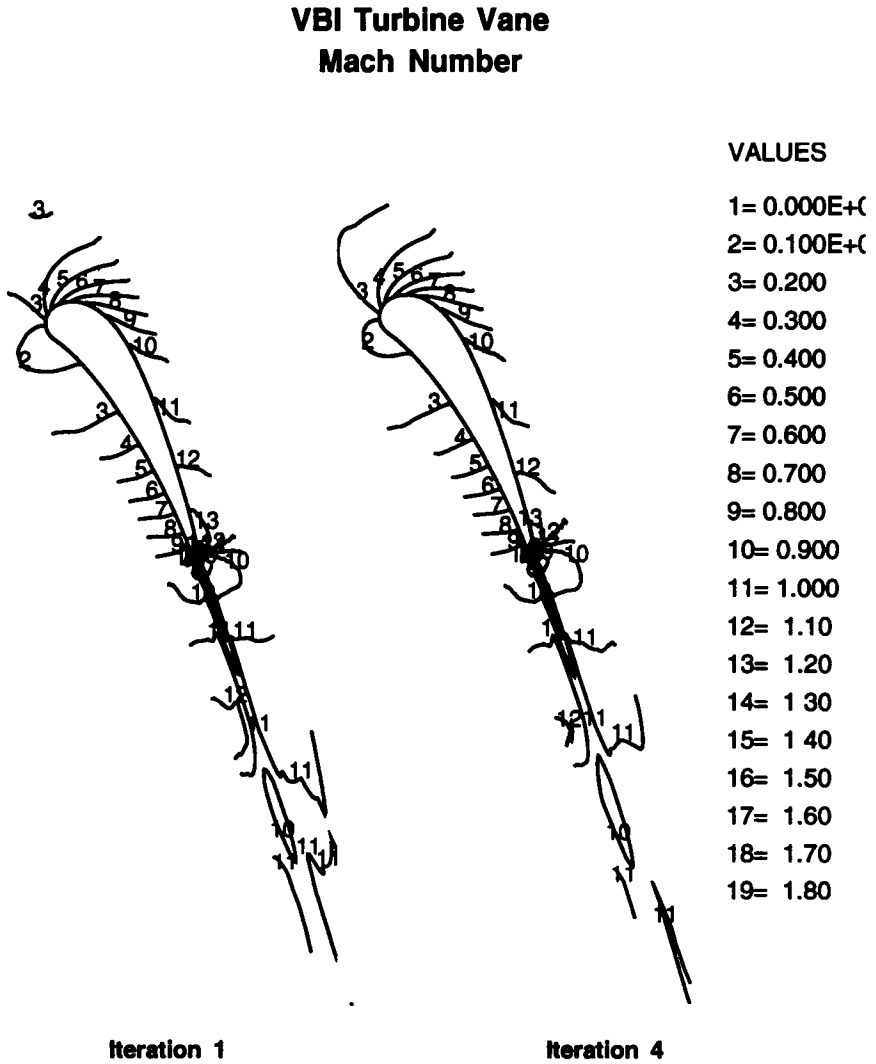


Figure 7.18: The midspan Mach number contours from the blade-to-blade analysis are effectively the same between the first and fourth iteration of the *TADS* system.

Table 7.4: The *TADS* iterations show good convergence, and reasonable agreement with the *ADPAC* 3-D Euler solution for the VBI Turbine Vane.

	Flow (lbms/sec)
TADS Iter. 1	22.89
TADS Iter. 2	22.04
TADS Iter. 3	24.78
TADS Iter. 4	24.80
ADPAC 3-D Euler	23.67

from the *ADPAC* 3-D Euler solution for the VBI vane. The mass flow reaches the converged value on the third iteration, consistent with the meridional Mach number contours presented in Figure 7.16. The converged mass flow is also in reasonable agreement with the 3-D Euler solution.

Figures 7.19, 7.20, and 7.21 show the comparison between the *RVCQ3D* blade-to-blade solutions and the *ADPAC* 3-D Euler prediction for the hub, midspan, and tip sections, respectively. As seen, the solutions are generally in good agreement, although there are minor differences in the position of some contours.

7.5 Summary

In each test case, the *TADS* system predictions are reasonable, and agree with 3-D Euler solutions at the same conditions. The good agreement demonstrates not only that the blade-to-blade solver is functioning properly, but that the system coupling is correct as well. The *TADS* system is a coupled system of quasi-3D solvers: the throughflow and blade-to-blade analyses both solve the governing equations in two dimensions, and rely on outside information to model the third dimension. The blade-to-blade analysis takes its boundary condition information from the throughflow analysis, and the throughflow analysis enforces flow tangency to the mean stream surface shape found by the blade-to-blade analysis in the bladed region. In order for the blade-to-blade results to agree with the 3-D results, the static pressure passed from the throughflow analysis must be correct. The static pressure in the throughflow solver is set by radial equilibrium at the grid exit. The radial equilibrium equation in the throughflow solver predicts the static pressure,

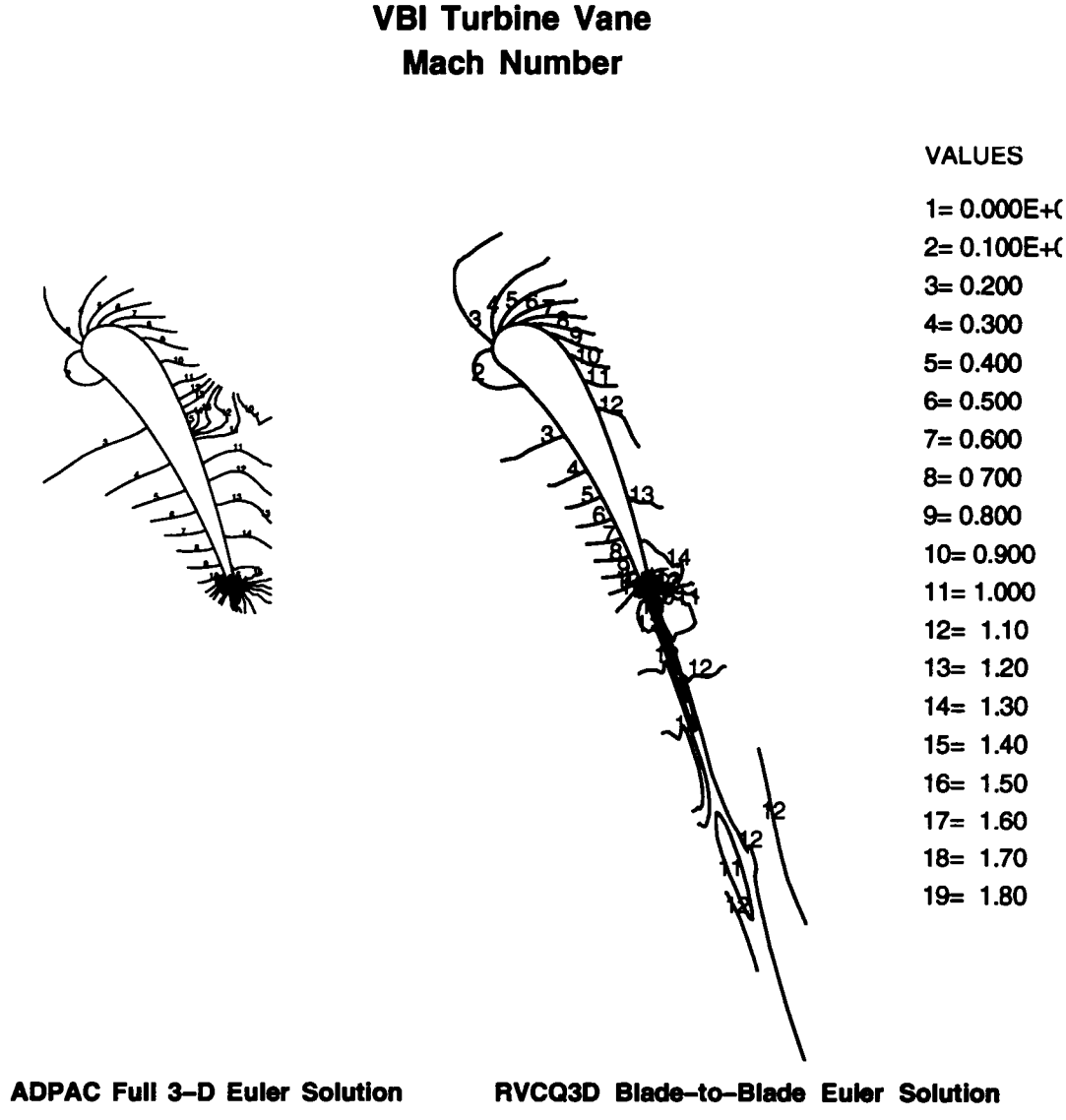


Figure 7.19: The Mach number contours from the hub section blade-to-blade analysis agree well with the 3-D Euler results.

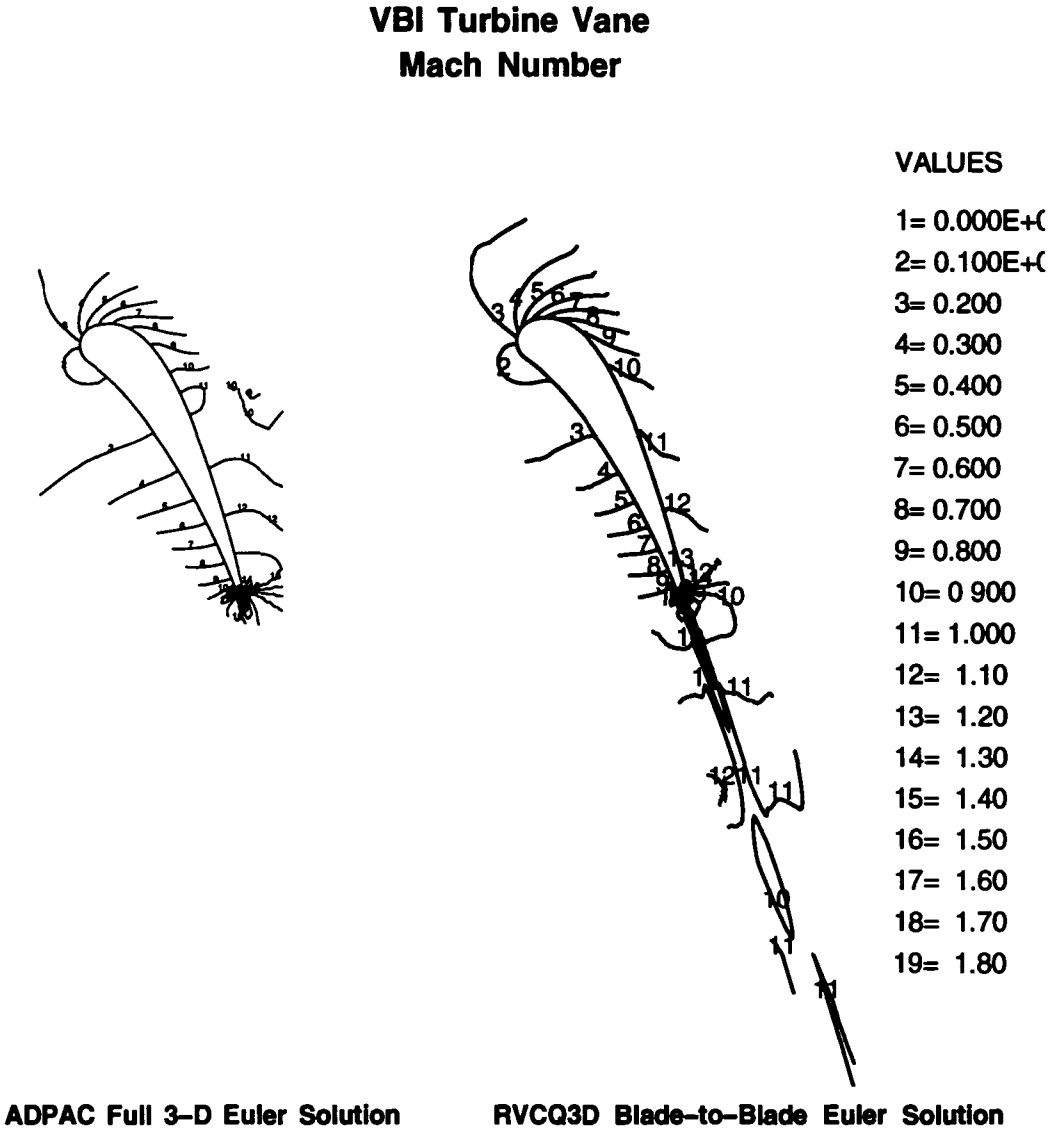


Figure 7.20: The Mach number contours from the midspan section blade-to-blade analysis agree well with the 3-D Euler results.

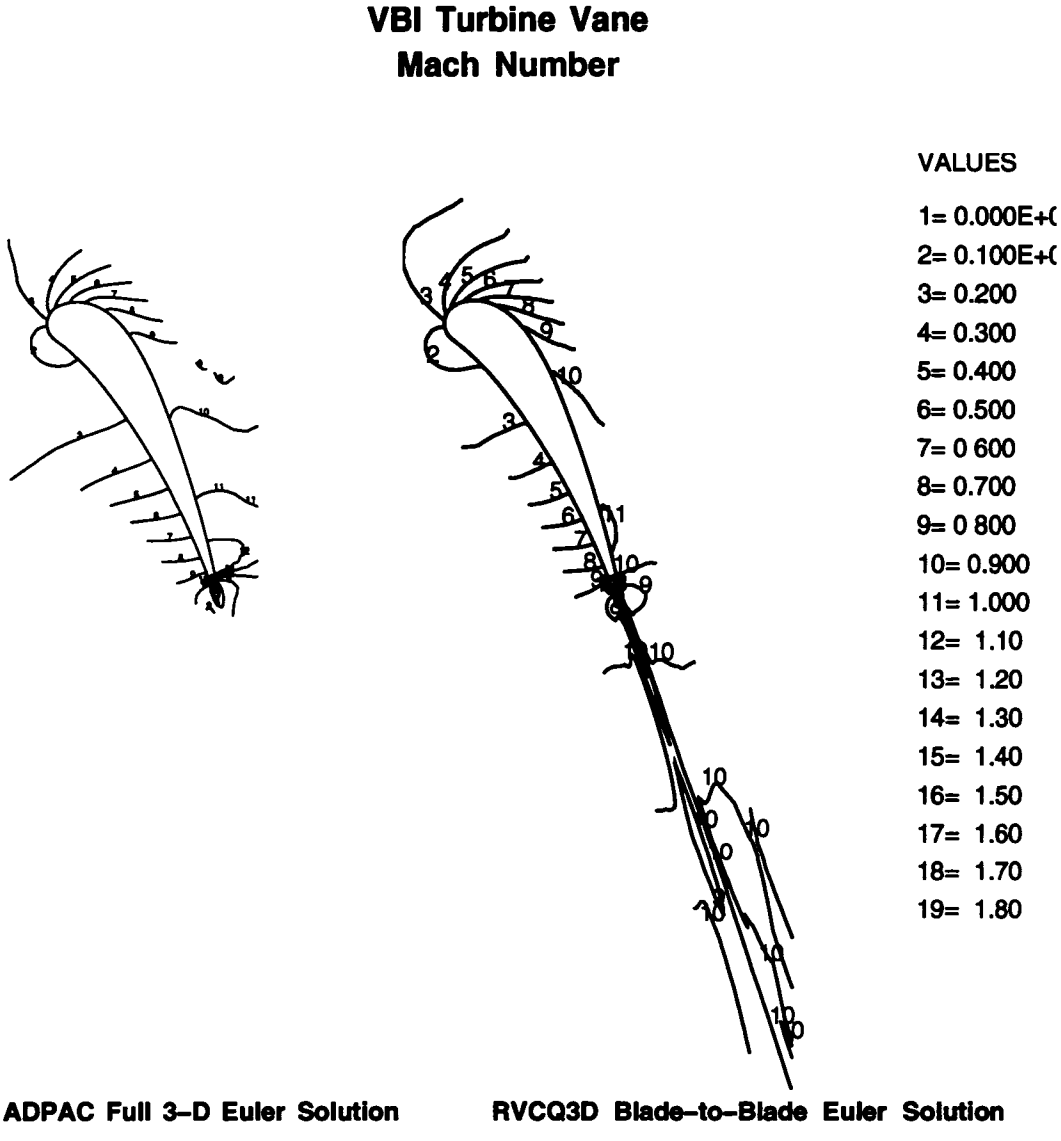


Figure 7.21: The Mach number contours from the tip section blade-to-blade analysis agree well with the 3-D Euler results.

accounting for swirl in the flow.

The test cases presented here demonstrate convincingly that the coupling between the analyses in *TADS* is done correctly. Further, the *TADS* analysis is applicable to a wide range of problems in turbines and compressor airfoil design. There are some difficulties with transonic fans, due to the shock structure. Because the actual shock structure is not axisymmetric, the throughflow analysis does not predict the the same flow pattern as the axisymmetric average of a 3-D prediction in the bladed region. This affects the location of the meridional streamlines, and in turn, the blade-to-blade analysis. The *TADS* predictions are good within the limits of the assumptions in the analysis, but oblique shock waves are not modeled properly in an axisymmetric calculation.

Chapter 8

Conclusions

A turbomachinery airfoil analysis system has been developed by coupling a throughflow analysis with a blade-to-blade analysis. This analysis, the Turbomachinery Analysis and Design System *TADS*, enables a designer to analyze airfoil shapes without the expense of a full 3-D calculation. A GUI was developed to assist the user in controlling the work flow in the analysis. Input panels were developed for each task in the analysis, and capability was included to run each task on a remote host. Programs were developed to link the various grid generators and flow solvers, passing information between them by way of files. The system was designed to enable new codes to be added to the list of choices for any of the major tasks (e.g.grid generation, throughflow analysis, or blade-to-blade analysis).

The throughflow analysis was developed by adding body force and blockage terms to the *ADPAC* code. These terms model the presence of the airfoil in the axisymmetric flow: the body force terms enforce a turning distribution, and the blockage term simulates the airfoil thickness. Convergence acceleration techniques such as multigrid and implicit residual smoothing were preserved in the throughflow analysis. The newly developed throughflow analysis was verified with simple test cases and with NASA Rotor 67.

The total coupled analysis was applied to four test cases: NASA Rotor 67, the AST compressor fifth stator, the Purdue Low Speed Turbine Rig first rotor, and the VBI turbine vane. These cases spanned the flow speed regime from incompressible to transonic flow. The body force and blockage terms were verified with highly cambered, thick airfoils as well as thin low camber shapes. In each case, the *TADS* system converged to a reasonable solution,

comparing favorably with 3-D Euler calculations performed at the same flow conditions. As a coupled system of codes, iteration is required to converge the *TADS* analysis. In all cases, *TADS* converged in three or fewer iterations through the coupled throughflow and blade-to-blade solutions. With the exception of flows with strong shock waves, the analysis has been shown to be a good approximation to a 3-D analysis.

Bibliography

[1] Adamczyk, J., "Model Equation for Simulating Flows in Multistage Turbomachinery," ASME Paper 85-GT-226, 1985.

[2] Chima, R., "Explicit Multigrid Algorithm for Quasi-Three-Dimensional Viscous Flows in Turbomachinery," Journal of Propulsion and Power, Vol. 3 No. 5, 1987.

[3] Chima, R., Turkel, E. Schaffer, S., "Comparison of Three Explicit Multigrid Methods for the Euler and Navier-Stokes Equations," NASA TM88878, Jan., 1987.

[4] Chima, R., "Revised GRAPE Code Input for Cascades," NASA-Lewis Research Center, June, 1990.

[5] Damle, S., Dang, T., and Reddy, D. R., "Throughflow method for Turbomachines Applicable for All Flow Regimes," ASME Paper 95-GT-395, 1995.

[6] Delaney, R., Helton, D., Bennett, W., Dunn, M., Rao, K.V., and Kwon, O., "Turbine Vane-Blade Interaction," WRDC-TR-89-2154, March 1990.

[7] Hall, E., Topp, D., and Delaney, R., "Task 7 - ADPAC User's Manual" NASA CR195472, 1995.

[8] Hall, E., Topp, D., Heidegger, N., McNulty, G., Weber, K., and Delaney, R., "Endwall Treatment Inlet Flow Distortion Final Report," NASA CR195468, 1995.

[9] Jennions, I. K., and Stow, P., "A Quasi-Three-Dimensional Turbomachinery Blade Design System: Part 1 - Throughflow Analysis," ASME Paper 84-GT-26, 1984.

[10] Lieblein, S., "Experimental Flow in Two-Dimensional Cascades," Aerodynamic Design of Axial FLow Compressors, NASA SP-36,1965.

[11] Miller, D., and Reddy, D., "The Design/Analysis of Flows Through Turbomachinery in a Viscous/Invisid Approach," AIAA Paper AIAA-91-2010.

[12] Miller, D., "TIGGERC - Turbomachinery Interactive Grid Generator for 2-D Grid Applications and Users Guide," NASA TM106586, 1994

[13] Pierzga, M., and Wood, J., "Investigation of the Three-Dimensional Flow Field Within a Transonic Fan Rotor: Experiment and Analysis," ASME Journal of Engineering for Gas Turbines and Power, Vol. 107, pp. 436-449, 1985.

[14] Overmars, M., "Forms Library: A Graphical User Interface Toolkit for Silicon Graphics Workstations," Department of Computer Science, Utrecht University, The Netherlands, December, 1991.

[15] Sayari, N. and Bolcs, A., "A New Throughflow Approach for Transonic Axial Compressor Stage Analysis," ASME Paper 95-GT-195, 1995.

[16] Sorenson, R., "A Computer Program to Generate Two-Dimensional Grids About Airfoils and Other Shapes by Use of Poisson's Equation," NASA TM81198, 1980.

[17] Steger, J., and Sorenson, R., "Automatic Mesh Point Clustering Near a Boundary in Grid Generation with Elliptic Partial Differential Equations," Journal of Computational Physics, Vol. 33, Dec. 1979, pp.405-410.

[18] Spurr, A., "The Prediction of 3D Transonic Flow in Turbomachinery Using a Combined Throughflow and Blade-to-BLade Time Marching Method," Intl. Journal of Heat and Fluid Flow Vol. 2 No. 4, 1980.

[19] Walatka, P., Buning, P., Pierce, L, Elson, P., "PLOT3D User's Manual, Version 3.6" NASA TM101067, 1990.

[20] Whipple, D., "BDX-Binary Data Exchange Preliminary Information," NASA-Lewis Research Center 1989.

[21] Wu, C., "A General Theory of Three Dimensional Flow in Subsonic and Supersonic Turbomachines of Axial, Radial and Mixed Flow Types," NACA TN2604, 1952.

[22] Zhao, T., and Overmars, M., "Forms Library: A Graphical User Interface Toolkit for X" Department of Physics, University of Wisconsin-Milwaukee, Milwaukee, WI, 1995.

[23] Zheng, Y., and Hirsch, C., "Throughflow Model Using 3D Euler or Navier-Stokes Solvers,": European Turbomachinery Conference, Elangen, Germany, March 1-3, 1995.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1 AGENCY USE ONLY (*Leave blank*)	2 REPORT DATE	3 REPORT TYPE AND DATES COVERED
	December 1995	Final Contractor Report

4 TITLE AND SUBTITLE

TADS–A CFD-Based Turbomachinery and Analysis Design System With GUI
Volume I—Method and Results

5 FUNDING NUMBERS

WU–505–62–10
C–NAS3–25950

6. AUTHOR(S)

D A Topp, R A Myers, and R A Delaney

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Allison Engine Company
P O Box 420
Indianapolis, Indiana 46206–0420

8 PERFORMING ORGANIZATION REPORT NUMBER

E–10058

9 SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135–3191

10 SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA CR–198440

11 SUPPLEMENTARY NOTES

Project Manager, Kestutis C Civinskas, Propulsion Systems Division, NASA Lewis Research Center, organization code 2760, (216) 433–3944

12a DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 07

This publication is available from the NASA Center for Aerospace Information, (301) 621–0390

12b DISTRIBUTION CODE

13 ABSTRACT ***(Maximum 200 words)***

The primary objective of this study was the development of a CFD (Computational Fluid Dynamics) based turbomachinery airfoil analysis and design system, controlled by a GUI (Graphical User Interface) The computer codes resulting from this effort are referred to as *TADS* (Turbomachinery Analysis and Design System) This document is the Final Report describing the theoretical basis and analytical results from the *TADS* system, developed under Task 18 of NASA Contract NAS3–25950, *ADPAC* System Coupling to Blade Analysis & Design System GUI *TADS* couples a throughflow solver (*ADPAC*) with a quasi-3D blade-to-blade solver (*RVCQ3D*) in an interactive package Throughflow analysis capability was developed in *ADPAC* through the addition of blade force and blockage terms to the governing equations A GUI was developed to simplify user input and automate the many tasks required to perform turbomachinery analysis and design The coupling of the various programs was done in such a way that alternative solvers or grid generators could be easily incorporated into the *TADS* framework Results of aerodynamic calculations using the *TADS* system are presented for a highly loaded fan, a compressor stator, a low speed turbine blade and a transonic turbine vane

14 SUBJECT TERMS	15 NUMBER OF PAGES
Computational fluid dynamics, Design, Turbomachinery	121
	16 PRICE CODE: A06

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20 LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

End of Document