

NASA/CR—1999-206600



ADPAC v1.0 – User’s Manual

Edward J. Hall, Nathan J. Heidegger, and Robert A. Delaney
Allison Engine Company, Indianapolis, Indiana

February 1999

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the Lead Center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.

- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.
- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized data bases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at <http://www.sti.nasa.gov>
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA Access Help Desk at (301) 621-0134
- Telephone the NASA Access Help Desk at (301) 621-0390
- Write to:
NASA Access Help Desk
NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076

NASA/CR—1999-206600



ADPAC v1.0 – User’s Manual

Edward J. Hall, Nathan J. Heidegger, and Robert A. Delaney
Allison Engine Company, Indianapolis, Indiana

Prepared under Contract NAS3-27394 Task 15

National Aeronautics and
Space Administration

Lewis Research Center

February 1999

Available from

NASA Center for Aerospace Information
7121 Standard Drive
Hanover, MD 21076
Price Code: A12

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22100
Price Code: A12

Contents

1	SUMMARY	1
2	INTRODUCTION	3
2.1	Multiple-Block Solution Domain Concepts	3
2.2	Multiple Blade Row Solution Concepts	5
2.3	2-D/3-D Solution Zooming Concepts	9
2.4	Multi-grid Convergence Acceleration Concepts	10
2.5	General Solution Procedure Sequence	11
2.6	Consolidated Serial/Parallel Code Capability	15
2.7	Parallelization Strategy	15
3	ADPAC : 3-D EULER/ NAVIER-STOKES FLOW SOLVER OPERATING IN- STRUCTIONS	17
3.1	Introduction to <i>ADPAC</i>	17
3.2	General Information Concerning the Operation of the <i>ADPAC</i> Code	17
3.3	Configuring <i>ADPAC</i> Maximum Array Dimensions	18
3.4	<i>ADPAC</i> Compilation Using Makefile	22
3.5	<i>ADPAC</i> Input/Output Files	27
3.6	<i>ADPAC</i> Standard Input File Description	29
3.7	<i>ADPAC</i> Boundary Data File Description	53
	BCINT1	62
	BCINTM	66
	BCPRM	71
	BCPRR	74
	BCDATIN	79
	BCDATOU	82
	ENDDATA	85
	ENDTTA	87
	EXITG	90
	EXITN	93
	EXITP	95
	EXITT	98
	EXITX	101
	EXT2DP	104

EXT2DT	107
FIXED	110
FRE2D	112
FREE	114
INL2DA	117
INL2DT	119
INLETA	121
INLETG	124
INLETN	127
INLETR	129
INLETT	132
INLETX	135
KIL2D	138
KILL	141
LAMSS	144
MBCAVG	147
PATCH	150
PINT	154
PROBE	156
SSIN	159
SSVI	161
SYSTEM	164
3.8 Mesh File Description	166
3.9 Body Force File Description	168
3.10 Standard Output File Description	170
3.11 Plot File Description	170
3.12 Restart File Description	172
3.13 Convergence File Description	175
3.14 Image File Description	176
3.15 Running <i>ADPAC</i> With The One-Equation Turbulence Model	176
3.16 Running <i>ADPAC</i> With Two-Equation Turbulence Model	177
3.17 Troubleshooting an <i>ADPAC</i> Failure	178
4 RUNNING ADPAC IN PARALLEL	183
4.1 Description of Parallel Solution Sequence	183
4.2 <i>SIXPAC</i> (Block Subdivision) Program	188
4.3 <i>BACPAC</i>	192
4.4 Parallel <i>ADPAC</i> Block/Processor Assignment	194
5 ADPAC INTERACTIVE GRAPHICS DISPLAY	197
5.1 Setting up the Program	197
5.2 Graphics Window Operation	198
5.3 <i>AGTPLT-LCL</i> Program Description	199

6	ADPAC UTILITY PROGRAMS	201
6.1	<i>MAKEADGRID</i> Program Description	201
6.2	<i>COARSEN</i> Tool Program Description	201
6.3	<i>ADSPIN</i> Tool Program Description	202
6.4	<i>ADSTAT</i> Tool Program Description	202
6.5	<i>AOA2AXI</i> Tool Program Description	202
6.6	<i>PATCHFINDER</i> Tool Program Description	203
6.7	Miscellaneous Tool Programs Description	204
6.8	<i>PLOTBC</i> Tool Program Description	204
A	ADPAC NAVIER-STOKES NUMERICAL ALGORITHM	211
A.1	Nondimensionalization	211
A.2	Governing Equations	212
A.3	Fluid Properties	219
A.4	Numerical Formulation	219
A.5	Boundary Conditions	232
A.6	Turbulence Models	235
A.7	Algebraic Baldwin-Lomax Turbulence Model	236
A.8	One-Equation Spalart-Allmaras Turbulence Model	240
A.9	Two-Equation Turbulence Model	242
B	PARALLEL ADPAC EXECUTION SCRIPT	247

List of Figures

- 2.1 *ADPAC* 2-D single-block and multiple-block mesh structure illustration. 4
- 2.2 Coupled H-O-H grid system and computational domain communication scheme for compressor fan grid system. For clarity, only a single j =constant mesh slice extracted from a 3-D mesh system is shown. 6
- 2.3 Several approaches can be used to obtain multiple blade row numerical solutions. 7
- 2.4 2-D axisymmetric flow representation of a turbomachinery blade row. 9
- 2.5 Typical fan rotor flowpath geometry including bypass splitter. 10
- 2.6 Multi-grid mesh coarsening strategy and mesh index relation. 11
- 2.7 Typical sequence of tasks employed during an *ADPAC* analysis. 12

- 3.1 Sample *ADPAC* parameter file specification (parameter.inc). 19
- 3.2 Sample *ADPAC* input file specification (case.input). 31
- 3.3 Variation of C_{ep} and C_{Kleb} with Coles wake factor (Π). 33
- 3.4 *ADPAC* body-centered mesh turbulence model nomenclature summary. 35
- 3.5 *ADPAC* input keyword multi-grid cycle and time-marching iteration management flowchart. 39
- 3.6 *ADPAC* rotational speed orientation illustration. 52
- 3.7 Location of **XMOM**, **YMOM**, and **ZMOM** with respect to the calculation of moment components. 54
- 3.8 *ADPAC* 3-D boundary condition specification. 55
- 3.9 Effect of ordering in application of boundary conditions for the *ADPAC* code. . 56
- 3.10 *ADPAC* boundary data file specification format. 57
- 3.11 Sample *ADPAC* boundary condition data file specification (case.boundata). . . 61
- 3.12 *ADPAC* **INLETR** boundary specification flow angle reference 131
- 3.13 *ADPAC* **INLETT** boundary specification flow angle reference 134
- 3.14 *ADPAC* mesh coordinate reference description. 166

3.15	All <i>ADPAC</i> mesh systems must have a left-handed coordinate system description.	167
3.16	Sample <i>ADPAC</i> convergence file (<i>case.convergence</i>).	175
4.1	Illustration of domain decomposition parallel computing using the <i>ADPAC</i> code on a multiprocessor computing architecture.	184
4.2	Illustration of domain decomposition parallel computing using the <i>ADPAC</i> code on a network-connected workstation cluster computing architecture.	184
4.3	Illustration of <i>ADPAC</i> master-slave coding style and data input/output processing for parallel computing. Note that only the root process (Node 0) needs access to the data disk. The configuration shown represents a nine-node workstation cluster (nodes numbered 0-8).	185
4.4	Illustration of <i>ADPAC</i> code programming structure for parallel communication using the native <i>APPL</i> interprocessor communication interface.	186
4.5	Illustration of <i>ADPAC</i> code programming structure for parallel communication using the native <i>APPL</i> procedure calls, <i>applpvm</i> translation library, and the <i>PVM</i> interprocessor communication library.	186
4.6	Illustration of <i>ADPAC</i> code programming structure for parallel communication using the native <i>APPL</i> procedure calls, <i>applmpi</i> translation library, and the <i>MPI</i> interprocessor communication library.	186
4.7	Careful block division can preserve levels of multi-grid.	190
4.8	Sample input file for <i>SIXPAC</i> block division utility.	190
4.9	Sample input file employing user-specified block divisions for <i>SIXPAC</i> block division utility.	191
4.10	Sample input file for <i>BACPAC</i> utility.	193
4.11	Sample <i>case.blkproc</i> file used to distribute mesh blocks over parallel processors within <i>ADPAC</i> .	194
5.1	<i>ADPAC</i> interactive graphics display network configuration options.	198
6.1	Sample input file for <i>PATCHFINDER</i> utility.	203
A.1	<i>ADPAC</i> Cartesian coordinate system reference.	215
A.2	<i>ADPAC</i> cylindrical coordinate system reference.	217
A.3	Three-dimensional finite volume cell.	221
A.4	<i>ADPAC</i> finite volume cell centered data configuration and convective flux evaluation process.	222
A.5	<i>ADPAC</i> finite volume cell centered data configuration and diffusive flux evaluation process.	223

A.6	Multi-grid V-cycle strategy.	229
A.7	2-D mesh block phantom cell representation.	232
A.8	Near-wall computational structure for wall function turbulence model.	239

List of Tables

3.1	Description of input/output files and UNIX-based filenames for <i>ADPAC</i>	28
-----	-------------------------------------------------------------------------------------	----

Notation

A list of the symbols and acronyms used throughout this document and their definitions is provided below for convenience.

Roman Symbols

a ... speed of sound
 c_f ... skin friction coefficient
 c_p ... gas specific heat at constant pressure
 c_v ... gas specific heat at constant volume
 e ... total internal energy
 i ... first grid index of numerical solution
 j ... second grid index of numerical solution
 k ... third grid index of numerical solution or thermal conductivity
 k ... turbulent kinetic energy
 l ... Van Driest damping function
 n ... rotational speed (revolutions per second) or time step level
 p ... pressure
 r ... radius or radial coordinate
 t ... time
 v_x ... velocity in the Cartesian coordinate system x direction
 v_y ... velocity in the Cartesian coordinate system y direction
 v_z ... velocity in the Cartesian coordinate system z direction
 v_r ... velocity in the cylindrical coordinate system radial direction
 v_θ ... velocity in the cylindrical coordinate system circumferential direction
 w_{rel} ... relative velocity in the circumferential direction ($= v_\theta - r\omega$)
 x ... Cartesian coordinate system coordinate
 y ... Cartesian coordinate system coordinate
 z ... Cartesian coordinate system coordinate
 A^+ ... turbulence model constant
ADPAC ... Advanced Ducted Propfan Analysis Code Version v1.0
ADSPIN ... ADPAC post processing program
APPL ... NASA Application Portable Parallel Library
ASCII ... American Standard Code for Information Interchange

CFL... Courant-Freidrichs-Lewy number ($\Delta t/\Delta t_{max,stable}$)
D... diameter
F... *i* coordinate direction flux vector
FAST... NASA Flow Analysis Software Toolkit
G... *j* coordinate direction flux vector
GRIDGEN... Multiple block general purpose mesh generation system
H... *k* coordinate direction flux vector
H_{total}... total enthalpy
K... cylindrical coordinate system source vector
L... reference length
M... Mach number
MAKEADGRID... ADPAC multiple-block mesh assembly program
N... Number of blades
Q... vector of conserved variables
P... turbulence kinetic energy production term
PATCHFINDER... Multiple block mesh boundary data file construction routine
PLOT3D... NASA graphics flow visualization program
Pr... gas Prandtl Number
R... gas constant or residual or maximum radius
R... turbulent Reynolds number
Re... Reynolds Number
S... surface area normal vector
SDBLIB ... Scientific DataBase Library (binary file I/O routines)
T... Temperature
U... Freestream velocity (units of length/time)
V... volume

Greek Symbols

γ ... specific heat ratio
 Δ ... calculation increment
 ϵ ... turbulence dissipation parameter
 ∇ ... gradient vector operator
 ω ... vorticity
 ρ ... density
 μ ... coefficient of viscosity
 τ ... fictitious time or shear stress
 $\Pi_{i,j}$... fluid stress tensor

Subscripts

$[]_1$... inlet value
 $[]_2$... exit value
 $[]_{ax}$... pertaining to the axial (*x*) cylindrical coordinate

$[]_{coarse}$... coarse mesh value
 $[]_{effective}$... effective value
 $[]_{fine}$... fine mesh value
 $[]_{freestream}$... freestream value
 $[]_{i,j,k}$... grid point index of variable
 $[]_{laminar}$... laminar flow value
 $[]_{max}$... maximum value
 $[]_{min}$... minimum value
 $[]_{nearwall}$... near wall value
 $[]_{non-dimensional}$... non-dimensional value
 $[]_r$... pertaining to the radial (r) cylindrical coordinate
 $[]_{ref}$... reference value
 $[]_{stable}$... value implied by linear stability
 $[]_t$... turbulent flow value
 $[]_{total}$... total (stagnation) value
 $[]_{turbulent}$... turbulent flow value
 $[]_{wall}$... value at the wall
 $[]_x$... pertaining to the x Cartesian coordinate
 $[]_y$... pertaining to the y Cartesian coordinate
 $[]_z$... pertaining to the z Cartesian coordinate
 $[]_\theta$... pertaining to the circumferential (θ) cylindrical coordinate

Superscripts

$[]^+$... Turbulent velocity profile coordinate
 $[]^*$... Intermediate value
 $[]^n$... Time step index
 $[]$... (no overscore) nondimensional variable
 $[\hat{\quad}]$... Dimensional variable
 $[\bar{\quad}]$... Time-averaged variable
 $[\tilde{\quad}]$... Density-weighted time-averaged variable
 $[\vec{\quad}]$... Vector variable
 $[\overset{\rightharpoonup}{\quad}]$... Tensor variable

Chapter 1

SUMMARY

The current version of the computer codes described within this User's Manual is referred to as *ADPAC v1.0* (Advanced Ducted Propfan Analysis Codes - Version 1.0). The *ADPAC* program solves a discretized form of the Navier-Stokes equations based on a flexible multiple-block grid discretization scheme permitting coupled 2-D/3-D mesh block solutions with application to a wide variety of geometries. Aerodynamic calculations are based on a four-stage Runge-Kutta time-marching finite volume solution technique with added numerical dissipation. Steady flow predictions are accelerated by a multi-grid procedure. The code is capable of executing in either a serial or parallel computing mode from a single source code.

This Page Intentionally Left Blank

Chapter 2

INTRODUCTION

This document contains the Computer Program User's Manual for the *ADPAC v1.0* (Advanced Ducted Propfan Analysis Codes - Version 1.0) Euler/Navier-Stokes analysis code developed by the Allison Engine Company under NASA sponsorship. The objective was to develop a three-dimensional time-marching Euler/Navier-Stokes analysis tool for aerodynamic and/or heat transfer analysis of modern turbomachinery configurations. *ADPAC* is capable of predicting both steady state and time-dependent flowfields using coupled 2-D and 3-D solution zooming concepts (described in detail in Section 2.3). The code was developed to be capable of either serial execution or parallel execution on massively parallel or workstation cluster computing platforms from a single source. Throughout the rest of this document, this aerodynamic analysis code is referred to as *ADPAC*.

A theoretical development of the analyses in the *ADPAC* program is outlined in Appendix A for reference. Additional information is presented in the Final Reports from the work of Tasks V [1], VII [2], and VIII [3] of NASA Contract NAS3-25270. In brief, the program utilizes a finite-volume, time-marching numerical procedure in conjunction with a flexible, coupled 2-D/3-D multiple grid block geometric representation to permit detailed aerodynamic simulations about complex configurations. The analysis has been tested and results verified for both turbomachinery and non-turbomachinery based applications. The ability to accurately predict the aerodynamics due to the interactions between adjacent components of modern, high-speed turbomachinery was of particular interest during this program, and therefore, emphasis is given to these types of calculations throughout the remainder of this document. It should be emphasized at this point that although the *ADPAC* program was developed to analyze the steady and unsteady aerodynamics of high-bypass ducted fans employing multiple blade rows, the code possesses many features which make it practical to compute a number of other complicated flow configurations as well.

2.1 Multiple-Block Solution Domain Concepts

In order to appreciate and utilize the features of the *ADPAC* solution system, the concept of a multiple-block grid system must be fully understood. It is expected that the reader

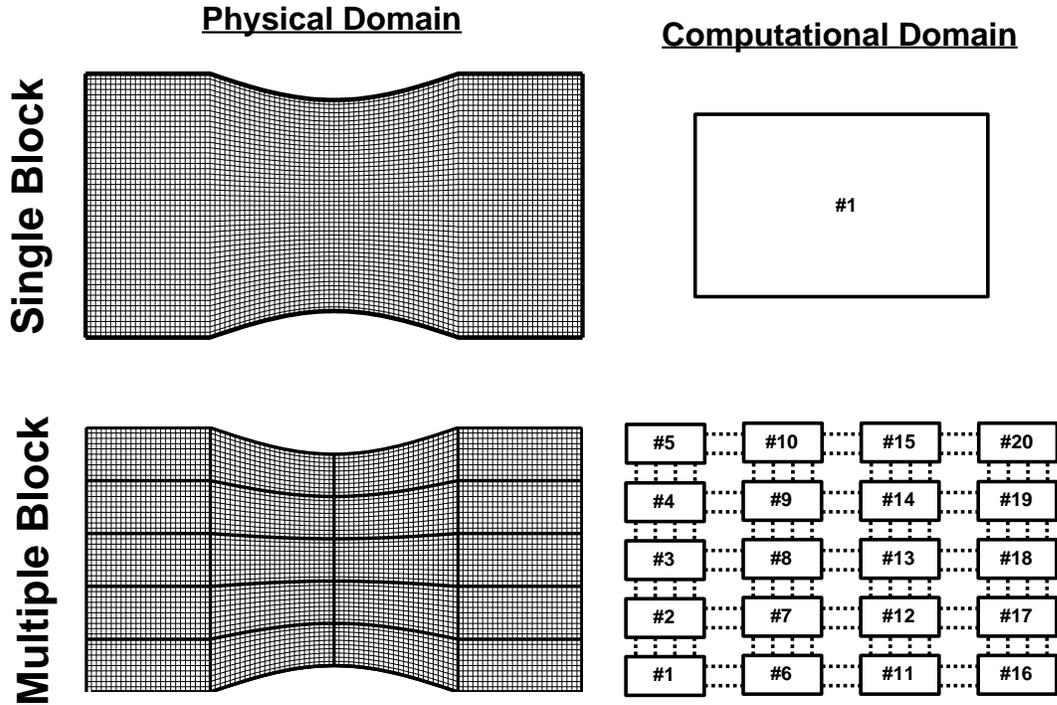


Figure 2.1: ADPAC 2-D single-block and multiple-block mesh structure illustration.

possesses at least some understanding of the concepts of computational fluid dynamics (CFD), so the use of a numerical grid to discretize a flow domain should not be foreign. Many CFD analyses rely on a single structured ordering of grid points upon which the numerical solution is performed. Multiple-block grid systems are different only in that several structured grid systems are used in harmony to generate the numerical solution. This concept is illustrated graphically in two dimensions for the flow through a nozzle in Figure 2.1.

The grid system in the top of Figure 2.1 employs a single structured ordering, resulting in a single computational space with which to contend. In theory, the nozzle flowpath could be subdivided into any number of domains employing structured grid blocks resulting in an identical number of computational domains to contend with, as shown in the 20 block decomposition illustrated in the bottom of Figure 2.1. The complicating factor in this domain decomposition approach is that the numerical solution must provide a means for the isolated computational domains to communicate with each other in order to satisfy the conservation laws governing the desired aerodynamic solution. Hence, as the number of subdomains used to complete the aerodynamic solution grows, the number of inter-domain communication paths increases in a corresponding manner. (It should be noted that this domain decomposition/communication overhead relationship is also a key concept in parallel processing for large scale computations, and thus, the ADPAC code possesses a natural domain decomposition division for parallel processing afforded by the multiple-block grid data structure.)

For the simple nozzle case illustrated in Figure 2.1 it would seem that there is no real advantage in using a multiple-block grid, and this is probably true. For more

complicated geometries, such as the turbine vane coupled O-H grid system shown in the top of Figure 2.2 and the corresponding computational domain communication scheme shown in the bottom Figure 2.2, it may not be possible to generate a single structured grid to encompass the domain of interest without sacrificing grid quality, and therefore, a multiple-block grid system has significant advantages.

The *ADPAC* code utilizes the multiple-block grid concept to the full extent by permitting an arbitrary number of structured grid blocks with user specifiable communication paths between blocks. The inter-block communication paths are implemented as a series of boundary conditions on each block which, in some cases, communicate flow information from one block to another. The advantages of the multiple-block solution concept are exploited throughout the remainder of this document as a means of treating complicated geometries, multiple blade row turbomachines of varying blade number, endwall treatments, and to exploit computational enhancements such as multi-grid.

2.2 Multiple Blade Row Solution Concepts

Armed with an understanding of the multiple-block mesh solution concept discussed in the previous section, it is now possible to describe how this numerical solution technique can be applied to predict complicated flows. Specifically, this section deals with the prediction of flows through rotating machinery with multiple blade rows. Historically, the prediction of three-dimensional flows through multistage turbomachinery has been based on one of three solution schemes. These schemes are briefly illustrated and described in Figure 2.3.

The first scheme involves predicting the time-resolved unsteady aerodynamics resulting from the interactions occurring between relatively rotating blade rows. Examples of this type of calculation are given by Rao and Delaney [4], Jorgensen and Chima [5], and Rai [6]. This approach requires either the simulation of multiple blade passages per blade row, or the incorporation of a phase-lagged boundary condition to account for the differences in spatial periodicity for blade rows with dissimilar blade counts. Calculations of this type are typically computationally expensive, and are presently impractical for machines with more than 2-3 blade rows.

The second solution technique is based on the average-passage equation system developed by Adamczyk [7]. In this approach, separate 3-D solution domains are defined for each blade row which encompasses the overall domain for the entire turbomachine. The individual solution domains are specific to a particular blade row, although all blade row domains share a common axisymmetric flow. In the solution for the flow through a specific blade passage, adjacent blade rows are represented by their time and space-averaged blockage, body force, and energy source contributions to the overall flow. A correlation model is used to represent the time and space-averaged flow fluctuations representing the interactions between blade rows. The advantage of the average-passage approach is that the temporally and spatially-averaged equation system reduces the problem to a steady flow case, and, within the accuracy of the correlation model, the solution is representative of the average aerodynamic condition experienced by a given blade row under the influence of all other blade rows in the machine. The disadvantage of the average-passage approach is that the solution complexity and cost grow rapidly as the number of blade

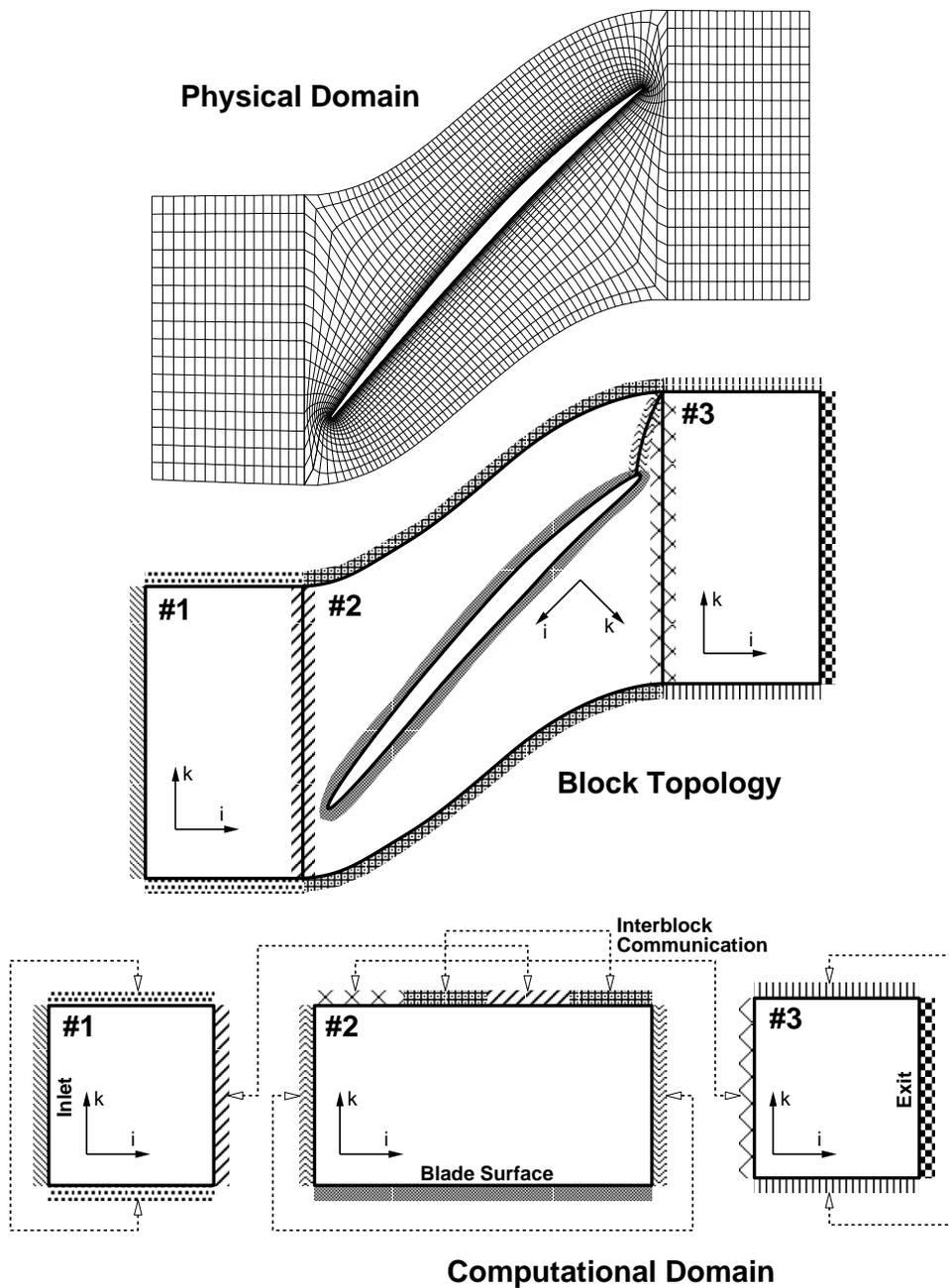
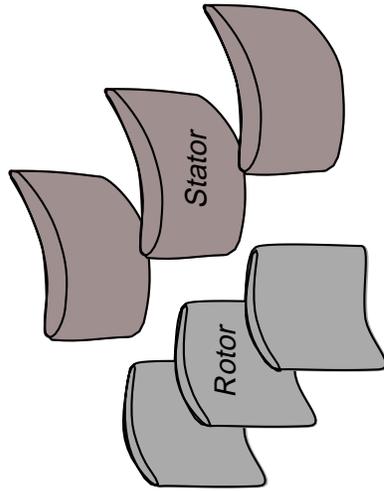


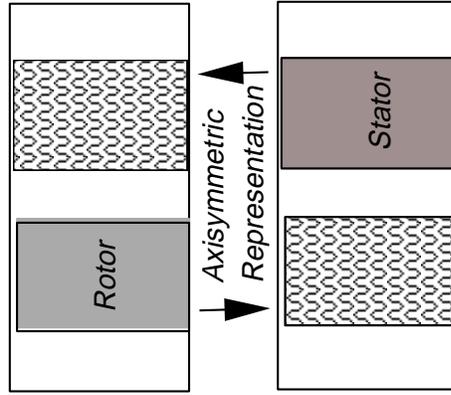
Figure 2.2: Coupled H-O-H grid system and computational domain communication scheme for compressor fan grid system. For clarity, only a single j -constant mesh slice extracted from a 3-D mesh system is shown.

3-D Rotor/Stator Interaction



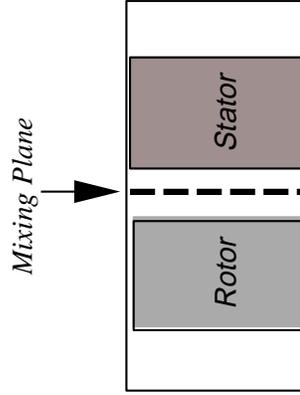
- 3-D time-dependent Navier-Stokes equations
- Multiple blade passages for each blade row or phase-lagged boundaries
- Time-dependent coupling of individual blade passage domains
- Computationally expensive – Multiple blade passages per blade row

Average-Passage Simulation



- Average-passage equation system
- 3-D steady solution of entire domain for each blade row
- Adjacent blade rows represented by blockage/body forces in 3-D solution
- Solutions have common axisymmetric flowfield
- Correlation model for mixing terms
- Computational cost still rather high

Circumferential Mixing Plane



- Steady Navier Stokes solution
- Computational domain limited to near blade region
- Circumferential mixing plane provides inter-blade row communication
- Lower computational cost

Figure 2.3: Several approaches can be used to obtain multiple blade row numerical solutions.

rows increases, and the accuracy of the correlation model is as yet unverified.

The third approach for the prediction of flow through multistage turbomachinery is based on the mixing plane concept. A mixing plane is an arbitrarily imposed boundary inserted between adjacent blade rows across which the flow is “mixed out” circumferentially. This circumferential mixing approximates the time-averaged condition at the mixing plane and allows the aerodynamic solution for each blade passage to be performed in a steady flow environment. The mixing plane concept was applied to realistic turbofan engine configurations by Dawes [8, 9] and extensive validation is available in recent work [10, 11]. Flow variables on either side of the mixing plane are circumferentially averaged and passed to the neighboring blade row as a means of smearing out the circumferential non-uniformities resulting from dissimilar blade counts. The mixing plane concept is a much more computationally cost-effective approach because the flow is steady, and the individual blade passage domains are limited to a near-blade region. Unfortunately, the accuracy of this approach is clearly questionable under some circumstances because of the placement of the mixing plane and the loss of spatial information resulting from the circumferential averaging operator.

The *ADPAC* program possesses features which permit multiple blade row solutions using either the time-dependent interaction approach or the mixing plane concept, as described above. Average-passage simulations for realistic turbofan engine configurations were reported under previous work [12]. Because there is no phase-lagged boundary conditions, *ADPAC* predictions utilizing the time-accurate rotor/stator interaction technique require that a sufficient number of blade passages be represented in each row such that the circumferential distance represented in each blade row is constant. This limits the blade counts which can be effectively simulated through this technique. For example, for the simple single-stage calculation suggested in Figure 2.3, if the rotor has 36 blades and the stator has 48 blades, a time dependent solution would require, as a minimum, 3 rotor blade passages and 4 stator blade passages to accommodate the common circumferential pitch requirement. If the rotor has 35 blades, and the stator has 47 blades, however, then both blade rows would require that every blade passage be modeled as there are no common divisors of the blade counts. This restriction will appear quite often, as turbomachinery designers often do not like to design neighboring blade rows with blade counts which have a common integer factor. Ultimately, this type of problem will require the incorporation of a phase-lagged boundary condition which would permit time-dependent interaction solutions for neighboring blades using only one blade passage per blade row.

If, instead, a mixing plane type of calculation is desired, then the multiple block scheme may again be invoked by utilizing a single blade passage per blade row, where each grid block has a common mating surface with a neighboring blade row. The only special requirement here is that boundary condition routines be available to adequately perform the circumferential averaging between blade rows and supply the block-to-block communication of this information in the multiple-block mesh solution algorithm. Section 3.7 describes the techniques for applying this type of boundary condition.

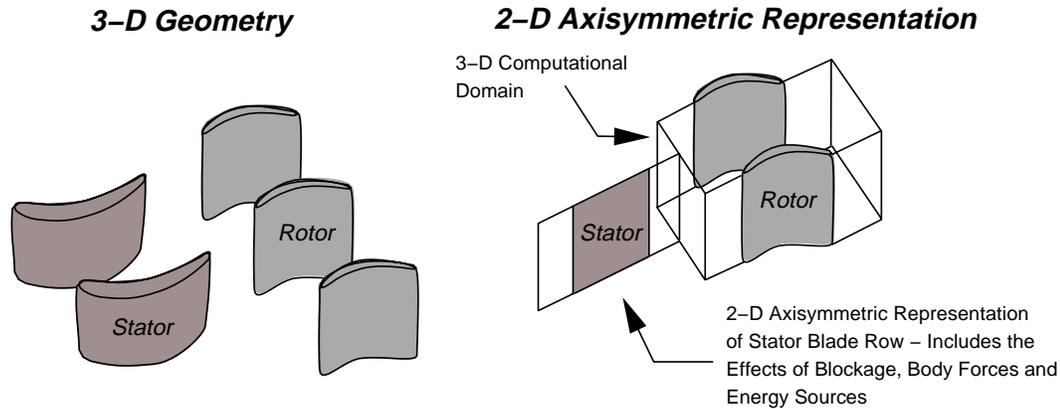


Figure 2.4: 2-D axisymmetric flow representation of a turbomachinery blade row.

2.3 2-D/3-D Solution Zooming Concepts

A fourth unique feature of the *ADPAC* solution system involves the concept of coupling two-dimensional and three-dimensional solution domains to obtain representative simulations of realistic high bypass ducted fan engine concepts. A complicating factor in the analysis of flows through turbofan engine systems results from the interactions between adjacent blade rows, and, in the case of a ducted fan, the effects of downstream blade rows on the aerodynamics of the upstream fan rotor. Historically, in the design of multistage turbomachinery, an axisymmetric representation of the flow through a given blade row has been used to effectively reduce the complexity of the overall problems to a manageable level. Similarly, an efficient approach to the numerical simulation of downstream blade rows could naturally utilize an axisymmetric representation of the effects of these rows through a two-dimensional grid system, with blade blockage, body force, and energy terms representing the axisymmetric averaged aerodynamic influence imparted by the embedded blade row. This concept is illustrated graphically in Figure 2.4 for a representative turbine stage.

A numerical solution of the flow through the fan rotor is complicated by the presence of the core stator, bypass stator, and bypass splitter as shown in Figure 2.5. It is undesirable to restrict the solution domain to the fan rotor alone as this approach neglects the potential interactions between the fan rotor and the downstream geometry. The *ADPAC* program permits coupled solutions of 3-D and 2-D mesh blocks with embedded blade row blockage, body force, and energy terms as a means of efficiently treating these more complicated configurations. Blade force terms may be determined from a separate 3-D solution, or may be directly specified based on simpler design system analyses. Neighboring 2-D and 3-D mesh blocks are numerically coupled through a circumferential averaging procedure which attempts to globally satisfy the conservation of mass, momentum and energy across the solution domain interface. The “dimensional zooming” capability permitted by the 2-D/3-D mesh coupling scheme is considered a vital asset for the accurate prediction of the flow through modern high-speed turbofan engine systems.

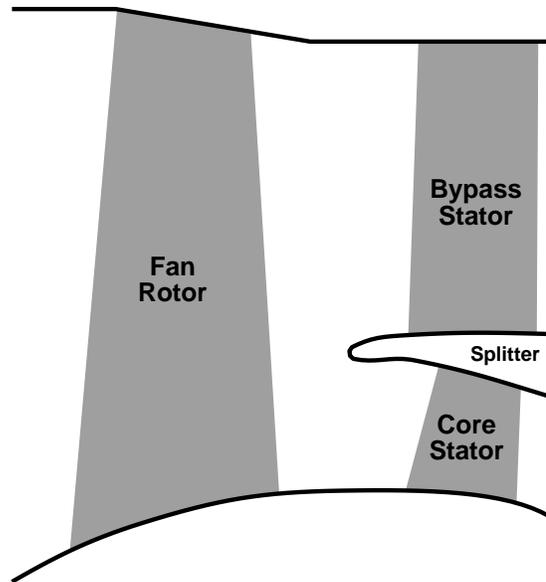


Figure 2.5: Typical fan rotor flowpath geometry including bypass splitter.

2.4 Multi-grid Convergence Acceleration Concepts

For completeness, a brief section is included here to discuss the multi-grid convergence acceleration solution technique incorporated into the *ADPAC* code. Multi-grid (please do not confuse this with a multiple-block grid!) is a numerical solution technique which attempts to accelerate the convergence of an iterative process (such as a steady flow prediction using a time-marching scheme) by computing corrections to the solution on coarser meshes and propagating these changes to the fine mesh through interpolation. This operation may be recursively applied to several coarsenings of the original mesh to effectively enhance the overall convergence. Coarse meshes are derived from the preceding finer mesh by eliminating every other mesh line in each coordinate direction as shown in Figure 2.6. As a result, the number of multi-grid levels (coarse mesh divisions) is controlled by the mesh size, and, in the case of the *ADPAC* code, by the mesh indices of the boundary patches used to define the boundary conditions on a given mesh block (see Figure 2.6). These restrictions suggest that mesh blocks should be constructed such that the internal boundaries and overall size coincide with numbers which are compatible with the multi-grid solution procedure. The mesh size should be 1 greater than any number which can be divided by 2 several times and remain whole numbers (e.g. 9, 17, 33, 65). This is illustrated in the following equation:

$$\frac{n-1}{2^{mg-1}} = \text{integer} \quad (2.1)$$

where n is the index of the feature (block edge or boundary condition limit) and mg is the desired number of multi-grid levels. It is generally recommended to employ 2 or 3 levels of multi-grid if possible. It should also be noted that specifying 1 level of multi-grid is equivalent to solving the flow solution using the fine mesh level *only* and essentially using *no* levels of multi-grid. Further details on the application of the *ADPAC* multi-grid scheme are given in Section 3.6 and in Reference [1].

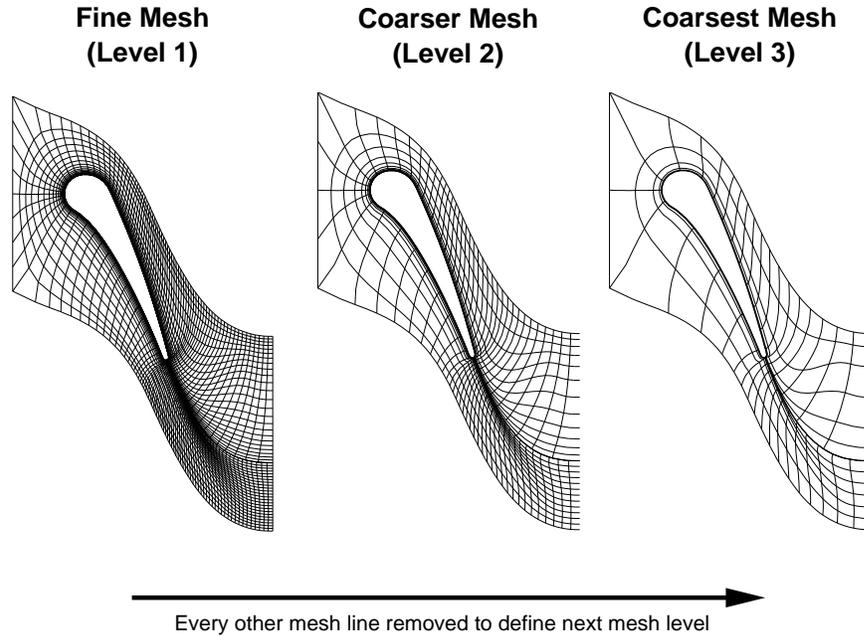


Figure 2.6: Multi-grid mesh coarsening strategy and mesh index relation.

A second multi-grid concept which should be discussed is the so-called “full” multi-grid startup procedure. The “full” multi-grid method is used to start up a solution by initiating the calculation on a coarse mesh, performing several time-marching iterations on that mesh (which, by the way could be multi-grid iterations if successively coarser meshes are available), and then interpolating the solution at that point to the next finer mesh, and repeating the entire process until the finest mesh level is reached. The intent here is to generate a reasonably approximate solution on the coarser meshes before undergoing the expense of the fine mesh multi-grid cycles using a “grid sequencing” technique. Again, the “full” multi-grid technique only applies to starting up a solution, and therefore, it is not normally advisable to utilize this scheme when the solution is restarted from a previous solution as the information provided by the restart data will likely be lost in the coarse mesh re-initialization.

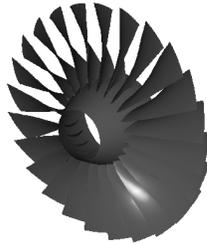
2.5 General Solution Procedure Sequence

The *ADPAC* code is distributed as a compressed *tar* file which must be processed before the code may be utilized. The instructions accompanying the code distribution describe how to extract the necessary data to run the code. For some systems, a UNIX-based compile script is provided to automate the set-up and compile procedure. This operation is typically required only once when the initial distribution is received. Once the source files have been extracted, the sequence of tasks illustrated in Figure 2.7 and described below are typical of the events required to perform a successful analysis using the *ADPAC* code. Separate sections are provided in the chapters which follow to describe in detail the basis and operation of the codes used in the steps below.

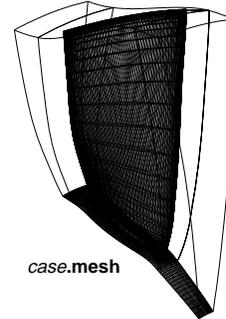
1. Define Problem

Inviscid vs. Viscous
Steady-State vs. Time-Accurate
Flow Conditions
Results Needed

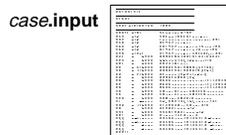
2. Define Geometry



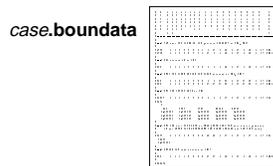
3. Generate Grid



4. Create Input File

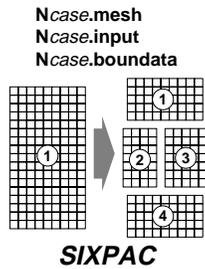


5. Create Boundary Condition File

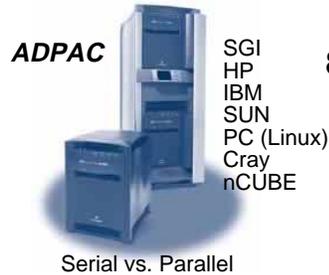


PATCHFINDER

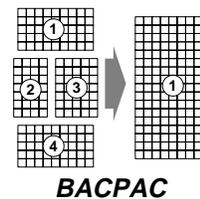
6. Subdivide Domain (Parallel Only)



7. Run Flow Solution



8. Consolidate Domain (Parallel Only)



9. Visualization / Post-Process

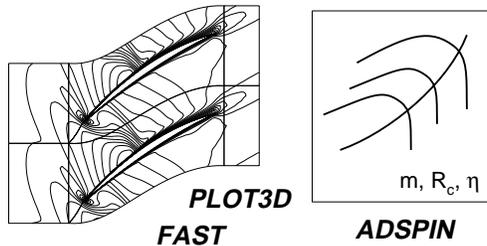


Figure 2.7: Typical sequence of tasks employed during an ADPAC analysis.

Step 1. Define the problem

This step normally involves selecting the geometry and flow conditions, and defining which specific results are desired from the analysis. The definition of the problem must involve specifying whether steady state or time-dependent data are required, whether an inviscid calculation is sufficient, or whether a viscous flow solution is required, and some idea of the relative merits of solution accuracy versus solution cost (CPU time) must be considered.

Step 2. Define the geometry and flow domain

Typically, geometric features such as airfoils, ducts, and flowpath endwalls are required to geometrically define a given problem. The solution domain may be chosen to include the external flow, internal engine passage flows, and/or leakage flows. The flow domain is normally defined large enough such that the region of interest is far enough away from the external boundaries of the problem to ensure that the solution is not unduly influenced by the external boundary conditions.

Step 3. Define a block structure and generate a numerical grid

Once the geometry and solution domain have been numerically defined, the implementation of the solution mesh structure must be considered. This process begins with a determination of the domain block structure, if and when more than one mesh block is required for a given solution. The possibility of incorporating 2-D mesh blocks should be considered whenever possible due to the computational savings afforded by this approach.

Most of the standard grid block structures defined in this document can be adequately handled through commercial packages such as the *GRIDGEN* [13] grid generation program. Other user-created grid generation programs may be equally useful, and a conversion program called *MAKEADGRID* is included to convert non-standard format meshes into *ADPAC* format.

Step 4. Generate a standard input file

The standard input file controls operations specific to a particular run of the *ADPAC* code. Options such as the number of iterations, damping parameters, and input/output control of the code execution may all be governed by the values specified in the standard input file.

Step 5. Generate a boundary data file

The boundary data file controls the application of boundary conditions on the grid block structure provided to the flow code. The boundary data specifications are specific to the mesh being used in a given calculation. For other block configurations, the user must construct the boundary data file by hand according to the format described in Section 3.7. A program is provided (*PATCHFINDER*) in the *ADPAC* standard distribution to aid the user in locating contiguous block interface connections for multiple block meshes.

Step 6. Subdivide the problem for parallel execution

For execution across multiple processors, it may be necessary to subdivide the original block structure to permit the use of additional processors, or to aid in load balancing. The *SIXPAC* program is provided for this purpose.

For parallel calculations using APPL, it is necessary to construct the *procdef* file which defines the computing environment (machine name, number of processes, etc.). The relationship between the number of processors and the number of mesh blocks should not be ignored, as it is up to the user to construct the *case.blkproc* file to adequately balance the overall problem in a multiprocessor computing environment.

Step 7. Run ADPAC to predict the aerodynamic parameters

Chapter 3 describes the commands necessary to perform this step. Depending on the number of mesh points generated *ADPAC* may require the maximum array dimensions to be adjusted and the code recompiled as described in Section 3.3. In many cases, a given calculation will involve several applications of the *ADPAC* code, restarted from the previous calculation as a means of breaking up a large problem into several shorter calculations.

Step 8. Consolidate the block structure

For solutions which have utilized the block subdivision process (see Step 7 above) it may be useful to consolidate the subdivided problem back into the original block structure. The *BACPAC* program is provided for this purpose.

Step 9. Plot and post-process the results

An interactive post processing program called *ADSPIN* is provided to handle tasks such as mass-averaging flow variables to simplify the interpretation of the computed results (see Chapter 6). Output data is also provided for widely available plotting programs such as *PLOT3D* [14] and *FAST* [15].

It is worthwhile mentioning that the development and application of the codes described in this manual were performed on Unix-based computers. All files are stored in machine-independent format. Small files utilize standard ASCII format, while larger files, which benefit from some type of binary storage format, use the Scientific DataBase Library (*SDBLIB*) format [16, 17]. The *SDBLIB* machine-dependent input/output routines provide machine independence of the binary data files. The *SDBLIB* routines were developed at the NASA Lewis Research Center.

Most of the plotting and graphical post-processing of the solutions is performed on graphics workstations. The *PLOT3D* [14], and *FAST* [15] graphics software packages developed at NASA Ames Research Center are extensively used for this purpose, and data files for these plotting packages are generated automatically. These data files are written in what is known as *PLOT3D* multiple-grid format. (See *ADPAC* File Description, Section 3.5).

2.6 Consolidated Serial/Parallel Code Capability

One of the practical difficulties of performing CFD analyses is finding sufficient computational resources to allow for adequate modeling of complex geometries. Oftentimes, workstations are not large enough, and supercomputers have either long queues, high costs, or both. Clearly, a means of circumventing these difficulties without giving up the flexibility of the CFD code or the complexity of the model would be welcome. One possibility is to write a code which could run in parallel across a number of processors, with each one having only a piece of the problem. Then, a number of lesser machines could be harnessed together to make a virtual supercomputer.

The most likely candidates for creating such a machine are the workstations which are otherwise utilized during the day, but sit idle at night. Tremendous power can be made available at no extra cost. There are also massively parallel computers available on the market designed specifically for such applications. These machines are aiming at order of magnitude improvements over present supercomputers.

The objective behind the development of the consolidated *ADPAC* code described in this manual was to create a platform independent parallel code. The intent was to design a parallel code which looks and feels like a traditional code, capable of running on networks of workstations, on massively parallel computers, or on the traditional supercomputer. User effort was to be minimized by creating simple procedures to migrate a serial problem into the parallel environment and back again.

2.7 Parallelization Strategy

The *ADPAC* code has some innate advantages for parallelization: it is an explicit, multi-block solver with a very flexible implementation of the boundary conditions. This presents two viable options for parallelization: parallelize the internal solver (the “fine-grained” approach), or parallelize only the boundary conditions (the “coarse-grained” approach). The fine-grained approach has the advantage that block size is not limited by processor size. This is the approach frequently taken when writing code for parallel computers which are made up of many small processors. The coarse-grained approach is favored when writing code for clusters of workstations, or other machines with a few large processors. The dilemma is that a parallel *ADPAC* needs to run well on both kinds of machines.

The fine grained approach is especially enticing for explicit solvers. Explicit codes have proven to be the easiest to parallelize because there is little data dependency between points. For a single block explicit solver, fine-grained parallelization is the clear choice. However, with a multi-block solver, the boundary conditions must be parallelized in addition to the interior point solver, and that can add a lot of programming effort. The coarse-grained approach is admittedly easier for multi-block solvers, but what if the blocks are too big for the processors? The simplest answer is to require the user to block out the problem so that it fits on the chosen machine. This satisfies the programmer, but the user is faced with a tedious chore. If the user decides to run on a different machine, then the job may have to be redone. The pain saved by the programmer is passed directly to the user.

A compromise position was reached for the *ADPAC* code. The coarse-grained approach is used, but supplemental tools are provided to automatically generate new grid blocks and boundary conditions for a user-specified topography. In this way, the parallel portions of the code are isolated to a few routines within *ADPAC*, and the user is not unduly burdened with architecture considerations. While the vast majority of the *ADPAC* boundary conditions can be employed in both serial and parallel modes of operation, the user should be cautioned that a few of the boundary conditions will not perform correctly in parallel. This restriction is noted within the sections describing the boundary conditions. Details of running *ADPAC* in parallel are given in Chapter 4.

Chapter 3

ADPAC : 3-D EULER/ NAVIER-STOKES FLOW SOLVER OPERATING INSTRUCTIONS

3.1 Introduction to ADPAC

This chapter contains the operating instructions for the *ADPAC* time-dependent multiple block grid 3-D Euler/Navier-Stokes aerodynamic analysis. These instructions include some general information covering executing the code, defining array limits, compiling the flow solver, setting up input files, running the code, and examining output data. The *ADPAC* flow solver routines are primarily written in FORTRAN and have been compiled under both F77 and F90 for both serial and parallel mode. *ADPAC* has been run successfully on several computing platforms including SGI, Cray, IBM, Sun, HP, DEC, and Pentium-based PC's running under the Linux operating system.

3.2 General Information Concerning the Operation of the ADPAC Code

Approximate computational storage and CPU requirements for the *ADPAC* code can be conservatively estimated from the following formulas:

$$\text{CPU (sec)} \approx 1.1 \times 10^{-4} (\# \text{ grid points})(\# \text{ iterations})$$

$$\text{Memory (Mb)} \approx 2.6 \times 10^{-4} (\# \text{ grid points})$$

These formulas are valid for a 64-bit executable running on a SGI Power Challenge computer (300 MFLOPS/cpu) operating under the IRIX 6.2 environment and the f90 compiler (version 7.0) using the compiler optimization string specified in the standard *ADPAC* Makefile. The times reported are for a single processor only. The formulas are based on the standard, explicit solution algorithm using the algebraic turbulence model. Use of the implicit flow solver or higher-order turbulence model could effectively increase both numbers by a factor of 1.4 or more. Use of parallel processing can substantially reduce these estimates on a per CPU basis.

Without multi-grid, a 250,000 grid-point steady inviscid flow calculation normally requires approximately 2000 iterations to reduce the maximum residual by three orders of magnitude (10^3) which is normally an acceptable level of convergence for most calculations. Viscous flow calculations generally require 3000 or more iterations to converge. When multi-grid is used, the number of iterations required to obtain a converged solution is often one-third to one-fourth the number of iterations required without multi-grid. Convergence for a viscous flow case is generally less well behaved than a corresponding inviscid flow calculation, and in many cases, it is not possible to reduce the maximum residual by three orders of magnitude due to oscillations resulting from vortex shedding, shear layers, etc. A determination of convergence for a viscous flow case must often be based on observing the mass flow rate, pressure ratio, or other global parameter, and terminating the calculation when these variables no longer change. The number of iterations required for an unsteady flow calculation is highly case-dependent, and may be based on mesh spacing, overall time-period, complexity of the flow, etc.

The *ADPAC* program produces output files suitable for plotting using the *PLOT3D* [14] and *FAST* [15] graphics software packages developed at the NASA Ames Research Center. *PLOT3D* multiple-block format data files are written for both absolute and relative flows (see Section 3.11 for a description of the *PLOT3D* format). The user may also elect to have additional *PLOT3D* absolute flow data files output at constant iteration intervals during the course of the solution. These files may be used as instantaneous flow “snapshots” of an unsteady flow prediction. *PLOT3D* formatted files containing turbulence parameters are also generated when using the Spalart-Allmaras model or $k-\mathcal{R}$ model.

3.3 Configuring ADPAC Maximum Array Dimensions

The first step required before attempting to compile and run the *ADPAC* program is to set the maximum array size required for the analysis. The maximum array size will ultimately determine the largest problem (in terms of total number of mesh points) which can be run with the code. The larger the array limits, the larger the number of grid points which may be used. Unfortunately, setting larger array limits also increases the total amount of memory required by the program, and hence, can impede the execution of the code on memory-limited computing systems. Ideally, the code should be dimensioned just large enough to fit the problem at hand. It should be mentioned that storage requirements are dependent on whether the multi-grid convergence acceleration technique is used or not. This dependency is explained in more detail in the paragraphs which follow.

Array dimensions are specified in the *ADPAC* program by a set of FORTRAN `parameter` statements. The array limits are specified in the source code file `parameter.inc` such as shown in Figure 3.1. Each statement in the `parameter.inc` file is ultimately embedded in every subroutine through a FORTRAN `include` statement. During execution, the *ADPAC* program automatically checks to make sure enough storage is available for all the blocks and issues a fatal error message if an array size is exceeded.

Before proceeding with a description of the various parameters, it should be mentioned that a computational tool is available called *ADSTAT* which will read in an *ADPAC* mesh file and determine the required parameter sizes for all possible levels of

```

parameter ( nbmax = 101 )
parameter ( nra3d = 300001 )
parameter ( nraid = 2001 )
parameter ( nbl2d = 50001 )
parameter ( nraint = 1 )
parameter ( nbcdbl = 25 )
parameter ( nbfra = 1 )
parameter ( lgrafx = 1 )
c---> nsyst = dimension of sdum array for system call BC
parameter ( nsyst = 1 )
c---> nbffile = dimension of bffile array for body force files
parameter ( nbffile = 16 )
c---> nbcnt1 = dimension of bcint stencil saving array (set =1 to turn off)
parameter ( nbcnt1 = 1 )
c
c---> the following parameters control 3-D array storage
c
c nimpra sets multi time level implicit array storage (0 = none, 1 = do it)
c n2eqra sets 2 equation turbulence model array storage (0 = none, 1 = do it)
c
parameter ( nimpra = 0 )
parameter ( n2eqra = 0 )

```

Figure 3.1: Sample *ADPAC* parameter file specification (parameter.inc).

multi-grid use. *ADSTAT* will also determine the minimum array sizes needed for parallel execution of *ADPAC* with mesh block split across multiple processors. The *ADSTAT* program is described in more detail in Chapter 7.

The various parameters utilized in the parameter.inc file are described below:

nbmax

The parameter **nbmax** defines the maximum number of grid blocks permitted during execution of the *ADPAC* multiple block solver. This number must be large enough to include every level of coarse mesh blocks created during a multi-grid run. The *ADPAC* code exploits the multiple block mesh structure during multi-grid runs by creating and storing coarse mesh blocks from the corresponding fine mesh blocks. For example, to run a 5 block mesh with 3 levels of multi-grid, the parameter **nbmax** must be at least 15.

nra3d

The parameter **nra3d** defines the maximum total number of computational cells permitted for the finite volume time-marching algorithm. This parameter essentially limits the maximum total number of mesh points (including multi-grid coarse meshes, when applicable) which are permitted during an *ADPAC* run. The minimum value for the **nra3d** parameter for a given mesh system may be calculated as follows:

$$nra3d \geq \left(\sum_{m=1}^{m=NBLKS} [(IMX)_m + 1][(JMX)_m + 1][(KMX)_m + 1] \right) + 1$$

where $(IMX)_m$, $(JMX)_m$, and $(KMX)_m$ indicate the number of mesh points in the i , j , and k mesh coordinate directions, respectively, for mesh block m , and $NBLKS$ is the

total number of grid blocks including all coarse meshes desired from fine meshes when multi-grid is employed. The requirement that the parameter variable **nra3d** (and others) be based on array sizes one element larger than the grid dimensions results from the use of phantom cells outside the computational domain to impose the numerical boundary conditions.

nbl2d

The parameter **nbl2d** is used to define the size of the temporary 2-D arrays utilized during the advancement of the time-marching algorithm for a given mesh block. As such, the parameter is based on the largest single dimension of *any* mesh block (2-D or 3-D) and may be determined by the following formula:

$$nbl2d \geq (\max_{m=1, NBLKS} [(IMX)_m + 1, (JMX)_m + 1, (KMX)_m + 1])^2 + 1$$

where the variables *IMX*, *JMX*, *KMX*, *NBLKS* are defined in the section describing **nra3d** above. This value is unchanged regardless of the number of multi-grid levels since coarser meshes always result in smaller mesh sizes.

nra1d

The parameter **nra1d** is used to define the size of several 1-D arrays used to do various bookkeeping operations during the execution of the *ADPAC* code. As such, the parameter is based on the sum of the maximum single dimension of all mesh blocks in the following manner:

$$nra1d \geq \left(\sum_{m=1}^{m=NBLKS} \max[(IMX)_m + 1, (JMX)_m + 1, (KMX)_m + 1] \right) + 1$$

nbcpbl

The parameter **nbcpbl** is used to define the size of the arrays used to store the boundary condition specifications for a given *ADPAC* run. Since the number of boundary conditions normally scales according to the number of mesh blocks (as a minimum, six boundary conditions are required for each 3-D mesh block, see Section 3.7), the parameter **nbcpbl** implies the maximum number of boundary conditions per block, and the overall number of boundary conditions is determined by multiplying the parameters **nbmax** and **nbcpbl**. It should be noted that a single block can, in fact, possess more than **nbcpbl** boundary condition specifications as long as the *total* number of boundary condition specifications for the entire problem does not exceed **nbmax** × **nbcpbl**.

nraint

The parameter **nraint** is used to define the size of the temporary arrays used to store interpolation data for the non-contiguous mesh patching boundary condition specification **PINT**, described in Section 3.7. The **PINT** specification controls the numerical coupling between two mesh blocks possessing non-contiguous mesh boundaries which lie on a common surface. The numerical scheme utilizes a rather simple interpolation scheme based on an electrical circuit analogy, and stores the “nearest neighbors” for each mesh point to avoid the expense of constantly searching for the interpolation stencil between the two mesh surfaces. Determining the value required for the parameter **nraint** is normally

performed by summing up all of the mesh elements involved in all of the **PINT** specifications (including coarse mesh specifications from a multi-grid run).

nbfra

The parameter **nbfra** is used to define the size of the 2-D arrays used to store the blade element blockage, body force, and energy source terms for the 2-D block solution scheme. Since these arrays are utilized for any 2-D mesh block regardless of whether blade element blockage and source terms are utilized, the arrays must be dimensioned large enough to store all the elements of all of the 2-D mesh blocks (including coarse meshes for multi-grid runs) much in the manner that **nra3d** is used to store all of the elements of all of the mesh blocks. Mathematically, the minimum value for the parameter **nbfra** may be calculated as:

$$nbfra \geq \left(\sum_{m=1}^{m=NBLKS} [(IMX)_m + 1][(JMX)_m + 1]L_{2D}(m) \right) + 1$$

where the variables IMX , JMX , KMX and $NBLKS$ are described in the definition of parameter **nra3d**, above. The variable $L_{2D}(m)$ is a trigger to indicate whether the grid block m is 2-D (1) or 3-D (0).

lgrafx

The parameter **lgrafx** is used to define the size of the temporary 3-D arrays used for the run-time graphics display option available in the *ADPAC* code. If the run-time graphics option is employed, then the parameter **lgrafx** can be determined in the same manner as the parameter **nra3d**. If the run-time graphics option is not employed, then the parameter **lgrafx** should be set to 1, resulting in a considerable savings in computational storage.

nsyst

The parameter **nsyst** is used to define the size of a character array which stores system call commands during the execution of the boundary condition routine **SYSTEM** (see Section 3.7). Normally, this is not used and may be set to a value of 1 to minimize storage. If the **SYSTEM** boundary routine is used, then **nsyst** must be at least as large as the number of **SYSTEM** boundary specifications in the *ADPAC* boundary data file.

nbffile

The parameter **nbffile** is used to define the size of a character array which stores body force file names specified by the input variable **BFFILE** (see Section 3.6). Normally, this is not used and may be set to a value of 1 to minimize storage. If the **BFFILE** input variable is used, then **nbffile** must be at least as large as **nbmax**.

nbcnt1

The parameter **nbcnt1** is used to define the size of the arrays used to save the interpolation stencils used in the **BCINT1** and **BCINTM** non-aligned mesh boundary coupling schemes. In an effort to increase computational and communication efficiency, the interpolation stencils used to update the non-aligned boundaries in these boundary condition routines are only calculated on the first step, and are subsequently saved to eliminate any redundant calculation. The **nbcnt1** parameter must be at least as large as the sum of the total number of points along all **BCINT1** and **BCINTM** non-aligned boundary patches. If **BCNT1** is set to 1, then the interpolation stencil saving feature is disabled, and the interpolation stencil is recalculated at every time step.

nimpra

The parameter **nimpra** is used to define the size of the arrays used in the *ADPAC* implicit solution algorithm. For time-dependent solutions involving the iterative implicit solution algorithm, up to two additional time levels of the conserved flow variables must be stored, and this storage is defined based on the value of $(\mathbf{nimpra} \times \mathbf{nra3d} + 1)$. The value of **nimpra** should therefore be either 0 (no implicit time level storage) or 1 (provide implicit time level storage). If an implicit solution is attempted when the code has been compiled with **nimpra**=0, the code prints an error message and halts execution.

n2eqra

The parameter **n2eqra** is used to define the size of the arrays used in the *ADPAC* one-equation or two-equation turbulence model solution algorithm. Additional storage is required for the dependent variables employed in the solution of the turbulence transport equations, and this storage is defined based on the value of $(\mathbf{n2eqra} \times \mathbf{nra3d} + 1)$. The value of **n2eqra** should therefore be either 0 (no one-equation or two-equation turbulence model storage) or 1 (provide one-equation or two-equation turbulence model storage). If an one-equation or two-equation turbulence model solution is attempted when the code has been compiled with **n2eqra**=0, the code prints an error message and halts execution.

3.4 ADPAC Compilation Using Makefile

Compilation of the *ADPAC* source code into an executable form is handled through a UNIX-based Makefile facility. A Makefile is included with the standard distribution which permits automatic compilation of the code for several operational capabilities (both serial and parallel) and computer systems. A sample UNIX shell script illustrating the basic steps required for compiling and running the *ADPAC* code in serial and in parallel is given in Appendix B. The format of the Makefile compiling command is described below.

Several items should be mentioned prior to detailed discussion on the actual Makefile utilities. Section 3.8 describes the format of the binary files using the Scientific Database Library developed at NASA-Lewis [16, 17]. The original (FORTRAN) version of the Scientific Database Library was found to be rather slow on some machines, and an equivalent limited capability C-based library was developed to accelerate the I/O processing in the code. This library is referred to as CSDB, and separate options for utilizing the CSDB library are included in the Makefile. In addition, the consolidated code is capable of both serial and parallel operation depending on the Makefile operation selected.

In the directory containing the FORTRAN source of the *ADPAC* code, compilation is performed by executing the command:

`make option`

The `make` command is standard on UNIX systems and automatically interrogates the file Makefile for instructions on how to perform the compilation. At the completion of the compilation process on any system, an executable version of the code is written in the source directory. The *option* argument may be any of the variables listed below:

Standard UNIX (Silicon Graphics) Make Options

- (No argument) This is the standard UNIX system compilation for the serial version of the *ADPAC* code. All non-standard programming constructs are avoided (such as graphics, or multi-processor features). This option will deliver a working executable on most UNIX systems which support standard naming conventions (f77 as the standard compiler, etc.). The compilation includes basic compiler optimization (f77 -O). Executable name: *adpac*.
- csdb* This is the same as above, except that the faster C-based scientific database library is linked instead of the standard scientific database library. Prior to performing this compilation, the appropriate make command must be issued in the CSDB directory to assemble the CSDB library for the local machine. Executable name: *adpac*.
- graphics* This option compiles *ADPAC* with the necessary routines needed to permit interactive graphics between network connected Silicon Graphics workstations. This option will only work when compiling on a Silicon Graphics workstation with IRIX operating system 4.0.1 or above. The full Silicon Graphics shared graphics libraries and X-windows system graphics libraries must be installed on the compiling workstation in order for this option to work. This feature is not recommended as it generally decreases performance and other visualization techniques are likely to produce more desirable results. Executable name: *adpac_graphics*.
- csdb_graphics* This option is the same as *graphics* above, except that the faster C-based scientific database library is linked instead of the standard scientific database library. See compiling notes under *graphics* and *csdb*. Executable name: *adpac_graphics*.
- dbx* This option is used for generating an executable version of the serial code which is compatible with the standard UNIX *dbx*-based debugging facility. This should work on any standard UNIX machine which supports *dbx* (Note: the code will run much more slowly when compiled in this fashion.) This option is used mainly for code development or debugging. Executable name: *adpac_dbx*.
- csdb_dbx* This option is the same as *dbx* above, except using the CSDB library. See compiling notes under *dbx* and *csdb*. Executable name: *adpac_dbx*.
- dbx_graphics* This option is used for generating an executable version of the serial code with run-time graphics enabled which is compatible with the standard UNIX *dbx*-based debugging facility using the CSDB library. See compiling notes under *graphics* and *dbx*. Executable name: *adpac_dbx_graphics*.
- parallel* This is the standard UNIX system compilation for the parallel version of the *ADPAC* code. The standard APPL message passing

library is incorporated, and therefore creation of this executable requires that a make has been issued in the APPL directory on the current machine. The *parallel* code may only be executed using the APPL compute function with a corresponding APPL procdef file. Prior to performing this compilation, the appropriate *make* command must be issued in the APPL directory to assemble the APPL library for the local machine. Executable name: `adpacp`.

- `parallel_csdb` This is the same as `parallel` above, except that the faster C-based scientific database library is linked instead of the standard scientific database library. See compiling notes under `parallel` and `csdb`. Executable name: `adpacp`.
- `parallel_dbx` This is the same as `parallel` above, except that specific compiler options have been enabled to utilize the UNIX *dbx* debugging facility. See compiling notes under `parallel` and `dbx`. Executable name: `adpacp`.
- `parallel_csdb_dbx` This is the same as `parallel_dbx` above, except that the faster C-based scientific database library is linked instead of the standard scientific database library. See compiling notes under `parallel_dbx` and `csdb`. Executable name: `adpacp_dbx`.
- `parallel_pvm` This is the standard UNIX system compilation for the `parallel` version of the *ADPAC* code using the PVM message passing library. The standard APPL message passing library is incorporated as a sublayer between the native PVM calls and the *ADPAC* programming message-passing calls, and therefore creation of this executable requires that a make has been issued in the APPL directory on the current machine. The *parallel* code may only be executed using the APPL compute function with a corresponding APPL procdef file. Prior to performing this compilation, the appropriate *make* command must be issued in the APPL directory to assemble the APPL library for the local machine. Executable name: `adpacp_pvm`.

Silicon Graphics Power Challenge Make Options

- `power_challenge` This option is utilized when compiling the standard serial code on a Silicon Graphics R8000 or R10000 Indigo² or Power Challenge computer. For best performance, several machine specific optimizations are enabled. Executable name: `adpac_power_challenge`.
- `power_challenge_mpich` This option is utilized when compiling the parallel code on a Silicon Graphics R8000 or R10000 Indigo² or Power Challenge computer using the public domain MPI library MPICH. Executable name: `adpac_power_challenge_mpich`.
- `power_challenge_mpi` This option is utilized when compiling the parallel code on a Silicon Graphics R8000 or R10000 Indigo² or Power Challenge computer using the SGI version of MPI. Executable name: `adpac_power_challenge_mpi`.

HP Workstations Make Options

-
- hp This option is utilized when compiling the standard serial code on a Hewlett-Packard computer running OS 10.0 or greater. For best performance, several machine specific optimizations are enabled. Executable name: `adpac_hp`.
 - hp_mpich This option is utilized when compiling the parallel code on a Hewlett-Packard computer using the public domain MPI library MPICH. Executable name: `adpac_hp_mpich`.
 - hp_mpi This option is utilized when compiling the parallel code on a Hewlett-Packard computer using the HP version of MPI. Executable name: `adpac_hp_mpi`.

Sun Workstations Make Options

- sun This option is utilized when compiling the standard serial code on a Sun Microsystems computer. For best performance, several machine specific optimizations are enabled. Executable name: `adpac_sun`.
- sun_mpich This option is utilized when compiling the parallel code on a Sun Microsystems computer using the public domain MPI library MPICH. Executable name: `adpac_sun_mpich`.
- sun_mpi This option is utilized when compiling the parallel code on a Sun Microsystems computer using the Sun version of MPI. Executable name: `adpac_sun_mpi`.

Cray (UNICOS) Make Options

- cray This option is utilized when compiling the standard code on a Cray computer (implies a serial code). For best performance, the aggressive optimization option of the Cray compiler has been invoked. Executable name: `adpac_cray`.
- cray_dbx This option is used for generating an executable version of the code which is compatible with the Cray *dbx* debugging facility. See compiling notes under `cray` and `dbx`. Executable name: `adpac_cray_dbx`.

IBM AIX Make Options

- aix This option is used when compiling the standard serial code on an IBM RS-6000 workstation running the AIX operating system. Executable name: `adpac_aix`.
- aix_csdb This option is identical to `aix` above, except that the faster C-based scientific database library is used. See compiling notes under `aix` and `csdb`. Executable name: `adpac_aix`.
- aix_dbx This option is used for generating an executable version of the code which is compatible with the IBM AIX *dbx* debugging facility. See compiling notes under `aix` and `dbx`. Executable name: `adpac_aix_dbx`.
- aix_csdb_dbx This option is identical to `aix_dbx` above, except that the faster

C-based scientific database library is used. See compiling notes under `aix`, `csdb`, and `dbx`. Executable name: `adpac_aix_dbx`.

`aix_parallel` This is the standard IBM RS-6000 AIX UNIX system compilation for the parallel version of the *ADPAC* code. See compiling notes under `aix` and `parallel`. Executable name: `adpacp_aix`.

`aix_parallel_csdb` This option is identical to `aix_parallel` above, except that the faster C-based scientific database library is used. See compiling notes under `aix_parallel` and `csdb`. Executable name: `adpacp_aix`.

`aix_parallel_dbx` This option is used for generating an executable parallel version of the code which is compatible with the IBM AIX *dbx* debugging facility. See compiling notes under `aix_parallel` and `dbx`. Executable name: `adpacp_aix_dbx`.

Linux Make Options

`linux` This option is utilized when compiling the standard serial code under a computer running under the Linux operating system. Executable name: `adpac_linux`.

`linux_csdb` This option is the same as `linux` except that the faster C-based scientific database library is used. Executable name: `adpac_linux`.

`linux_mpich` This option is utilized when compiling the parallel code under Linux using the public domain MPI library MPICH. See the compiling notes under `linux` and `adpac_mpich`. Executable name: `adpac_linux_mpich`.

`linux_csdb_mpich` This option is the same as `linux_mpich` except that the faster C-based scientific database library is used. See the compiling notes under `linux`, `csdb`, and `adpac_mpich`. Executable name: `adpac_linux_mpich`.

nCUBE2 Vertex Operating System Make Options

`ncube` This is the standard nCUBE 2 system compilation for the parallel version of the *ADPAC* code. The nCUBE version of the APPL message passing library is incorporated. See compiling notes under `parallel`. Executable name: `adpacp_ncube`.

`ncube_csdb` This option is identical to `ncube` above, except that the faster C-based scientific database library is linked instead of the standard scientific database library. See compiling notes under `ncube` and `csdb`. Executable name: `adpacp_ncube`.

`ncube_dbx` This option is used when compiling the parallel code for the nCUBE parallel computer using the VERTEX operating system and the nCUBE *dbx* debugging facility. See compiling notes under `ncube` and `dbx`. Executable name: `adpacp_ncube_dbx`.

`ncube_csdb_dbx` This option is the same as `ncube_dbx` above, except that the faster C-based scientific database library. See compiling notes under `ncube`, `dbx`, and `csdb`. Executable name: `adpacp_ncube_dbx`.

`ncube_csdb_no2d` This option is used when compiling the parallel code for the nCUBE parallel computer using the VERTEX operating system and the faster C-based scientific database library. This option also eliminates all 2-D subroutines to minimize the size of the executable, which is useful for strictly 3-D problems on multiprocessing computers with limited memory per processor. See compiling notes under `ncube` and `csdb`. Executable name: `adpacp_ncube_no2d`.

NASA-Lewis Research Center Specific Make Options

`lace` This option is used when compiling the standard parallel code for the NASA-Lewis LACE workstation cluster computing environment. Executable name: `adpacp_lace`.

`lace_csdb` This option is identical to `lace` above, except that the faster C-based scientific database library is used. See compiling notes under `lace` and `csdb`. Executable name: `adpacp_lace`.

Source Directory Maintenance Make Options

`help` This option lists and describes all available Makefile options. No executable is created.

`clean` This option cleans up the source directory by removing all object files. No executable is created.

`realclean` This option really cleans up the source directory by removing all object and library files. No executable is created.

`onesource` This option concatenates all the *ADPAC* source files into a single source file name `adpac.onesource.f`. This single source can then be compiled by hand on those machines for which an appropriate make option is not available. It is up to the user to link in the necessary library files (CSDB, APPL, etc) for creation of an executable. No executable is created.

3.5 ADPAC Input/Output Files

In this section, the various input/output data files related to a calculation using the *ADPAC* program are described. In order to understand the file naming convention, the concept of a case name must first be detailed. All files used in an *ADPAC* calculation are named according to a standard naming convention of the form:

case.extension

where *case* is a unique, user-specifiable name usually identifying the geometry or flow condition being investigated, and *extension* is a name describing the type of file. The *case* name must be specified in the standard input file described below. A list and description of each of the files used or generated by *ADPAC* is given in Table 3.1.

Table 3.1: Description of input/output files and UNIX-based filenames for *ADPAC* .

Name	Description
<i>case.input</i>	Standard input file
<i>case.boundata</i>	Block boundary definition file
<i>case.output</i>	Standard output file (from output redirection only)
<i>case.mesh</i>	Mesh file (<i>PLOT3D</i> compatible)
<i>case.p3dabs</i>	Final <i>PLOT3D</i> output file (absolute flow)
<i>case.p3drel</i>	Final <i>PLOT3D</i> output file (relative flow)
<i>case.p3d1eq</i>	Final <i>PLOT3D</i> output file (1-eq. turbulence model data)
<i>case.p3d2eq</i>	Final <i>PLOT3D</i> output file (2-eq. turbulence model data)
<i>case.bf.#</i>	2-D blockage/body force file for block number #
<i>case.p3fr.#</i>	Instantaneous <i>PLOT3D</i> interval output file (absolute flow). The frame number is given by #.
<i>case.img.#</i>	Instantaneous Silicon Graphics image file for graphics interactive display. The frame number is given by #.
<i>case.restart.new</i>	New restart file (output by code)
<i>case.restart.old</i>	Old restart file (used as input for restart runs)
<i>case.restart_SA.new</i>	New 1-eq. turbulence model restart file (output by code)
<i>case.restart_SA.old</i>	Old 1-eq. turbulence model restart file (used as input for restart runs)
<i>case.restart_KR.new</i>	New 2-eq. turbulence model restart file (output by code)
<i>case.restart_KR.old</i>	Old 2-eq. turbulence model restart file (used as input for restart runs)
<i>case.converge</i>	Solution residual convergence history file
<i>case.converge_SA</i>	1-eq. turbulence model residual convergence history file
<i>case.converge_KR</i>	2-eq. turbulence model residual convergence history file
<i>case.sixpac</i>	<i>SIXPAC</i> block subdivision file (parallel only)
<i>Ncase.bacpac</i>	<i>BACPAC</i> block reconstruction file (parallel only)
<i>case.blkproc</i>	<i>ADPAC</i> block/processor assignment file (parallel only)
<i>procdef</i>	APPL process description file (APPL parallel only)
<i>case.axi.mesh</i>	Equivalent axisymmetric mesh output by body force calculation
<i>case.axi.restart.new</i>	Equivalent axisymmetric flow restart output by body force calculation

The standard input, standard output, boundary data, and convergence history files are stored in ASCII format. All other files utilize the Scientific DataBase Library (*SDBLIB*) [16, 17] format. The mesh file and *PLOT3D* plot output files are compatible with the *PLOT3D* multiple grid, binary definition (see Sections 3.8 and 3.11 for a description and coding examples of the *SDBLIB* binary format). Files dealing with the parallel execution of the *ADPAC* code are described in Chapter 4.

The standard input and standard output files are directed at runtime using the standard UNIX redirection syntax as:

$$adpac_executable_name < case.input > case.output$$

If a restart run is desired, the user must move the most current output restart file from:

$$case.restart.new$$

to the default input restart file name:

$$case.restart.old$$

each time the code is restarted. A more detailed description of the use and format of the *ADPAC* files is presented in the sections which follow.

3.6 ADPAC Standard Input File Description

The standard *ADPAC* input file *case.input* contains the user-specified parameters which control the basic operation of the code during execution. During code execution, the input file is read one line at a time as a character string, and each string is parsed sequentially to determine the specific program action in each case. The standard input file utilizes a keyword input format, such that any line which does not contain a recognizable keyword is treated as a comment line. Therefore, the user may place any number of comments in the file (so long as the line does not contain a keyword input string in the form described below), and code execution is unaltered. Comments may also be placed after the variable assigned to the keyword as long as there are one or more blanks separating the keyword value from the comment string. All input file lines are echoed to the standard output, and the program response to each line is listed when a specific action is taken.

All keyword input lines are given in the following format:

$$\text{KEYWORD} = \text{Value} \quad \text{Comment String}$$

where **KEYWORD** is one of the recognized keywords described below, and **Value** is the specific value to be assigned to that variable. The input line must contain the equals sign (=) with one or more blanks on *both* sides in order to be recognized. The **Comment String** must be separated by one or more blank spaces from the **Value**. Therefore, the lines:

```
DIAM    = 10.000
DIAM                                =                10.000
DIAM    = 10.000    This is the diameter.
```

are valid keyword input lines assigning the value 10.0 to the variable associated with the keyword **DIAM**. Conversely, the lines:

```
DIAM= 10.000
DIAM =10.000
DIAM=10.000
```

are not recognizable keyword input lines (in spite of the presence of the keyword **DIAM**), because of the lack of proper placement of the blanks about the equals sign (=). The purpose for this restriction is to permit keyword variables in comment lines, and to help users to generate readable input files. All keyword values are either real numbers (which, in many cases, are converted to integers in the code) or character strings. A sample *ADPAC* standard input file containing a number of typical use keywords is listed in Figure 3.2.

It is unnecessary to specify all possible keywords in every input file. The *ADPAC* code is programmed with a default set of input variables such that the only input variable which must be present is the **CASENAME** (described below) which is used to assign input/output file names. A list and description of all input keywords and their default values are listed below.

ADPAC Input File

NASA 1.15 PRESSURE RATIO FAN - 2 BLADE ROWS

VARNAME	=	VARIABLE VALUE	COMMENT
CASENAME	=	nasa	The case name is "nasa"
FMULTI	=	3.0	Three mesh levels for multi-grid
FSUBIT	=	1.0	1 subiteration on each coarse mesh level
FFULMG	=	1.0	Use "full" multi-grid
FCOAG1	=	3.0	Start "full" multi-grid on 3rd mesh level
FCOAG2	=	2.0	End "full" multi-grid on 2nd mesh level
FITFMG	=	150.0	150 "full" multi-grid iterations per level
RMACH	=	0.750000	Reference Mach Number for initialization
FINVVI	=	0.000000	0.0-Inviscid Flow, 1.0-viscous flow
GAMMA	=	1.400000	Specific heat ratio
PREF	=	2116.220000	Reference Total Pressure (lbf/ft**2)
TREF	=	518.670000	Reference Total Temperature (Deg. R)
RGAS	=	1716.260010	Gas constant (ft-lbf/slug-deg R)
DIAM	=	9.000000	Reference diameter (ft.)
EPSX	=	1.000000	Residual smoothing coefficient in i direction
EPSY	=	1.000000	Residual smoothing coefficient in j direction
EPSZ	=	1.000000	Residual smoothing coefficient in k direction
VIS2	=	0.500000	Fine mesh 2nd order dissipation coefficient
VIS4	=	0.015625	Fine mesh 4th order dissipation coefficient
VISCG2	=	0.125000	Coarse mesh dissipation coefficient
CFL	=	-5.000000	<0.0, Steady flow, >0.0, unsteady flow
FNCMAX	=	150.000000	150 iterations on fine mesh level
PRNO	=	0.700000	Gas Prandtl number = 0.7
PRTNO	=	0.900000	Turbulent Prandtl number = 0.9
FREST	=	0.000000	No restart file is read in
RPM(1)	=	-1000.000000	Rotational speed for block #1 is -1000.00 RPM
RPM(2)	=	-1000.000000	Rotational speed for block #2 is -1000.00 RPM
RPM(3)	=	0.000000	Rotational speed for block #3 is 0.00 RPM
RPM(4)	=	0.000000	Rotational speed for block #4 is 0.00 RPM
ENDINPUT			

Figure 3.2: Sample ADPAC input file specification (case.input).

ADPAC Standard Input File Keyword Description

ADVR(*num*) [0.0]

The **ADVR** keyword value determines the rotational speed (in terms of an advance ratio) of the mesh block number specified by the value *num*. By default, block rotational speeds are zero unless either a **RPM** or an **ADVR** keyword are specified. The advance ratio is inherently tied to the freestream Mach number specified in the value associated with the keyword **RMACH**. If the mesh has not been correctly non-dimensionalized, or if the value of **RMACH** is incorrect, it is possible that an incorrect value of rotational speed would be specified in the calculation. The use of **ADVR** is normally only employed for propeller performance calculations.

BFFILE(*num*) [default_file_name]

The **BFFILE** keyword value determines the name of the file used to read in the data for the blade blockage and body force source terms used to represent the effects of embedded blade rows in 2-D axisymmetric flow calculations. The file specified by **BFFILE** is used to describe the terms for the block number indicated by the value of *num*. Body force data files created by the *ADPAC* program are named according to the file naming convention described in Section 3.5.

CASENAME

The **CASENAME** keyword value is used to set the case name which is used to define all input/output file names during an *ADPAC* run (see Section 3.5 for details). The case name is limited to an 80 character string, and cannot contain embedded blanks. The case name has no default value, and as such, *all* input files *must* contain the **CASENAME** keyword.

CCP [1.6]

The **CCP** keyword sets one of the two variable coefficients in the algebraic Baldwin-Lomax turbulence model; the other user-defined coefficient is **CKLEB**. Based on a sensitivity analysis by Granville [18], the option to modify the coefficients in the standard Baldwin-Lomax turbulence model was added to *ADPAC*. The standard value for this coefficient from the original Baldwin-Lomax model is $C_{cp} = 1.6$. The variation of these model coefficients with respect to pressure gradient is shown in Figure 3.3; the values for C_{cp} are read off the left-hand y-axis and range from 1.0 to 1.80. The plot shows regions for both favorable and adverse pressure gradients, such that the values of the coefficients can be chosen properly for either compressor or turbine applications.

CFL [-5.0]

The **CFL** keyword defines the value of the time step multiplier used in the time-marching solver. The algorithm is sensitive to the sign of the value used for **CFL** in determining the manner in which the time-marching solver is applied. If **CFL** < 0.0, local time stepping is used (steady flow only) and each cell is advanced in time according to the local maximum allowable time step. If **CFL** > 0.0, then a time-accurate time-marching solution is performed using an explicit Runge-Kutta algorithm where the time step is based on the value of $|\mathbf{CFL}| \times (\Delta t)_{min}$ and $(\Delta t)_{min}$ is the minimum of all local time steps. The absolute value of **CFL** is used as a multiplier for the time step (larger absolute values indicate larger time steps). A value of -5.0 is normally used for steady flow calculations,

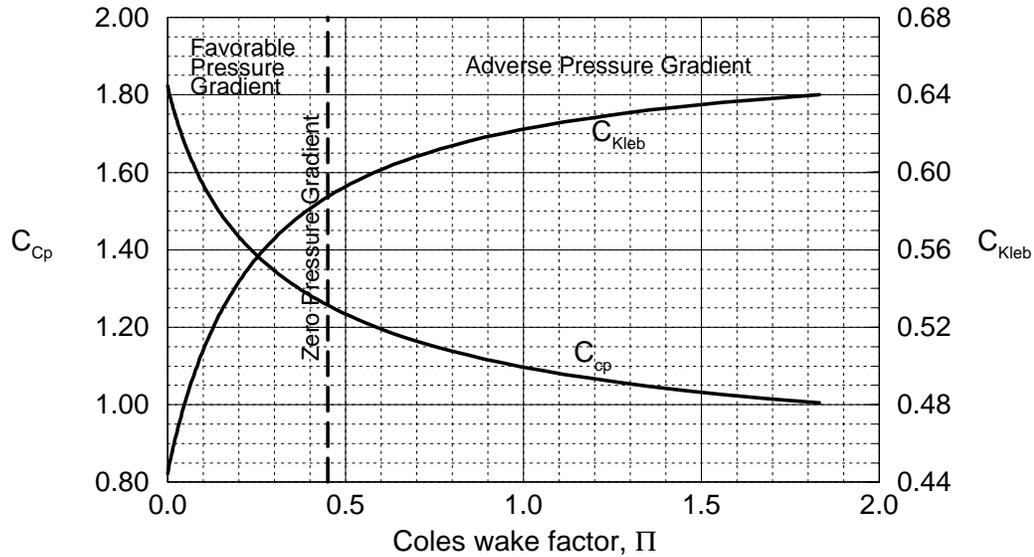


Figure 3.3: Variation of C_{cp} and C_{Kleb} with Coles wake factor (Π).

and values as high as 7.0 have been used successfully for time-accurate calculations. The value of **CFL** is also used implicitly in the eigenvalue scaling terms in the implicit residual smoothing algorithm, such that larger values of **CFL** imply increased residual smoothing (see the description of the implicit residual smoothing algorithm in Appendix A and the description of **CFMAX**).

When **FIMPLIC** = 1.0, the code uses another implicit techniques and the global time step is set by the input keyword **FDELAT**. In this case, the value of **CFL** now controls the “pseudo” time step used during the inner iteration strategy at each global time step. In this case, each global time step can be viewed as a steady state solution, and the variables **CFL**, **CFMAX**, **FMULTI**, etc. retain their meanings in this context, and should be set as in any steady state solution. The implicit time-dependent solution (and global time-dependent iteration strategy) is controlled by the variables **FIMPLIC**, **FDELAT**, **FNTSTEP**, **FMGTSTEP**, **FIMPAC**, and **FTIMFAC**.

CFMAX [2.5 (four-stage scheme), 3.5 (five-stage scheme)]

The **CFMAX** variable is used to define the maximum allowable time step multiplier for the explicit time-marching scheme without residual smoothing. This value is used in the implicit residual smoothing routine to adjust the smoothing coefficients for variations in time steps (see Appendix A). Normally referred to as a *CFL* number, the variable **CFMAX** represents the maximum allowable *CFL* number for the time-marching scheme without residual smoothing, while the variable **CFL** represents the actual *CFL* number used in the calculation with residual smoothing. The ratio of **CFL** to **CFMAX** is used to adjust the amount of smoothing in the residual smoothing operator. Increasing **CFMAX** decreases the magnitude of the residual smoothing coefficients and therefore decreases the overall smoothing. Based on linear stability analysis, the four-stage Runge-Kutta time-marching scheme (**FSOLVE** = 1.0) permits a maximum *CFL* number of $2\sqrt{2}$. For

simplicity, this value is normally approximated as 2.5 which provides an additional margin of safety. Under certain circumstances, it may be desirable to reduce **CFMAX** as low as 2.0 to aid convergence by artificially increasing the amount of residual smoothing. For the five-stage scheme (**FSOLVE** = 2.0) values of 3.0 to 3.5 are recommended.

CKLEB [0.3]

The **CKLEB** keyword sets one of the two variable coefficients in the algebraic Baldwin-Lomax turbulence model; the other user-defined coefficient is **CCP**. Based on a sensitivity analysis by Granville [18], the option to modify the coefficients in the standard Baldwin-Lomax turbulence model was added to *ADPAC*. The standard value for this coefficient from the original Baldwin-Lomax model is $C_{Kleb} = 0.3$. The variation of these model coefficients with respect to pressure gradient is shown in Figure 3.3; the values for C_{Kleb} are read off the right-hand y-axis and range from 0.44 to 0.64. The plot shows regions for both favorable and adverse pressure gradients, such that the values of the coefficients can be chosen properly for either compressor or turbine applications. While it is recognized that the default value of **CKLEB** (0.3) from the original model development does *not* necessarily correlate with the default value of **CCP** (1.6), it does lie in the favorable pressure gradient region and it is left to the user to be aware of the selection of this turbulence model coefficient.

CMUTPS, CMUTSS [14.0]

The **CMUTSS** and **CMUTPS** keywords determine the ratio of local turbulent to laminar viscosity required to initiate transition for the point transition model in the *ADPAC* body centered mesh algebraic turbulence model activated by the keyword **FTURBCHT**. This simplified transition model maintains laminar flow until the ratio of near wall turbulent viscosity to near wall laminar viscosity exceeds the value of **CMUTSS** or **CMUTPS** for the “suction side” and “pressure side”, respectively, of the airfoil in question. The transition model parameters are illustrated in Figure 3.4. A ratio of 14.0 is recommended for all cases unless specific testing has indicated an alternate value. These input variables are only valid for C-type meshes about airfoils. It should be noted that these variables will have no effect when **FINVVI**=0.0 (inviscid flow), or when either **F1EQ**=1.0 or **F2EQ**=1.0 (one- or two-equation turbulence model enabled) or when **FTURBCHT**=0.0 (transition model not activated).

DIAM [1.0]

The **DIAM** keyword is used as a dimensionalizing length scale for the mesh system for a given case. The *ADPAC* code assumes that the mesh has been generated in a nondimensional fashion, and must be dimensionalized before execution. The value of the **DIAM** variable is used to convert the supposed nondimensional mesh coordinates into dimensional coordinates with units of feet. In other words, if the mesh has been generated using a length scale of inches, then the value of **DIAM** should be $\frac{1}{12}$, or 0.083333 in order to convert the mesh units to units of feet. If the mesh units are already in feet, then the value of **DIAM** should be simply 1.0. Many mesh generation systems for turbomachinery geometries nondimensionalize the mesh by a reference diameter determined from the turbomachinery geometry such that the maximum value of any radial coordinate in the mesh is 0.5. In this case, the value of **DIAM** should be the diameter of the turbomachine in feet. Proper specification of the **DIAM** value is critical to achieve the correct flow

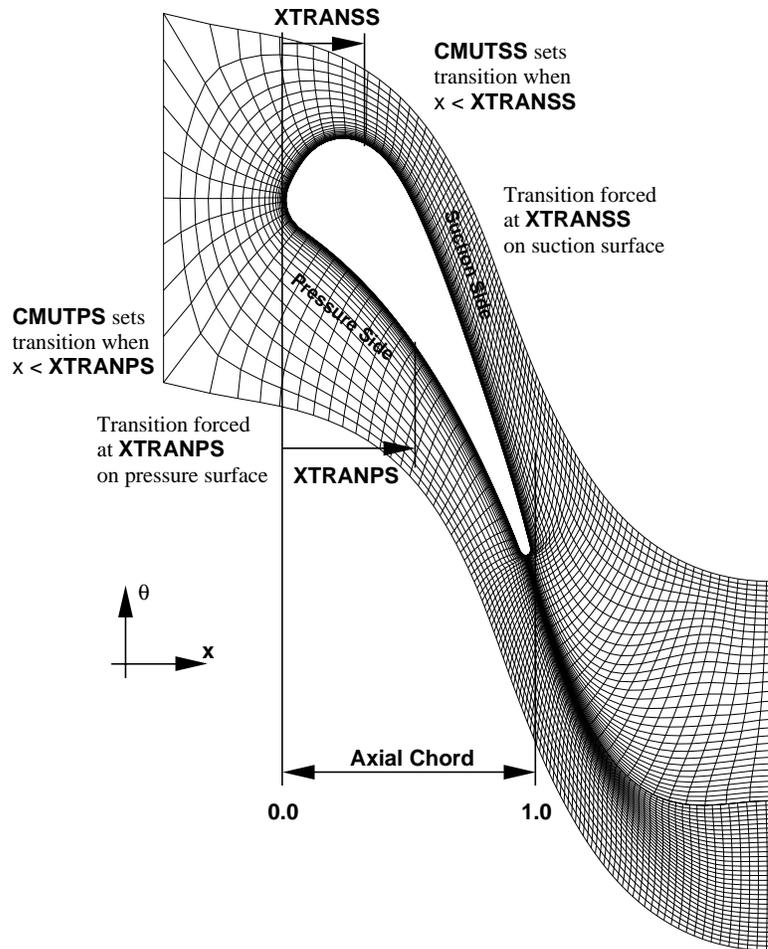


Figure 3.4: ADPAC body-centered mesh turbulence model nomenclature summary.

Reynolds number and tip speed for rotating geometries. Many problems can be traced to improper specification of the **DIAM** value and the user should take care to understand the use of this keyword. When in doubt, the user should remember the simple rule that the actual mesh units, when multiplied by the value of **DIAM** should result in physical lengths expressed in feet.

ENDINPUT

When the *ADPAC* program encounters the keyword **ENDINPUT** no additional input file lines are parsed for input keyword values. Any lines following the **ENDINPUT** statement are ignored, except when the graphics display system is in effect across a network, in which case the statements following the **ENDINPUT** statement must contain two blank lines and the Internet network address of the destination display device (see Chapter 5).

EPSTOT [0.1]

The **EPSTOT** keyword determines the value of the smoothing coefficient employed in the post multi-grid smoothing algorithm described by the trigger **FTOTSM**. This coefficient is only used when **FTOTSM** = 1.0. The value of the coefficient may be any positive number, but for most circumstances, a value between 0.0 and 0.25 is suggested (larger values imply more smoothing). This option is normally employed for enhanced code stability during the multi-grid solution process (**FMULTI** > 1).

EPSX, EPSY, EPSZ [1.0]

The **EPSX**, **EPSY**, and **EPSZ** keywords set the value of the implicit residual smoothing coefficient multipliers in the *i*, *j*, and *k* coordinate directions, respectively. The values of **EPSX**, **EPSY**, and **EPSZ** are used as simple multipliers for the residual smoothing coefficients calculated by the eigenvalue scaled residual smoothing scheme described in Appendix A. If **EPSX**, **EPSY**, or **EPSZ** = 0.0, then no smoothing is applied for the given coordinate direction. The user should be aware that the keyword variable **FRESID** controls the global application of residual smoothing in the solution algorithm, and in the case where **FRESID**=0.0 (residual smoothing disabled), the **EPSX**, **EPSY**, and **EPSZ** have no impact on the solution. The default value for the coefficient multipliers is 1.0. Any value larger than 1.0 simply implies excess smoothing and may be useful for cases with poor convergence or undesirable mesh quality. If a value larger than 3.0 is required to stabilize a solution, this generally indicates some sort of problem in the calculation (i.e., poor mesh aspect ratio, bad boundary specification) or it might suggest that **FRESID** has been set inadvertently to 0.0. Values less than 1.0 will likely cause code instabilities for values of **|CFL|** greater than 2.0. Occasionally for cylindrical coordinate system solutions involving a centerline or sting with very small radii, the value of **EPSX**, **EPSY**, or **EPSZ** which corresponds to the “radial” direction must be reduced to 0.25-0.5 to maintain stability. For these cases, increasing the sting radius is likely to yield a more robust model.

F1EQ [0.0]

The **F1EQ** keyword assigns a trigger which determines the activation of the one-equation Spalart-Allmaras turbulence model. This turbulence model can provide superior prediction of turbulent flows particularly in free shear layers and/or separated flow regions. The one-equation model utilizes additional specification of data at inflow boundaries (see e.g. **INLETT**, **INLETG**) to prescribe the “freestream” turbulence level, and the turbulent

viscosity is updated through the solution of one additional transport equation (see Appendix A). The one-equation model is *not* enabled when **F1EQ**=0, and *is* enabled when **F1EQ**=1.0. Setting **F1EQ**=1.0 will disable all other turbulence models in *ADPAC*.

F2EQ [0.0]

The **F2EQ** keyword assigns a trigger which determines the activation of the two-equation (k - \mathcal{R}) turbulence model. This turbulence model can provide superior prediction of turbulent flows with separation, but at a substantially larger computational cost. The two-equation model utilizes additional specification of data at inflow boundaries (see e.g. **INLETT**, **INLETG**) to prescribe the “freestream” turbulence level, and the turbulent viscosity is updated through the solution of two additional transport equations and a “pointwise” eddy viscosity evaluation (see Appendix A). The two-equation model is *not* enabled when **F2EQ**=0, and *is* enabled when **F2EQ**=1.0. If both **F1EQ** and **F2EQ** are set to 1.0, *ADPAC* will use the one-equation turbulence model and set **F2EQ** to 0.

FBCONF [0.0]

The **FBCONF** keyword assigns a trigger which determines the iteration number at which the boundary conditions are frozen. This trigger was added for those cases where convergence is apparently hindered by “noise” from the boundary conditions. Caution must be exercised when using the **FBCONF** variable due to the fact that the *ADPAC* code could ultimately diverge when all of the boundary conditions are frozen. This option is normally turned off [0.0] and is only used for debugging purposes.

FBCWARN [1.0]

The **FBCWARN** keyword assigns a trigger which controls the error checking for outer block boundary conditions normally performed by the *ADPAC* code prior to execution. Following initialization, *ADPAC* normally processes each mesh block and checks to see if the user has adequately specified boundary conditions over the six outer surfaces of each mesh block. If any portion of an outer boundary does not have some type of boundary conditions specified, an error message is normally issued and the code will terminate processing. Under some conditions, the user may have intentionally neglected to specify a boundary condition on an outer mesh block surface, and it is therefore convenient to eliminate this error processing. The **FBCWARN** trigger controls the actuation of this error handling facility. When **FBCWARN**=1.0, the error handling is enabled; when **FBCWARN**=0.0, the error handling is disabled.

FBFRLX [1.0]

The **FBFRLX** keyword sets the relaxation factor used when updating the body forces. This is used for 2-D axisymmetric solutions. Additional details about using body forces in *ADPAC* is found in Sections 2.3 and 3.9.

FCARB(*num*) [0.0]

The keyword **FCARB**(*num*) sets a block specific trigger for the mesh block number specified by *num* which determines, on a block-by-block basis, whether the Cartesian (**FCARB**(*num*) = 1.0) or the cylindrical (**FCARB**(*num*) = 0.0) solution algorithm is employed by that block. The *ADPAC* code permits mixed cylindrical and Cartesian solution blocks in a single calculation. While the variable **FCART** may be used to set the

global value of mesh blocks for either cylindrical or Cartesian solution status, the variable **FCARB**(*num*) may be utilized to set specific blocks one way or the other. It must be noted that the variable **FCARB**(*num*) will always override the status implied by **FCART**. At present, the only boundary condition which permits inter-block communication between mixed cylindrical and Cartesian blocks is **BCPRR**.

FCART [0.0]

The **FCART** keyword assigns a trigger which controls the cylindrical/Cartesian coordinate system solution scheme in the the *ADPAC* code. If **FCART** = 0.0, then all blocks (except those specifically altered by the **FCARB** input variable) are treated as cylindrical coordinate system blocks. If **FCART** = 1.0, then all blocks (except those specifically altered by the **FCARB** input variable) are treated as Cartesian coordinate system blocks.

FCOAG1 [1.0]

The **FCOAG1** keyword specifies the initial, or coarsest coarse mesh level upon which the “full” multi-grid calculation is initially applied (for additional details, see the description of **FFULMG**). When multiple coarse mesh levels are available for processing, it is occasionally useful to specify the initial coarse mesh level in the “full” multi-grid sequence in order to avoid wasted computations on lower mesh levels. Typically, **FCOAG1** is set to **FMULTI** (start “full” multi-grid on coarsest mesh level). In some cases (when **FMULTI** is larger than 3.0) it may be advisable to set **FCOAG1** to 3.0, and avoid additional processing on coarser meshes during the “full” multi-grid startup process. A flowchart of the *ADPAC* iteration and multi-grid control algorithm is given in Figure 3.5. An example is given in the description of **FCOAG2**.

FCOAG2 [2.0]

The **FCOAG2** keyword specifies the final or finest coarse mesh level upon which the “full” multi-grid calculation is applied (for additional details, see the description of **FFULMG**). When multiple coarse mesh levels are available for processing, it is occasionally useful to specify the final coarse mesh level in the “full” multi-grid sequence in order to examine the flowfield without actually performing any calculations on the fine mesh. Typically, **FCOAG2** is set to 2.0, which indicates that the “full” multi-grid startup procedure utilizes all mesh levels from **FCOAG1** to 2 before starting any processing on the fine mesh. A flowchart of the *ADPAC* iteration and multi-grid control algorithm is given in Figure 3.5.

FCOCOM [0.0]

The **FCOCOM** keyword assigns a trigger which determines the method by which cell face areas and cell volumes are determined for the coarse mesh levels of a multi-grid solution. When **FCOCOM**=0.0, coarse mesh cell face areas and cell volumes are determined in exactly the same manner as the fine mesh (using the 8 mesh points defining the vertices of the cell and computing a series of cell face diagonal cross products for the areas, and computing a cell center and additional dot products for the cell volume). When **FCOCOM**=1.0, the coarse mesh cell face areas and cell volumes are computed by summing the cell face areas and cell volumes of the enclosed fine mesh cells relative to each coarse mesh cell. The procedure defined by **FCOCOM**=1.0 is believed to be a more consistent representation of the coarse to fine mesh injection/interpolation process, but experience has not shown any great difference using either method.

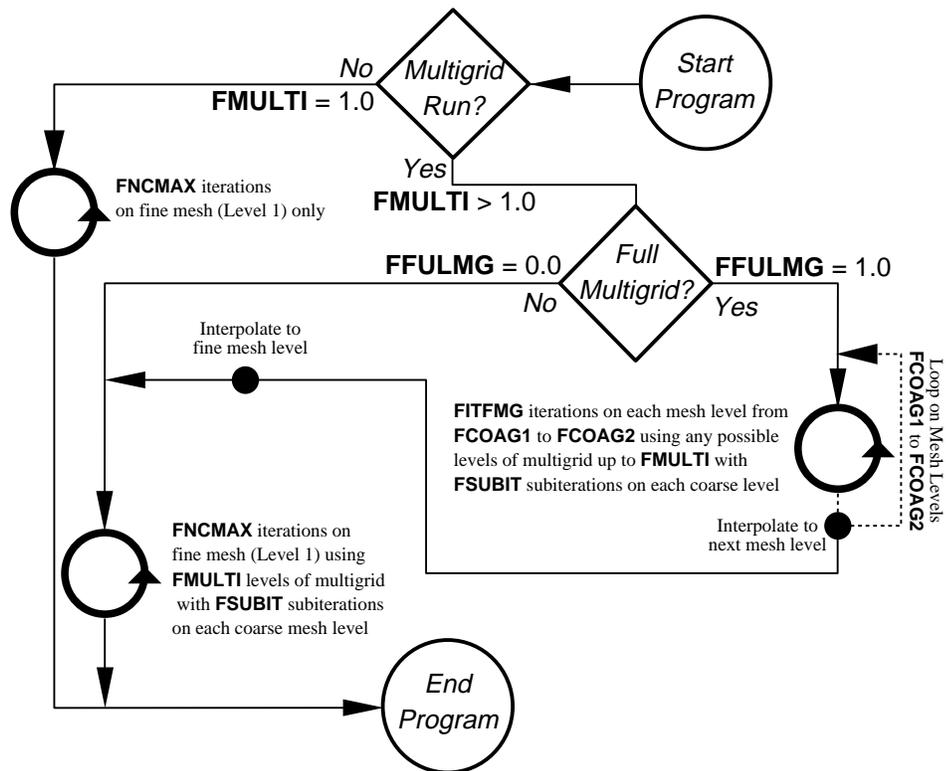


Figure 3.5: ADPAC input keyword multi-grid cycle and time-marching iteration management flowchart.

FCONVRG [-100.0]

The **FCONVRG** keyword specifies the \log_{10} root-mean-square residual level at which the solution is deemed converged. The solution convergence is monitored and when the \log_{10} root-mean-square residual level is less than **FCONVRG**, the time-marching process is terminated and the solution is output. In general, the solution should not be considered converged unless the \log_{10} root-mean-square residual is less than -6.0. The authors do not recommend the use of this variable for explicit time-dependent calculations. The **FCONVRG** variable is useful for terminating the inner iterations of an implicit time-dependent solution.

FDEBUG(*type*) [0.0]

The keyword **FDEBUG**(*type*) defines a block number for the debug output type specified by *type* which determines, on a type by type basis, whether debug output from the *ADPAC* run is printed to the standard output. When enabled, this variable will generate an extreme amount of output and should therefore be used only in a controlled debugging environment. The value of the variable **FDEBUG**(*type*) determines for which blocks the particular type of output is enabled. The following debug output types are currently supported:

- FDEBUG(1)** Print the input (Cartesian) mesh points
- FDEBUG(2)** Print the (converted) cylindrical mesh points
- FDEBUG(3)** Print the cell face areas
- FDEBUG(4)** Print the cell volumes
- FDEBUG(5)** Print the cell flow data
- FDEBUG(6)** Print the cell time steps
- FDEBUG(7)** Print the cell convective fluxes
- FDEBUG(8)** Print the cell dissipative fluxes
- FDEBUG(9)** Print the cell diffusive fluxes
- FDEBUG(10)** Print the cell implicit residual smoothing data

FDELTAT [0.0]

The **FDELTAT** keyword assigns the value for the physical time increment (in seconds) used during a time-dependent solution (either explicit or implicit). For explicit solutions, it is up to the user to ensure that the value of **FDELTAT** does not violate the stability characteristics of the explicit flow solver. For implicit solutions, this value should reflect a reasonable number of global time steps (approximately 100 for periodic flow) over the course of the unsteady aerodynamic phenomena of interest.

FDESIGN [0.0]

The **FDESIGN** keyword assigns a trigger which determines directly whether to use the body force design system calculation procedure. When enabled, this option dictates that a

solution be performed on a 2-D axisymmetric grid representative of a throughflow calculation for a turbomachinery flowpath. The solution requires the input of a body force file (see the description of **BFFILE**) which specifies the blade blockage (a description of the contents of a body force file is given in Section 3.9). The 2-D grid *must* be constructed such that it represents the 2-D hub to tip mean flow stream surface. In other words, if the mean flow involves swirl, then the grid is warped in the circumferential direction to indicate the degree of swirl represented by the mean stream surface. The *ADPAC* flow solver then iteratively updates the body forces internal to the code until the predicted mean stream surface matches the mean stream surface defined by the mesh. This feature was developed for use in a design-like environment wherein gross flow properties such as turning may be known, but specific airfoil shapes may not be known.

FFAST [0.0]

The **FFAST** keyword assigns a trigger which incorporates some simplifications of the *ADPAC* multi-grid algorithm which reduces the CPU time per multi-grid cycle. The CPU savings afforded when this option is enabled (**FFAST**=1.0) is estimated to be about 20%. Unfortunately, the *ADPAC* algorithm is less stable using the procedures enabled by **FFAST**=1.0 and in general, this option is not recommended.

FFILT [1.0]

The **FFILT** keyword assigns a trigger which determines directly whether the added dissipation routines are called during the time-marching process. If **FFILT** = 0.0, then no added dissipation is calculated. It is also possible to turn off the added dissipation by setting the values of **VIS2** and **VIS4** to 0.0; however, the use of **FFILT** avoids the calculation of the dissipation terms entirely. It is unlikely that any value other than 1.0 is required except for code debugging purposes.

FFULMG [0.0]

The **FFULMG** keyword assigns a trigger which determines whether the “full” multi-grid solution procedure is applied or whether the standard multi-grid procedure is used. The use of “full” multi-grid is advisable (but not required) when a new calculation is being started as a means of rapidly generating a better initial guess for the final flowfield. If the solution is being restarted from a previous calculation (**FREST**=1.0), it is usually advisable to set **FFULMG** to 0.0 to avoid destroying the initial data read from the restart file (a warning message is issued when this combination is specified). A flowchart of the *ADPAC* iteration and multi-grid control algorithm is given in Figure 3.5.

FGRAFINT [1.0]

The **FGRAFINT** keyword determines the number of iterations between flowfield display updates for the *ADPAC* real time graphics display system. This option is only valid when **FGRAFIX** = 1.0, and is subject to a number of other restrictions for the graphics display system (see the description of input keywords **FGRAFIX** and **FIMGSAV**, and the description of the graphics display system, Chapter 5). The default value for **FGRAFINT** is 1.0, which indicates that the graphics display will be updated every iteration. This can cause excessive computational and network overhead, and the user should be aware of the potential problems when using the graphics display features. The use of the graphical display features of the *ADPAC* code are not generally recommended.

FGRAFIX [0.0]

The **FGRAFIX** keyword sets a trigger which controls the generation of the real time interactive graphics display in the *ADPAC* program. A value of **FGRAFIX** = 1.0 indicates that the interactive graphics display facility is desired, while **FGRAFIX** = 0.0 turns this option off. When functional, the graphics screen is updated with the latest available flow data every **FGRAFINT** iterations. Graphics images can be automatically captured on specific computer hardware every **FIMGSAV** iterations as a means of creating flowfield animations (see Chapter 5). The generation of interactive, real time graphics images increases the overall computational cost, and can cause network overloading in some cases due to the transmission of graphics information. The use of the graphical display features of the *ADPAC* code are not generally recommended.

FIMGINT [0.0]

The **FIMGINT** keyword determines the number of iterations between flowfield graphics display image capturing available on Silicon Graphics computers for the *ADPAC* real time graphics display system. This option is only valid when **FGRAFIX** = 1.0, and **FIMGSAV** = 1.0, and is subject to a number of other restrictions for the graphics display system (see Chapter 5). The default value for **FIMGINT** is 0.0, which turns off the image capturing. This default value was chosen to prohibit accidental image capturing, which can quickly fill up a large amount of disk storage. The graphics display system can cause excessive computational and network overhead, and the user should be aware of the potential problems when using this feature of the *ADPAC* code. The use of the graphical display features of the *ADPAC* code are not generally recommended.

FIMGSAV [0.0]

The **FIMGSAV** keyword sets a trigger which controls the Silicon Graphics computer screen image capturing facility of the real time interactive graphics display in the *ADPAC* program. A value of **FIMGSAV** = 1.0 indicates that the graphics image capturing facility is desired, while **FIMGSAV** = 0.0 turns this option off. When the interactive graphics display option has been enabled (see details for input keywords **FGRAFIX**, **FGRAFINT**) the graphics screen is updated with the latest available flow data every **FGRAFINT** iteration. When the image capturing facility is enabled, these graphics images can be automatically captured on specific computer hardware every **FIMGINT** iterations as a means of creating flowfield animations (see Chapter 5). The capturing of many screen images will also require a large amount of file storage space (see Section 3.5 for a description of the image capturing file naming convention). The use of the graphical display features of the *ADPAC* code are not generally recommended.

FIMPFAC [2.0]

The **FIMPFAC** keyword is a simple trigger which determines the degree of time-accuracy employed in the implicit solution algorithm enabled by the keyword **FIMPLIC**. There are four options available for this keyword. Values of 1.0 and 2.0 utilize the “fully” implicit strategy described in Appendix A and are first- and second-order accurate in time, respectively. Values of -1.0 and -2.0 utilize a lagged “fully” implicit strategy (thought to improve stability) and are first- and second-order accurate in time, respectively.

FIMPLIC [0.0]

The **FIMPLIC** keyword is a simple trigger to determine whether the iterative implicit solution mode is enabled (enabled when **FIMPLIC** = 1.0). The implicit algorithm is used *only* for time dependent flow calculations, and requires that the additional input flow variables **FDELTAT** and **FNTSTEP** be specified. The *ADPAC* code must also be compiled with array parameter **FIMPRA** set to 1 to properly allocate array storage in the code. When enabled, the implicit algorithm performs a global time marching loop strategy whereby each iteration of the global loop involves an iterative solution of a pseudo steady-state problem using the standard explicit time-marching strategy. The features of this inner iteration problem is still governed by input variables such as **CFL**, **CFMAX**, **FMULTI**, **FNCMAX**, etc. The global time increment for the outer loop is set by the value of **FDELTAT**. The implicit algorithm occasionally exhibits unstable behavior and the solution should initially be closely monitored. The user should fully understand the implications and use of the implicit algorithm before attempting to utilize this option. A brief theoretical description of the procedure is given in Appendix A.

FINVVI [0.0]

The **FINVVI** keyword is a simple trigger to determine whether the solution mode is for inviscid flow (**FINVVI** = 0.0) or for viscous flow (**FINVVI** = 1.0). This trigger controls whether the viscous stress flux contributions are calculated during the time-marching process. This does not affect the application of boundary conditions, as this is completely controlled by the specifications in the boundary data file. As such, it is possible to run viscous boundary conditions in an inviscid flow solution, and inviscid boundary conditions in a viscous flow solution.

FITCHK [100.0]

The **FITCHK** keyword controls the number of iterations between job check-pointing in the *ADPAC* program. Job check-pointing refers to the process of periodically saving the flowfield information to minimize the loss of data in the event that the job does not terminate normally. As a safety feature, the *ADPAC* program writes out an updated restart file every **FITCHK** iterations. It is not necessary to write out intermediate restart files, but this is considered a good precaution against unexpected problems such as computer failures or system administration quotas. A reasonable interval for check-pointing is either 100 iterations (**FITCHK** = 100.0) or roughly the number of iterations completed within one hour. The intermediate restart files, as well as the final restart file, are all written to the same file name, and therefore previous checkpoints cannot be retrieved when the file is overwritten (see Section 3.5 for restart file naming conventions). Job check-pointing only applies to the iterative cycles involving the fine mesh and does not apply to the coarse mesh iterations calculated during a “full” multi-grid startup (see **FFULMG**).

FITFMG [100.0]

The **FITFMG** keyword dictates the number of iterations to be performed on each of the coarse mesh levels during a “full” multi-grid startup sequence (see **FFULMG**). Typically, the startup sequence is used only to generate a reasonable initial guess for the fine mesh, so the value of **FITFMG** is kept relatively low (≈ 100). The function of the keyword **FITFMG** is illustrated graphically in Figure 3.5.

FKINF [0.0001]

The **FKINF** keyword sets the initial value for the turbulence field and the value is dependent upon the model selected. If the algebraic Baldwin-Lomax model is employed (**F1EQ** = 0.0 and **F2EQ** = 0.0), the keyword **FKINF** is ignored. If the one-equation Spalart-Allmaras model is employed (**F1EQ** = 1.0 and **F2EQ** = 0.0), **FKINF** sets the initial value of χ and should be set to approximately 20. If the two-equation k - \mathcal{R} model is employed (**F1EQ** = 0.0 and **F2EQ** = 1.0), **FKINF** sets the initial value of the nondimensional k field (k_{inf}/U_{ref}^2) and should be on the order of 0.0001.

FMGTSTEP [0.0]

The **FMGTSTEP** keyword assigns the number of global time steps to be evaluated on coarse mesh levels for the implicit flow solver when the “full” multi-grid option **FFULMG** is enabled. In the processing of large time-dependent calculations using the implicit flow solver, it has been found useful to employ the “full” multi-grid type of startup procedure for the time-dependent runs as well as the steady-state analyses. This basically initializes the multiple time levels data arrays such that when the fine mesh solution is activated, the data in the previous time level arrays are initialized with something other than the current solution, and a hopefully better approximation of the time-derivatives can be performed. **FMGTSTEP** sets the number of *implicit* time steps performed on the coarse meshes during the “full” multi-grid startup. This keyword will only be active when **FIMPLIC**=1.0 and **FFULMG**=1.0. For a more detailed description of the “full” multi-grid startup procedure, see the descriptions of **FFULMG**, **FCOAG1**, and **FCOAG2**.

FMIXLEN [-1.0]

The **FMIXLEN** keyword triggers the mixing-length turbulence model. For positive values of **FMIXLEN**, the algebraic Baldwin-Lomax model is bypassed and the turbulent eddy viscosity is calculated using a mixing-length formulations. **FMIXLEN** should be set such that **FMIXLEN** × **DIAM** is approximately 0.0025 ft (0.030 in).

FMULTI [1.0]

The **FMULTI** keyword assigns the number of multi-grid levels to be used during the calculation. The code will analyze the dimensions of the fine mesh to determine whether it can be properly subdivided according to the number of multi-grid levels requested. If **FMULTI** ≤ 1.0, then the number of multi-grid levels is set to 1, and the calculation is performed on the finest mesh only without multi-grid acceleration. For unsteady flows using the explicit time-marching strategy, the current multi-grid scheme is not valid, and even though it is allowed, **FMULTI** should be set to 1.0. For time-dependent flows based on the implicit time-marching strategy (**FIMPLIC**=1.0), the multi-grid algorithm may be enabled as with any steady-state solution. A flowchart of the *ADPAC* iteration and multi-grid control algorithm is given in Figure 3.5.

FNCMAX [100.0]

The **FNCMAX** keyword controls the total number of iterations for a calculation without multi-grid (**FMULTI** ≤ 1.0), or the number of global iterations on the finest mesh for a multi-grid calculation (**FMULTI** > 1.0). The total number of iterations performed on all meshes for a multi-grid run is controlled by a combination of **FNCMAX**, **FMULTI**, **FCOAG1**, **FCOAG2**, **FFULMG**, **FITFMG**, and **FSUBIT**. See the descriptions of the

variables **FNCMAX**, **FMULTI**, **FCOAG1**, **FCOAG2**, **FFULMG**, **FITFMG**, and **FSUBIT** for further details. A flowchart of the *ADPAC* iteration and multi-grid control algorithm is given in Figure 3.5.

FNTSTEP [0.0]

The **FNTSTEP** keyword assigns the number of global time steps to be evaluated on the finest mesh during an implicit time-dependent solution. **FNTSTEP** sets the number of global time steps performed while **FNCMAX** sets the number of inner iterations for each global time step. The total physical time of the time-dependent implicit solution is therefore **FNTSTEP** × **FDELTAT**. This variable is only active when **FIMPLIC**=1.0.

FRDMUL [0.0]

The **FRDMUL** keyword determines whether the boundary condition data for the coarse mesh levels of a multi-grid run are generated from the fine mesh boundary conditions specified in the *ADPAC* boundary data file (**FRDMUL** = 0.0), or whether the coarse mesh boundary specifications are read in from the boundary data file (**FRDMUL** = 1.0). In most cases, **FRDMUL** should be set to 0.0, and the program will determine the equivalent coarse mesh boundary conditions from the fine mesh specifications. For the purposes of code debugging, or to permit multi-grid calculation on a mesh which does not possess perfect “multi-grid” boundary segments (a boundary condition for the fine mesh does not begin or end at a mesh index which is compatible with the multi-grid sequence), it is possible to allow the program into running multi-grid by artificially specifying an equivalent coarse mesh boundary condition.

FRESID [1.0]

The **FRESID** keyword assigns a trigger which determines directly whether the implicit residual smoothing routines are called during the time-marching process. If **FRESID** = 0.0, then no residual smoothing is applied. It is also possible to turn off the residual smoothing by setting the values of **EPSX**, **EPSY**, and **EPSZ** to 0.0; however, the use of **FRESID** avoids the calculation of the smoothed residuals entirely. It is unlikely that any value other than 1.0 is required except for code debugging purposes, or for calculations involving $CFL \leq 2.0$.

FREST [0.0]

The **FREST** keyword assigns a trigger which controls the restart characteristics of the *ADPAC* code. If **FREST** = 0.0, then no restart file is used, and the flow variables are initialized according to the scheme described under the input keyword **RMACH**. If **FREST** = 1.0, then the code attempts to open a restart file (*case.restart.old*), and the flow variables are initialized by the values given in the restart file. Restarting a calculation from a previous calculation is often useful for breaking up large calculations into smaller computational pieces, and may also provide faster convergence for cases which involve minor changes to a previous calculation.

A third option is provided (**FREST** = -1.0) which allows *ADPAC* to restart from a coarser mesh solution restart file. For very large meshes which permit multi-grid, it may be desirable to initiate the solution using less powerful computers by running on a coarsened mesh file (generated independently from the fine mesh using the *COARSEN*

INPUT KEYWORDS

program). The coarsened mesh *must* represent an exact coarsened mesh from the ultimate fine mesh (every other mesh line removed) for this procedure to work properly.

It should be mentioned that restart files generated by the implicit time-dependent solution strategy will contain additional time level data compared to a steady state restart file. It is possible to restart a time-dependent implicit solution from an explicit steady-state restart file and vice versa, at the expense of some loss of information. For long time-dependent solutions involving multiple restarts, the most recent restart files *must* be utilized to properly predict the time-dependent flow behavior.

FRINF [0.001]

The **FRINF** keyword sets the initial value for the turbulence field and the value is dependent upon the model selected. If the algebraic Baldwin-Lomax model (**F1EQ** = 0.0 and **F2EQ** = 0.0) or the one-equation Spalart-Allmaras model (**F1EQ** = 1.0 and **F2EQ** = 0.0) is employed, the keyword **FRINF** is ignored. If the two-equation k - \mathcal{R} model is employed (**F1EQ** = 0.0 and **F2EQ** = 1.0), **FRINF** sets the initial value of the nondimensional \mathcal{R} field ($\mathcal{R} / V_{ref} L_{ref}$) and should be set along the order of 0.001.

FSAVE [1.0]

The **FSAVE** keyword assigns a trigger which controls the restart file output characteristics of the *ADPAC* code. If **FSAVE** = 0.0, then no restart file is written at the end of an *ADPAC* run. If **FSAVE** = 1.0, then the code attempts to open a restart file (*case.restart.new*), and the flow variables are written to the restart file for future processing. Restarting a calculation from a previous calculation is often useful for breaking up large calculations into smaller computational pieces, and may also provide faster convergence for cases which involve minor changes to a previous calculation. The recommended procedure is to *always* write a restart file.

FSOLVE [1.0]

The **FSOLVE** keyword assigns a trigger which determines which type of explicit time-marching strategy is employed on both fine and coarse meshes. For **FSOLVE** = 0.0, the standard four-stage Runge Kutta time-marching scheme is used with a single added dissipation evaluation, and implicit residual smoothing at alternating stages. For **FSOLVE** = 1.0, a modified four-stage Runge Kutta time-marching scheme is used with two evaluations of the added dissipation, and implicit residual smoothing at every stage. For **FSOLVE** = 2.0, a five-stage Runge Kutta time-marching scheme is used with three weighted added dissipation evaluations, and implicit residual smoothing at every stage. **FSOLVE** = 1.0 is the recommended value, although the other schemes may provide improved convergence at a somewhat different computational cost per iteration.

FSUBIT [1.0]

The **FSUBIT** keyword determines the number of subiterations performed on coarse meshes during the coarse mesh portion of the multi-grid cycle. As such, this variable is actually only used when **FMULTI** > 1.0. Additional subiterations on coarse meshes during the multi-grid cycle can often improve convergence, at the expense of some additional computational work. The number of subiterations specified by **FSUBIT** is applied at each coarse mesh level during the multi-grid calculations process. A value of 1.0, 2.0, or 3.0 is

typically best. A flowchart of the *ADPAC* iteration and multi-grid control algorithm is given in Figure 3.5.

FTIMEI [1.0]

The **FTIMEI** keyword assigns a trigger which determines the number of iterations between time step evaluations. For best results, this should be 1.0, which implies that the time step is re-evaluated at every iteration. However, this value can be increased to reduce CPU time by re-evaluating the time step every **FTIMEI** iterations instead (at the possible expense of irregular convergence). It is recommended that the value of **FTIMEI** not exceed 10.

FTIMERM [0.0]

The **FTIMERM** keyword is utilized to control CPU quota during a run of the *ADPAC* code. This variable is used *primarily* during execution under the Network Queuing System (NQS), and the effect of this variable is different between execution on a Cray computer and a UNIX workstation. On the Cray, for jobs running under NQS, any non-zero value for **FTIMERM** directs the code to determine how much CPU time remains allocated to the current job during each time-marching iteration, and the *ADPAC* code estimates how much of that CPU time is required to normally shut down the current job. If the time remaining to the job allocation is indicated by *TIME*, and if the time required to shutdown is *SHUT*, then the code will evaluate the expression:

$$TIME - SHUT + FTIMERM$$

where each term is in CPU seconds. If this expression is less than 0.0, then the code will halt the time marching process and attempt to shut down so the various output files can be written prior to termination by NQS due to CPU quota. Note that if **FTIMERM** is a negative number, then the code will shut down “early” in case additional programs must run under a given NQS run.

On a UNIX workstation, NQS is usually not available, and the code keeps track of accumulated CPU time and terminates normal job processing when the accumulated CPU time exceeds the value of **FTIMERM**. This option can be used to cleanly stop an *ADPAC* job after a prescribed amount of time. As an alternative to using **FTIMERM**, it is generally recommended to determine the time required for a single iteration and set **FNCMAX** accordingly to complete the *ADPAC* job prior to exceeding the time limit specified by **FTIMERM**. If **FTIMERM** = 0.0, then no action is taken under any circumstances.

FTIMFAC [1.0]

The **FTIMFAC** keyword is a coefficient multiplier of the pseudo time step which determines the limiting time step for the implicit iteration strategy enabled by the keyword **FIMPLIC** = 1.0. Under most circumstances, the pseudo time step (as opposed to the *physical* time step) employed by the implicit inner iteration strategy requires some limiting to prevent instabilities in the global time marching process. The value of **FTIMFAC** is a “safety factor” of sorts by simply limiting the value of the pseudo time step. Larger values of **FIMPFAC** imply more limiting, and hence improved stability (theoretically). Under no circumstances should **FTIMFAC** ever be less than 1.0. Values of between 1.0 and 10.0 are recommended.

FTOTSM [0.0]

The **FTOTSM** keyword is used to trigger the post multi-grid smoothing algorithm. In this scheme, the residual corrections from the multi-grid process are combined with the fine mesh residuals and are smoothed globally using a simple constant coefficient version of the implicit residual smoothing algorithm. The smoothing coefficient is determined by the value of the input keyword variable **EPSTOT**. The scheme is disabled when **FTOTSM** = 0.0, and is employed when **FTOTSM** = 1.0. This scheme has been found to aid stability, but can actually hinder convergence in some cases.

FTURBB [10.0]

The **FTURBB** keyword assigns a trigger which determines the number of iterations before the turbulence model is activated. For laminar flow, set **FTURBB** to a very large number (i.e., 99999.0) so the turbulence model is never called. For turbulent flow, the value should be large enough (≥ 10) to ensure that the solution has developed adequately enough to permit stable implementation of the turbulence model (e.g., the flowfield should at least exhibit the gross characteristics such as correct flow direction, some boundary layer development) of the expected final flow before the turbulence model is activated.

FTURBCHT(*num*) [0.0]

The keyword **FTURBCHT**(*num*) sets a block specific trigger for the mesh block number specified by the value *num* to enable the body-centered mesh turbulence model described in Figure 3.4. If **FTURBCHT**(*num*) is set to 0.0, the standard turbulence model is used for the block specified by *num*. If **FTURBCHT**(*num*) is set to 1.0, then the special transition and body centered turbulence model is used for the block specified by *num*. The body-centered turbulence model locates the airfoil leading and trailing edges, and utilizes the input variables **XTRANS**, **XTRANPS** and **CMUTSS**, **CMUTPS** to determine the natural transition point on the airfoil. This turbulence model was developed during an analysis of surface heat transfer (where transition plays a critical role) on a turbine vane cascade using a C-type mesh. The use of this model is recommended whenever the mesh topology is compatible with the scheme illustrated in Figure 3.4. Note that the “pressure” and “suction” surfaces defined in Figure 3.4 actually refer to geometric orientation rather than aerodynamic function.

It should be noted that these variables will have no effect when **FINVVI**=0.0 (inviscid flow), or when either **F1EQ**=1.0 or **F2EQ**=1.0 (using the one- or two-equation turbulence model) or when **FTURBCHT**=0.0 (transition model not activated).

FTURBF [0.0]

The **FTURBF** keyword assigns a trigger which determines the iteration number at which the turbulence model is frozen. This trigger was added for those cases where convergence is apparently hindered by “noise” from the turbulence model. The default value is set to 0.0 in order to prevent accidental freezing of the turbulence field. Caution must be exercised when using the **FTURBF** variable due to the fact that the *ADPAC* restart file does not contain any turbulent viscosity data when the algebraic Baldwin-Lomax or mixing-length turbulence model is used. If the *ADPAC* code is restarted from a turbulent flow solution when the value of **FTURBF** is less than the current iteration level, no turbulent quantities will be generated and the flow will exhibit laminar flow characteristics. In general, it is safest to make this a very large number to avoid problems.

FTURBI [1.0]

The **FTURBI** keyword assigns a trigger which determines the number of iterations between turbulence model evaluations. For best results, this should be 1.0, which implies that the turbulence parameters are re-evaluated at every iteration. However, this value can be increased to reduce CPU time by re-evaluating the turbulence quantities every **FTURBI** iterations at the possible expense of irregular convergence. It is recommended not to exceed a value of 10 for **FTURBI**.

FUNINT [0.0]

The **FUNINT** keyword is used to determine the number of iterations between instantaneous *PLOT3D* absolute flow file output. For a time-dependent calculation, it is often desirable to print out data at several intervals during the course of the solution to examine the time-dependent nature of the flow. For steady-flow calculations, it is normally desirable to keep **FUNINT** set to its default [0.0] to suppress flow file output and simply use the final output *PLOT3D* format files if needed. For unsteady-flow calculations, the value of **FUNINT** can be highly case dependent, and some numerical experimentation may be required to prevent excessive output or a deficiency in data. The file naming convention for the unsteady output files is explained in Section 3.5 (*case.p3fr.#*). Proper selection of **FDELTA** and **FUNINT** resulting in *PLOT3D* output files at specific intervals of time will likely produce the most useful results.

FUPWIND [0.0]

The **FUPWIND** keyword is a simple trigger to activate the upwind differencing scheme (on = 1.0, off = 0.0) available for the 2-D mesh block solver in the *ADPAC* code. The upwind differencing scheme is a first-order scheme available for experimentation only, and is not a recommended solution technique for actual flow calculations at this point.

FVTSFAC [2.5]

The **FVTSFAC** keyword determines the value of the viscous time step evaluation factor used to stabilize the time-marching solution for viscous flows. This factor is used to magnify the importance of the diffusion-related contributions to the time step evaluation (larger values suggest larger restrictions due to diffusion related parameters). This factor is particularly useful for meshes with rapid changes in grid spacing, and the default value of 2.5 was prescribed somewhat arbitrarily following numerical experimentation. It is unlikely that this value needs modification for most cases.

FWALLF [1.0]

The **FWALLF** keyword is used to trigger the use of wall functions in the algebraic model and one-equation turbulence model (**F1EQ** = 1.0), but not the two-equation model (**F2EQ** = 1.0). Wall functions are normally desirable for meshes which are not highly clustered near solid surfaces. The *ADPAC* code can determine when the wall function model is necessary and will automatically disable the wall function model (even if **FWALLF** is enabled) in favor of the standard turbulence model wall treatment for meshes with acceptable near-wall spacing ($y^+ < 5$ or roughly 0.0001 times airfoil chord for turbomachinery applications). At this point, the wall function model is not recommended for applications involving significant heat transfer or massive flow separation.

GAMMA [1.4]

The **GAMMA** keyword sets the value for the gas specific heat ratio. For most cases involving air at moderate pressures and temperatures, a value of 1.4 is adequate. For cases involving combustion products, this value may be quite different, and should be considered appropriately. Extreme care must be taken when post-processing a calculation which is based on a value of **GAMMA** other than 1.4 as many post processors use an assumed value of the specific heat ratio equal to 1.4 (*PLOT3D* is a common example). It should be mentioned that the present version of the code does not permit user specification of the fluid viscosity, as the formula for air is hardwired into the code.

NBLD(*num*)

The **NBLD**(*num*) keyword specifies the number of blades in the mesh block numbered *num*. The **NBLD** factor is used when calculating mass flows and forces to simulate the entire geometry, even though only a single blade row may be actually calculated. The specification of **NBLD** is very important when using the one-equation Spalart-Allmaras turbulence model (**F1EQ** = 1.0); the near-wall distance routine needs the information regarding the pitch of the blades in a cylindrical setting to correctly calculate the near-wall distance accounting for periodicity.

P3DPRT [1.0]

The **P3DPRT** keyword assigns a trigger which determines whether *PLOT3D* format output files are written at the end of a calculation. A value of **P3DPRT** = 1.0 indicates that the output files should be written. Conversely, a value of **P3DPRT** = 0.0 indicates that the *PLOT3D* format output files should not be written. The *PLOT3D* output files are useful for graphically examining the predicted flow quantities using widely available plotting software such as *PLOT3D* and *FAST*. Occasionally, however, due to disk space limitations or simply to speed up execution during initial runs, it may be desirable to eliminate this output feature.

PREF [2116.22]

The **PREF** keyword sets the dimensional value (lbf/ft²) of the reference total pressure used to nondimensionalize the flowfield. For viscous flows, this value must be accurately specified in order to properly set the nondimensional flow viscosity, and the Reynolds number. For inviscid flow predictions, this value has no real significance because of the similarity of inviscid flows with Mach number. It is very important with respect to stability and convergence to choose an average representative value for this variable, such that the nondimensional total pressure at any point in the flow is near a value of 1.0. An extended discussion on the reason for this choice is given in the description of **RMACH**. In general, **PREF** is set to the freestream or inlet flow average total pressure.

PRNO [0.7]

The **PRNO** keyword assigns the value of the gas Prandtl number. For air (and many other gases) at moderate pressures and temperatures, a value of 0.7 is appropriate.

PRTNO [0.9]

The **PRTNO** keyword assigns the value of the gas turbulent Prandtl number. The turbulence model in *ADPAC* determines the turbulent thermal conductivity via a turbulent Prandtl number and the calculated turbulent viscosity. The recommended value is 0.9.

RGAS [1716.26]

The **RGAS** keyword sets the dimensional value (ft-lbf/slug-°R) of the gas constant. The default value is for atmospheric air at standard pressure and temperature. This value is used in conjunction with **GAMMA** in determining the gas specific heats at constant pressure and constant volume.

RMACH [0.5]

The **RMACH** keyword value is intended to set an initial flow Mach number. This value is used primarily to set the initial freestream flow variables (density, pressure, temperature, and axial velocity) for a given calculation based on a fixed Mach number. The freestream values are used to initialize the flowfield prior to the execution of the time-marching solver and should be used only in the absence of a restart file. It should be mentioned that the initial data values are assigned based on the assumption that the nondimensional freestream total pressure and total temperature are 1.0 (the dimensional values are the **PREF** and **TREF** input values). This implies that it is advisable to set up all input variables (in particular **PREF** and **TREF**) and boundary data such that the imposed inlet and exit flow boundary conditions are compatible with the initial conditions derived from **RMACH**. In addition, the value of **RMACH** is used in conjunction with the value of advance ratio specified by the keyword **ADVR**, when the rotational speed is defined in this manner. In this case, the value of **RMACH** must be the freestream Mach number associated with the advance ratio specified by **ADVR** or an incorrect rotational speed will be calculated. A common error when using the **RMACH** input variable is to assume that the specification of the reference Mach number will set the flow for the case of interest. This is not true, as the boundary condition specifications will ultimately determine the flow conditions.

RPM(*num*) [0.0]

The **RPM** keyword value determines the rotational speed (rev/min) of the mesh block number specified by the value *num*. The value of **RPM** is by nature a dimensional value. Block rotational speeds are zero by default unless either a **RPM** or an **ADVR** keyword are specified. The user should be aware that if the mesh has not been correctly non-dimensionalized, it is then possible that an incorrect value of rotational speed would be used in the calculation (see the description of the keyword **DIAM**). The user should also be aware that this value is sign specific, and many computational problems have been traced to geometries which were rotating the wrong way. The proper orientation for the **RPM** specification is illustrated in Figure 3.6.

TREF [518.67]

The **TREF** keyword sets the dimensional value (°R) of the reference total temperature used to nondimensionalize the flowfield. For viscous flows, this value must be accurately specified in order to properly set the nondimensional flow viscosity and the Reynolds number. This value is also important for the specification of wall temperature used in the viscous wall boundary condition (i.e., **SSVI**, **SS2DVI**). For inviscid flow predictions, this value has no real significance because of the similarity of inviscid flows with Mach number. It is very important with respect to stability and convergence to choose an average representative value for this variable, such that the nondimensional total temperature at any point in the flow is near a value of 1.0. In general, **TREF** is set to the freestream or

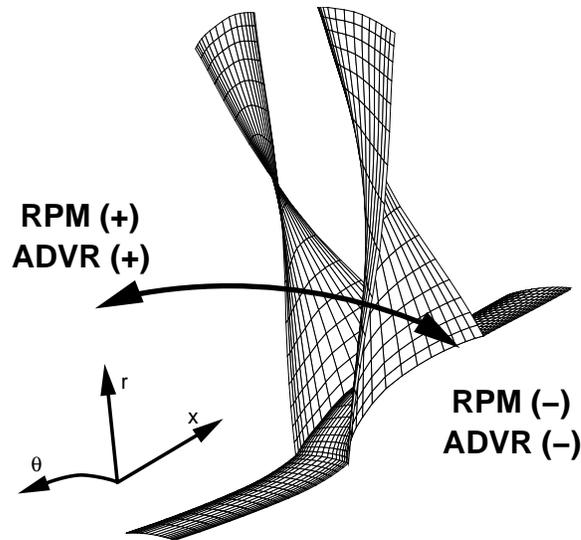


Figure 3.6: ADPAC rotational speed orientation illustration.

inlet flow average total temperature.

VIS2 [0.500]

The **VIS2** keyword defines the value of the second-order added dissipation term used in the fine mesh time-marching solver (see Appendix A). This value is a simple multiplier of the second-order dissipation term (larger values imply more added dissipation). Second-order dissipation is used mainly to control the solution in the vicinity of flow discontinuities such as shock waves, but can also be important in any high gradient region. The recommended value is 0.5, but values from 0.0 (no second-order dissipation) to 2.0 may be necessary. Any value larger than 2.0 is of questionable use, as the added dissipation will likely dominate the solution.

VIS4 [0.015625]

The **VIS4** keyword defines the value of the fourth-order added dissipation term used in the fine mesh time-marching solver (see Appendix A). This value is a simple multiplier of the fourth-order dissipation term (larger values imply more added dissipation). Fourth order dissipation is used mainly to provide a background dissipation to control the odd/even point decoupling associated with centered differencing schemes. The recommended value is 0.015625 ($\frac{1}{64}$), but values from 0.0 (no fourth-order dissipation) to 0.0625 ($\frac{1}{16}$) may be necessary. Any value larger than 0.0625 is of questionable use, as the added dissipation will likely dominate the solution.

VISCG2 [0.125]

The **VISCG2** keyword controls the value of the second-order added dissipation coefficient for coarse mesh subiterations during the multi-grid time-marching solution process. Coarse mesh subiterations utilize a simpler dissipation scheme than the fine mesh time-marching scheme, and therefore, a different damping constant is required. Larger values imply increased dissipation. The recommended value is **VISCG2** = 0.125, although

values from 0.0 (no dissipation) to 1.0 are possible. Values larger than 1.0 are not recommended as the solution would then likely be dominated by the dissipation.

WBF(*num*) [0.0]

The **WBF**(*num*) keyword triggers whether or not to write out a body force file for block *num*. If **WBF**(*num*) is set to 1.0, the file for block *num* will be written. This is valid only for H-grids with constant axial and radial locations across the circumferential pitch (Dawes-type).

XMOM, YMOM, ZMOM [0.0, 0.0, 0.0]

The three keywords **XMOM**, **YMOM**, and **ZMOM** specify the point in space about which the six moment components are calculated in Cartesian space as shown in Figure 3.7. The keywords are only used when the geometry is solved in the Cartesian reference frame (**FCART** = 1.0).

XTRANS, XTRANPS [0.0]

The **XTRANS** and **XTRANPS** keywords determine the percentage of axial chord at which transition is forced to occur for the point transition model in the *ADPAC* body centered mesh turbulence model activated by the keyword **FTURBCHT**. This simplified transition model maintains laminar flow either until transition occurs based on the values of **CMUTPS** and **CMUTSS** or until the percentage of axial chord indicated by **XTRANS** and **XTRANPS** is exceeded at which point complete transition is forced. Separate variables are provided for the “suction side” and “pressure side”, respectively, of the airfoil in question. The transition model parameters are illustrated in Figure 3.4. Fully turbulent (non-transitional) flows should set **XTRANS** and **XTRANPS** to 0.0 (transition occurs at leading edge). Other values must be determined on a case by case basis. “Natural” transition can occur based on **CMUTPS** and **CMUTSS**.

It should be noted that these variables will have no effect when **FINVVI**=0.0 (inviscid flow), or when either **F1EQ**=1.0 or **F2EQ**=1.0 (one- or two-equation turbulence model enabled) or when **FTURBCHT**=0.0 (transition model not activated).

ZETARAT [0.6]

The keyword **ZETARAT** controls a parameter used in the eigenvalue scaling operator in the residual smoothing algorithm (see Appendix A). The value of **ZETARAT** represents the exponent the ratio of two coordinate eigenvalues and therefore large values of **ZETARAT** (≥ 0.6) imply increased bias for meshes with large differences in coordinate spacing while small values of **ZETARAT** (≤ 0.5) imply decreased bias for meshes with large differences in coordinate spacing. Normally, values between 0.5 and 0.6 are recommended. Physically, this implies that large values of **ZETARAT** give preference to smoothing in a single coordinate direction (preference given to most highly clustered coordinate).

3.7 ADPAC Boundary Data File Description

The *ADPAC* boundary data file contains the user-specified parameters which control the application of boundary conditions on the multiple-block mesh during a solution. These

Six Moment Components listed in the ADPAC case.forces file

$$F_x * delY = |\vec{F}_x| \Delta y = \left\{ \left(\vec{F}_p + \vec{F}_v \right) \cdot \hat{x} \right\} \Delta y$$

$$F_x * delZ = |\vec{F}_x| \Delta z = \left\{ \left(\vec{F}_p + \vec{F}_v \right) \cdot \hat{x} \right\} \Delta z$$

$$F_y * delX = |\vec{F}_y| \Delta x = \left\{ \left(\vec{F}_p + \vec{F}_v \right) \cdot \hat{y} \right\} \Delta x$$

$$F_y * delZ = |\vec{F}_y| \Delta z = \left\{ \left(\vec{F}_p + \vec{F}_v \right) \cdot \hat{y} \right\} \Delta z$$

$$F_z * delX = |\vec{F}_z| \Delta x = \left\{ \left(\vec{F}_p + \vec{F}_v \right) \cdot \hat{z} \right\} \Delta x$$

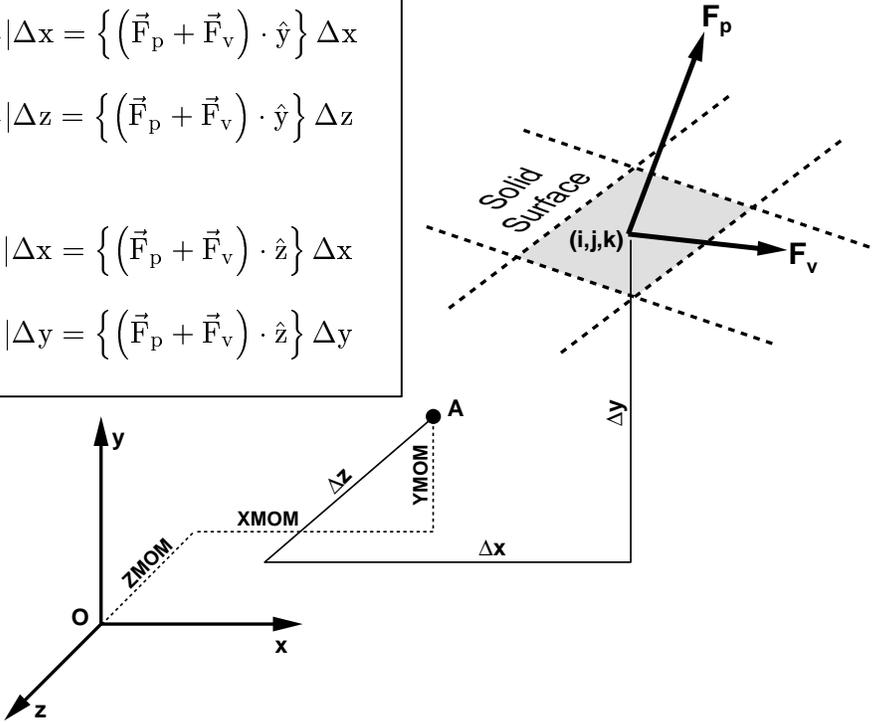
$$F_z * delY = |\vec{F}_z| \Delta y = \left\{ \left(\vec{F}_p + \vec{F}_v \right) \cdot \hat{z} \right\} \Delta y$$


Figure 3.7: Location of **XMOM**, **YMOM**, and **ZMOM** with respect to the calculation of moment components.

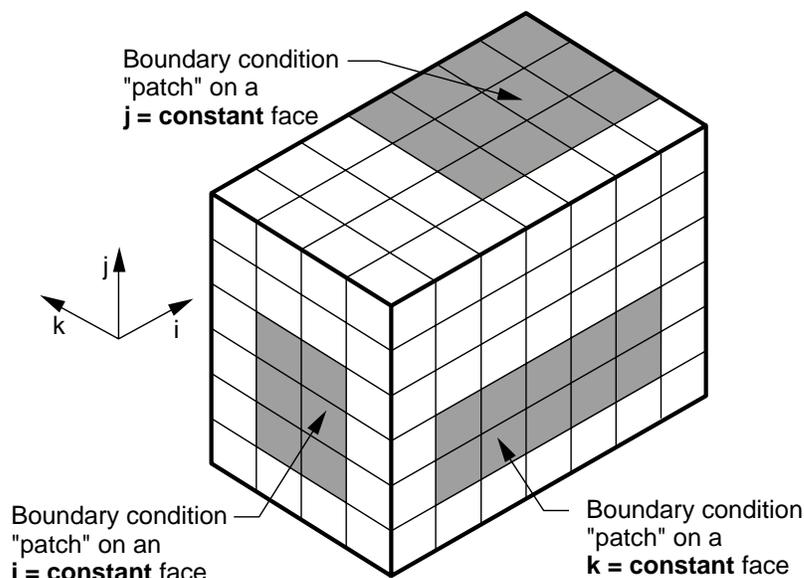


Figure 3.8: ADPAC 3-D boundary condition specification.

boundary specifications determine the location of solid walls, inflow/outflow regions, and block-to-block communication paths. Prior to a detailed discussion of the individual boundary condition specifications, several boundary condition application concepts should be explained. It is important to understand how boundary conditions are applied in the ADPAC finite volume solution scheme.

Finite volume solution algorithms typically employ the concept of a phantom cell to impose boundary conditions on the external faces of a mesh block. This concept is briefly detailed in Appendix A. All ADPAC boundary condition specifications provide data values for phantom cells to implement a particular mathematical boundary condition on the mesh. It should be emphasized that the phantom cells are automatically defined within the ADPAC code, and the user need not be concerned about generating fictitious points within the mesh to accommodate the boundary condition application procedure (mesh points need only be generated for the actual flow domain).

Although boundary conditions are imposed at phantom cells in the numerical solution, the boundary specification is still most conveniently *defined* in terms of grid points, not computational cells. An illustration of the boundary specification method for ADPAC is given in Figure 3.8. All boundary conditions are specified in terms of the grid points on either an i -constant, j -constant, or k -constant mesh surface. In practice, these surfaces are typically on the outer boundaries of the mesh block, but it is also possible to impose a boundary condition in the interior of a mesh block.

Another important aspect of the application of boundary conditions in the ADPAC code involves the order in which boundary conditions are applied. During the execution of the ADPAC code, all boundary conditions are applied to the various mesh blocks in the order in which they are specified in the *case.boundata* file. As a result, it is possible to overwrite a previously specified boundary patch with subsequent different boundary conditions. This concept is illustrated graphically in Figure 3.9. The user must

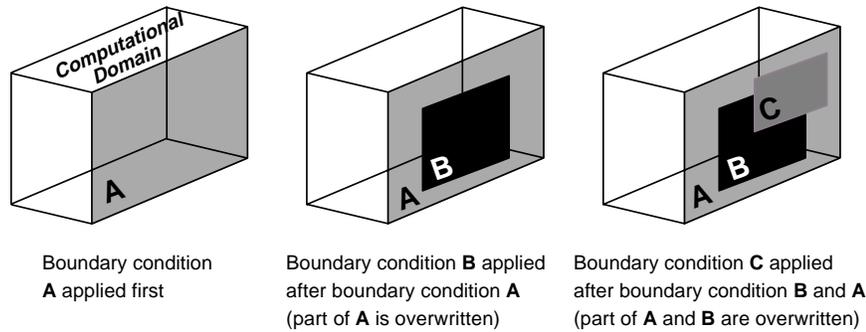


Figure 3.9: Effect of ordering in application of boundary conditions for the *ADPAC* code.

take proper precautions to prohibit accidentally overwriting a desired boundary patch as the *ADPAC* code cannot distinguish the proper order for the user. For example, overwriting a communication boundary condition (**PATCH**) with a solid no-slip surface boundary condition (**SSVI**) will result in the incorrect viscosity data at the solid boundary.

During code execution, the boundary data file is read one line at a time as a character string, and each string is parsed sequentially to determine the specific program action in each case. The boundary data file utilizes a keyword input format, such that any line which does not contain a recognizable keyword is treated as a comment line. Therefore, the user may place any number of comments in the file (so long as the line does not contain a keyword input string in the form described below), and code execution is unaltered. All boundary data file lines are echoed to the standard output, and the program response to each line is listed when a specific action is taken. A line in the boundary data file can also be changed to a comment by inserting a # character in the first column.

All keyword input lines are given in the format listed in Figure 3.10. The actual specification in the boundary data file may be free format, as long as the individual parameter specifications are given in the correct order and are separated by one or more blank spaces. All boundary specifications begin with a line containing 19 variables as outlined by the labels in Figure 3.10. This figure also shows the relationship of the ordered variables to the mesh surface they define (**A** or **B**). A description of the function of each of the variables in the boundary specification line is given in the proper order in the section below:

Boundary Specification Line Variables Descriptions

BCTYPE The first variable, **BCTYPE**, is a character string defining the type of boundary condition to be applied to a given mesh block. **BCTYPE** must correspond to one of the reserved boundary condition keywords defined later in this section to be a proper boundary specification. If **BCTYPE** is not one of the reserved names, then the boundary specification line is ignored.

LBLOCK1 The variable **LBLOCK1** is an integer defining the grid block number to which the boundary condition implied by **BCTYPE** is applied.

BCTYPE	LBLOCK1	LBLOCK2	LFACE1	LFACE2	LDIR1	LDIR2	LSPEC1	LSPEC2	L1LIM	L2LIM	M1LIM1	M1LIM2	N1LIM1	N1LIM2	M2LIM1	M2LIM2	N2LIM1	N2LIM2
PATCH	1	2	I	I	P	M	J	K	1	33	1	17	1	25	33	49	1	25
	A	B	A	B	A	B	A _B	A _B	A	B	A	A	A	A	B	B	B	B

A First Surface in **PATCH** Statement
B Second Surface in **PATCH** Statement

Figure 3.10: ADPAC boundary data file specification format.

Naturally, this implies $1 \leq \mathbf{LBLOCK1} \leq \mathbf{NBLKS}$, where **NBLKS** represents the last mesh block number, unless **FRDMUL** = 1.0 and the coarse mesh boundary conditions are read in.

LBLOCK2 The variable **LBLOCK2** is an integer defining the grid block number from which the boundary condition data implied by **BCTYPE** and applied to mesh block **LBLOCK1** is obtained. In some cases, a boundary specification may involve more than one block (i.e., **PATCH**), and the **LBLOCK2** variable is provided for this purpose. If the boundary specification only involves a single block, then set **LBLOCK2** = **LBLOCK1**.

LFACE1 The variable **LFACE1** is a single character (I, J, or K) specifying the grid plane (i, j, or k constant, respectively) to which the boundary condition is applied in block **LBLOCK1**, based on the method by which boundary conditions are implemented in the finite-volume solution scheme.

LFACE2 The variable **LFACE2** is a single character (I, J, or K) specifying the grid plane (i, j, or k constant, respectively) from which the boundary condition data is derived in block **LBLOCK2**. Naturally, this variable is only useful for boundary specifications involving more than one block; if only one block is involved, simply set **LFACE2** = **LFACE1**.

LDIR1 The variable **LDIR1** is a single character (P or M) specifying the direction (P=plus, M=minus) along the **LDIR1** coordinate in **LBLOCK1** which is from the boundary surface patch directed towards the interior of the flow region. The specification of this variable is normally automatic when the boundary specification is applied to the external surface of a grid block (e.g., **LDIR1** = P when **L1LIM** = 1, and **LDIR1** = M when **L1LIM** = i_{max} , j_{max} , or k_{max}). The intent is to specify which side of the boundary surface plane the interior computational cells (non-phantom cells) lie on when the boundary condition is applied to a grid block.

-
- LDIR2** The variable **LDIR2** is a single character (P or M) specifying the direction (P=plus, M=minus) along the **LDIR2** coordinate in **LBLOCK2** which is towards the interior flow region from the boundary surface patch. This variable is only used in boundary specifications cases involving more than one mesh block. See notes on **LDIR1** for details. If the boundary specification involves only a single mesh block, then simply set **LDIR2 = LDIR1**.
- LSPEC1** The variable **LSPEC1** is a single character (I, J, K, L, or H) which implies some special information about the boundary condition specification. This parameter is boundary condition dependent. For boundary conditions involving more than one mesh block, it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. See the descriptions of **M2LIM** and **N2LIM** below and the boundary conditions **PATCH** and **EXITT** for additional details and specific examples.
- LSPEC2** The variable **LSPEC2** is a single character (I, J, K, L, or H) which implies some special information about the boundary condition specification. See the notes on **LSPEC1** above.
- L1LIM** The variable **L1LIM** is an integer specifying the index of the constant mesh face determined by **LFACE1** to which the boundary condition should be applied in block **LBLOCK1**.
- L2LIM** The variable **L2LIM** is an integer specifying the index of the constant mesh face determined by **LFACE2** from which the boundary condition data is derived in block **LBLOCK2**.
- M1LIM1** The variable **M1LIM1** is an integer representing the initial index of the first remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an *i*, *j*, *k* constant surface, the variables **M1LIM1**, **M1LIM2**, **N1LIM1**, and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. The indices specified in **M1LIM1** and **M1LIM2** must be given in increasing order (**M1LIM1 < M1LIM2**).
- Natural order is defined as follows: if **LFACE1=I**, then the variables **M1LIM1** and **M1LIM2** refer to the extent in the *j* direction and the variables **N1LIM1** and **N1LIM2** refer to the extent in the *k* direction. If **LFACE1=J**, then the variables **M1LIM1** and **M1LIM2** refer to the extent in the *i* direction and the variables **N1LIM1** and **N1LIM2** refer to the extent in the *k* direction. If **LFACE1=K**, then the variables **M1LIM1** and **M1LIM2** refer to the extent in the *i* direction and the variables **N1LIM1** and **N1LIM2** refer to the extent in the *j* direction.
- M1LIM2** The variable **M1LIM2** is an integer representing the final index of the

first remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an i , j , or k constant surface, the variables **M1LIM1**, **M1LIM2**, **N1LIM1**, and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. The indices specified in **M1LIM1** and **M1LIM2** must be given in increasing order.

N1LIM1 The variable **N1LIM1** is an integer representing the initial index of the second remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an i , j , or k constant surface, the variables **M1LIM1**, **M1LIM2**, **N1LIM1**, and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. The indices specified in **N1LIM1** and **N1LIM2** must be given in increasing order. For boundaries on 2-D mesh blocks, this must always be 1.

N1LIM2 The variable **N1LIM2** is an integer representing the final index of the second remaining grid coordinate direction to which the boundary condition is applied in block **LBLOCK1**. Since the boundary specification applies to either an i , j , or k constant surface, the variables **M1LIM1**, **M1LIM2**, **N1LIM1**, and **N1LIM2** determine the extent of the patch in the remaining coordinate directions. The remaining coordinate directions for block **LBLOCK1** are specified in the natural order. The indices specified in **N1LIM1** and **N1LIM2** must be given in increasing order. For boundaries on 2-D mesh blocks, this must always be 2.

M2LIM1 The variable **M2LIM1** is an integer representing the initial index of the grid coordinate direction in block **LBLOCK2** corresponding to the first remaining coordinate in block **LBLOCK1**. For boundary conditions involving more than one mesh block, it is possible that the connection between blocks may involve connections between different grid surfaces, and that the indices in block **LBLOCK2** correspond to a different coordinate in block **LBLOCK1**. The variables **M2LIM1** and **M2LIM2** control the indices in the **LSPEC1** direction in block **LBLOCK2** which correspond to the indices determined by **M1LIM1** and **M1LIM2** in block **LBLOCK1**. The user should note that it is possible for the **M2** limits to be in decreasing order. If only a single mesh block is involved in the boundary specification, set **M2LIM1** = **M1LIM1**.

M2LIM2 The variable **M2LIM2** is an integer representing the final index of the grid coordinate direction in block **LBLOCK2** corresponding to the first remaining coordinate in block **LBLOCK1**. The variables **M2LIM1** and **M2LIM2** control the indices in the **LSPEC1** direction in block

LBLOCK2 which correspond to the indices determined by **M1LIM1** and **M1LIM2** in block **LBLOCK1**. The user should note that it is possible for the **M2** limits to be in decreasing order. If only a single mesh block is involved in the boundary specification, set **M2LIM2 = M1LIM2**.

N2LIM1 The variable **N2LIM1** is an integer representing the initial index of the grid coordinate direction in block **LBLOCK2** corresponding to the second remaining coordinate in block **LBLOCK1**. The variables **N2LIM1** and **N2LIM2** control the indices in the **LSPEC2** direction in block **LBLOCK2** which correspond to the indices determined by **N1LIM1** and **N1LIM2** in block **LBLOCK1**. The user should note that it is possible for the **N2** limits to be in decreasing order. If only a single mesh block is involved in the boundary specification, set **N2LIM1 = N1LIM1**. For boundary data on 2-D mesh blocks, this must always be 1.

N2LIM2 The variable **N2LIM2** is an integer representing the final index of the grid coordinate direction in block **LBLOCK2** corresponding to the second remaining coordinate in block **LBLOCK1**. The variables **N2LIM1** and **N2LIM2** control the indices in the **LSPEC2** direction in block **LBLOCK2** which correspond to the indices determined by **N1LIM1** and **N1LIM2** in block **LBLOCK1**. The user should note that it is possible for the **N2** limits to be in decreasing order. If only a single mesh block is involved in the boundary specification, set **N2LIM2 = N1LIM2**. For boundary data on 2-D mesh blocks, this must always be 2.

Some boundary condition specifications require additional data beyond that incorporated in the boundary specification line. In these cases, described in detail for the specific boundary types later in this section, the additional data is included immediately after the boundary specification line. A sample *ADPAC* boundary data file containing several keywords is listed in Figure 3.11 followed by the detailed descriptions of all the boundary conditions available in *ADPAC*.

```

# THIS IS A COMMENT LINE
#
This is also a comment line since there are no keywords.
#
# B      L L L L L L L L L L L M M N N M M N N C
# C      B B F F D D S S 1 2 1 1 1 1 2 2 2 2 O
# T      L L A A I I P P L L L L L L L L L L M
# Y      O O C C R R E E I I I I I I I I I I M
# P      C C E E 1 2 C C M M M M M M M M M M E
# E      K K 1 2      1 2      1 2 1 2 1 2 1 2 N
#          1 2                                          T
#-----
#
#---> The next two lines do the periodic boundary at K=1, K=17
#
PATCH   1 1 K K P M I J 1 17 1 49 1 17 1 49 1 17 K=1
PATCH   1 1 K K M P I J 17 1 1 49 1 17 1 49 1 17 K=KL
#
#---> Hub surface is at J=1
#
SSIN     1 1 J J P P S S 1 1 1 49 1 17 1 49 1 17 Hub
#
#---> Next two lines define the blade surfaces at K=1, K=17
#
SSIN     1 1 K K P P S S 1 1 17 33 1 17 17 33 1 17 K=1
SSIN     1 1 K K M M S S 17 17 17 33 1 17 17 33 1 17 K=KL
#
#---> Set the inflow data at I=1
#
INLETT   1 1 I I P P S S 1 1 1 17 1 17 1 17 1 17 INL
NDATA
3
      RAD      PTOT      TTOT      BETAR      BETAT      CHI
0.100000  1.000000  1.000000  0.000000  0.000000  1.000000
0.300000  1.000000  1.000000  0.000000  0.000000  1.000000
0.500000  1.000000  1.000000  0.000000  0.000000  1.000000
#
#---> Set the exit flow data at I=49 (Note that the exit static pressure
#      is set here: this determines the blade loading and the flow rate
#
EXITT    1 1 I I M M H H 49 49 1 17 1 17 1 17 1 17 INL
PEXIT
1.200000
#
#---> Define the case surface at J=17
#
SSIN     1 1 J J M M S S 17 17 1 49 1 17 1 49 1 17 Case
#
ENDDATA

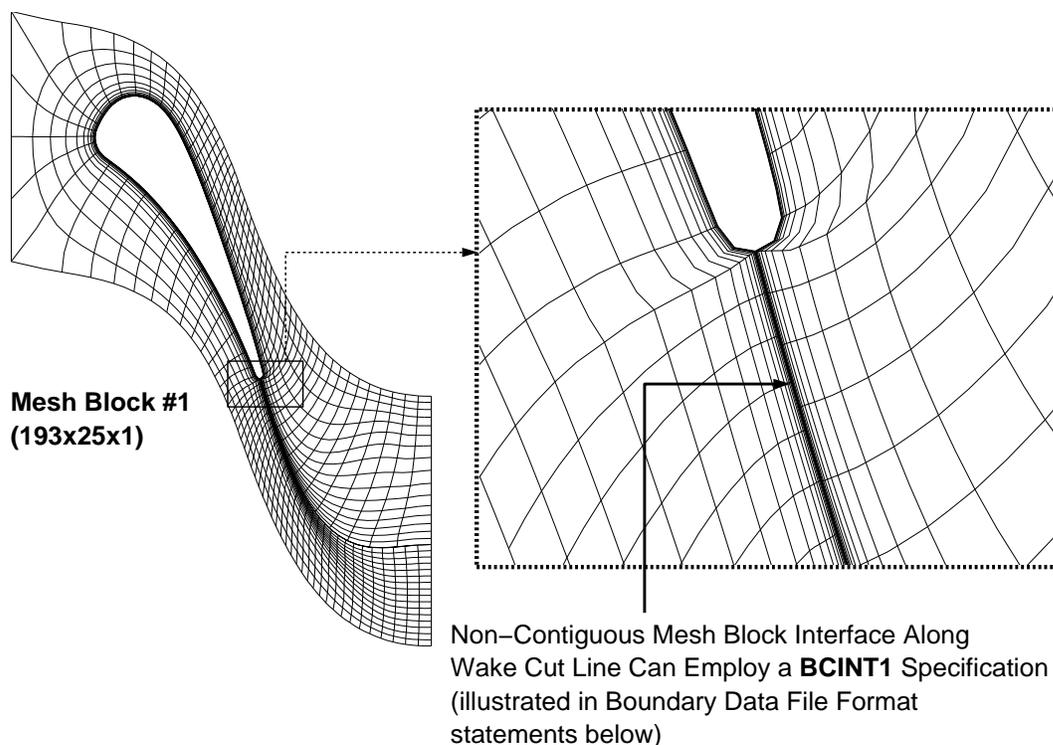
The code will never reach this line due to the ENDDATA above.

```

Figure 3.11: Sample *ADPAC* boundary condition data file specification (*case.boundata*).

BCINT1

BCINT1 Type Non-Contiguous Mesh Block Interface Patching Scheme



Application

The **BCINT1** specification is used in any application involving neighboring mesh blocks with a non-contiguous grid line to grid line interface in one coordinate direction. The interface must be contiguous in the other direction. For interfaces where both indices are misaligned the boundary condition **PINT** should be employed. **BCINT1** patches one block to one other block by interpolation along the non-contiguous index.

The example graphic above illustrates a two-dimensional mesh system used to predict the flow through a turbine vane passage. The C-type mesh utilizes a noncontiguous wake cut line as shown in the trailing edge detail. The **BCINT1** specification is applied along either side of the wake cut line to permit communication of flow variables across the noncontiguous mesh interface. Here, the interpolation direction is i , and part of the block is patched to itself. Note that the i index increases in different directions at the wake cut line. **BCINT1** can handle interpolation along any index, regardless of the orientation of the mating surface.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BCINT1** boundary condition are given below:

```
BCINT1  1  1  J  J  P  P  L  L  1  1  1  33  1  2 193 177  1  2
      IDIRNT1 IDIRNT2
           I      I
      ISHFTDR  DSHIFT
           2      0.0
```

```
BCINT1  1  1  J  J  P  P  L  L  1  1 177 193  1  2 33  1  1  2
      IDIRNT1 IDIRNT2
           I      I
      ISHFTDR  DSHIFT
           2      0.0
```

Note that a complete **BCINT1** specification generally requires two **BCINT1** statement lines in the boundary data file. In the example above, the first specification provides the inter-block communication for one side of the C-grid wake cut, while the second specification provides the communication for the other side of the C-grid wake cut. It is a common error to under-specify a **BCINT1** boundary by only providing a single line per interface.

Description

The **BCINT1** boundary statement provides a means for block-to-block communication for cases involving neighboring mesh boundaries which share a common surface, but are non-contiguous in one grid index. **BCINT1** can be applied to either stationary or rotating block interfaces, but the results are physically correct only if both blocks are rotating at the same speed. (The **BCPRR** specification should be used for cases with relatively rotating blocks.) A proper **BCINT1** boundary is specified much like a **PATCH** boundary. The **LFACE1** and **LFACE2** determine which faces are mated together. **BCINT1** also requires the specification of additional information. The second line in a **BCINT1** specification is a comment line, normally labeling the variables **INTDIR1** and **INTDIR2**. The third line defines the variables **INTDIR1** and **INTDIR2** as either I, J, or K, depending on the direction of interpolation for the receiving and sending blocks, respectively. One mesh restriction to note is that **BCINT1** allows only one interpolation direction for each side of the interface (e.g., the mesh *must* align in the other direction defining the boundary condition area). For mesh interfaces where neither direction is aligned, see **PINT**.

The fourth line is a comment line normally labeling the variables **ISHFTDR** and **DSHIFT**. The fifth line defines the values for the variables **ISHFTDR** and **DSHIFT**. These variables provide a mechanism for shifting the boundary in one of the three coordinate directions (x , y , z for Cartesian flows, or x , r , θ for cylindrical flows). The value of **ISHFTDR** identifies the variable to be shifted ($1-x$, $2-y$, $3-z$ for Cartesian and $1-x$, $2-r$, $3-\theta$ for cylindrical) and the value of **DSHIFT** is the increment by which the *sending* boundary is shifted (normalized in the same manner as the mesh coordinates) to mate to the receiving boundary. **BCINT1** expects that the two sides of the interface lie on a common

or parallel physical surface, but the grid itself may not have both sides of the interface in the same physical location. The most common use for this feature is a noncontiguous periodic boundary for a single passage turbomachinery blade row solution.

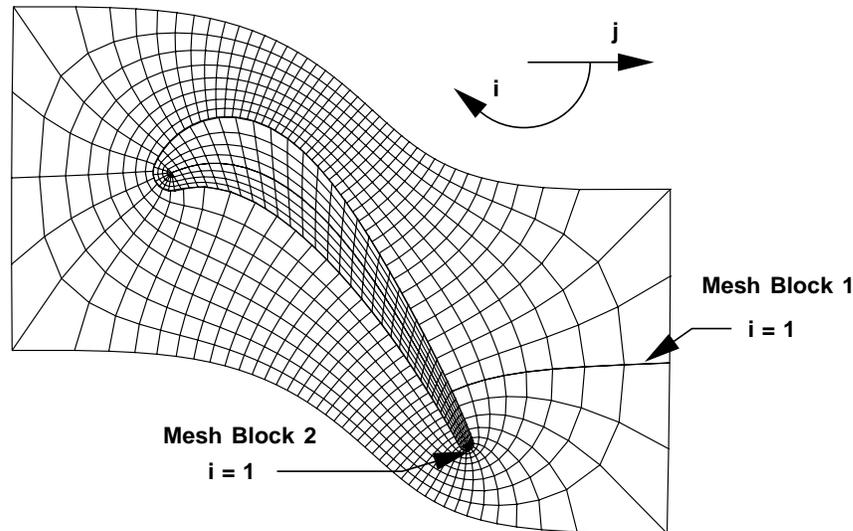
For the case of a non-contiguous periodic boundary in a turbine blade row solution, for example, `ISHFTDR` would be 3 (shift in the θ direction) and the amount of the shift defined by `DSHIFT` would be the circumferential spacing of the blade rows in radians. The `ISHFTDR` and `DSHIFT` variables are provided to allow the user to temporarily shift the physical location of the “sending” blocks to the “receiving” blocks physical location. The blocks are assumed to be contiguous in the remaining index. The `M2LIM` or `N2LIM` variables are specified much as they would be for a **PATCH** specification. The exception is that the number of points spanned by the limits in the direction of interpolation need not be equal.

The search routine which determines the interpolation stencil assumes that the mating grid lines are piecewise linear approximations to the same curve in the interpolation direction. A global search is performed for the proper mating cell of the first index. The closest cell to the point of interest is taken as the mating cell. A localized search is performed for the mating cells of the remaining points. The local search starts at the mating cell of the preceding point and searches along the mating boundary until the mating cell containing the new point is found. In the event that the mating cell is not found before the upper limit is reached in the mating block, the search continues from the lower limit in the mating block. This implies two things: the physical domain of the interpolation must be the same in the two blocks, and the domain is assumed to be periodic if the search routine goes past an endpoint.

Restrictions/Limitations

The **BCINT1** boundary specification is restricted to mesh interfaces which when shifted according to `ISHFTDR` and `DSHIFT` lie on a common surface (no significant overlap).

Generally, endpoints of the interpolated region in the two blocks should be coincident. There is at least one exception to this rule based on the above description of the search routine. In the case of concentric O-grids, the endpoints of the two blocks may be misaligned as shown in the figure below. The interpolation routine will find the appropriate stencil for each point because the grids are periodic.



The *COARSEN* program cannot properly handle subdivision of mesh boundaries involving a **BCINT1** or **BCINTM** specification.

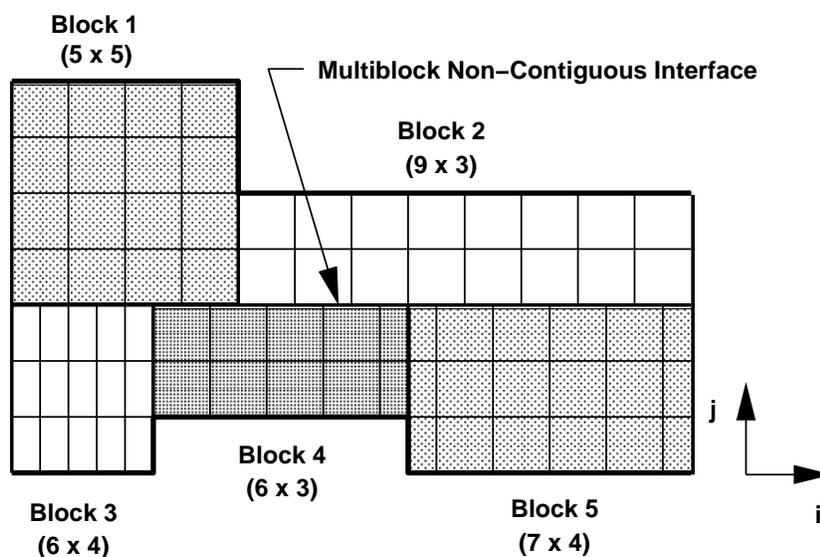
The **BCINT1** condition reduces to a **PATCH** condition if the mating blocks are actually contiguous. However, due to the linear interpolation used in **BCINT1**, the scheme does not maintain either global or local conservation of flow variables across a non-contiguous mesh interface. Also, there is no dissipation across a **BCINT1** boundary condition, but there is for a **PATCH** boundary condition (e.g., solutions may differ).

Common Errors

- Failure to provide 2 **BCINT1** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1**, **M2LIM2**, **N2LIM1**, **N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Incorrectly specified index for the interpolation direction for **LBLOCK1** or **LBLOCK2**.
- Attempt to use **BCINT1** for a boundary which has 2 misaligned coordinates (see **PINT**).
- Attempt to use **BCINT1** for boundaries which are not monotonic along the interpolated index.
- Attempt to use *COARSEN* for a problem involving a **BCINT1** boundary specification.

BCINTM

BCINTM Type Non-Contiguous Mesh Block Interface Patching Scheme



Non-contiguous mesh block interface involving multiple blocks requires a **BCINTM** specification (illustrated in boundary data file format statements below)

Application

The **BCINTM** specification is used in any application involving neighboring mesh blocks with a non-contiguous mesh interface in one coordinate direction. The interface must be contiguous in the remaining coordinate direction. **BCINTM** provides a mechanism whereby noncontiguous boundaries involving groups of blocks may be coupled to other groups of blocks by interpolation along the non-contiguous index. **BCINTM** is a multi-block version of **BCINT1**.

The example graphic above illustrates a two-dimensional mesh system used to predict the flow through a stepped duct passage. The grid was constructed with a non-contiguous interface between the various blocks on the top and bottom of the duct. The **BCINTM** specification is applied along either side of the interface to permit communication of flow variables along the interface. Here, the interpolation direction is the i coordinate direction.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative

graphic for the **BCINTM** boundary condition are given below:

```

BCINTM      1  3  J  J  P  M  I  K  1  4  1  5  1  1  1  5  1  1
INTDIR1 INTDIR2 - DIRECTION OF INTERPOLATION
      I      I
      ISHFTDR  DSHIFT
            2    0.0
NBLINT2 - NUMBER OF LBLOCK2 BLOCKS
      3
NBLDAT  LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
      3      J    M    4      1      6      1      1
      4      J    M    3      1      6      1      1
      5      J    M    4      1      7      1      1
NBLINT1 - NUMBER OF LBLOCK1 BLOCKS
      2
LBLK1RR LFACE1 LDIR1 L1LIM M1LIM1 M1LIM2 N1LIM1 N1LIM2
      1      J    P    1      1      5      1      1
      2      J    P    1      1      9      1      1

BCINTM      3  1  J  J  M  P  I  K  4  1  1  5  1  1  1  5  1  1
INTDIR1 INTDIR2 - DIRECTION OF INTERPOLATION
      I      I
      ISHFTDR  DSHIFT
            2    0.0
NBLINT2 - NUMBER OF LBLOCK2 BLOCKS
      2
NBLDAT  LFACE2 LDIR2 L2LIM M2LIM1 M2LIM2 N2LIM1 N2LIM2
      1      J    P    1      1      5      1      1
      2      J    P    1      1      9      1      1
NBLINT1 - NUMBER OF LBLOCK1 BLOCKS
      3
LBLK1RR LFACE1 LDIR1 L1LIM M1LIM1 M1LIM2 N1LIM1 N1LIM2
      3      J    M    4      1      6      1      1
      4      J    M    3      1      6      1      1
      5      J    M    4      1      7      1      1

```

Note that a complete **BCINTM** specification generally requires two **BCINTM** statement lines in the boundary data file. In the example above, the first specification provides the inter-block communication for the upper blocks along the interface, while the second specification provides the communication for the lower blocks along the interface. It is a common error to under-specify a **BCINTM** boundary by only providing a single line per interface.

Description

The **BCINTM** boundary statement provides a means for block-to-block communication for cases involving neighboring mesh boundaries which share a common surface, but are

non-contiguous in one grid index. A proper **BCINTM** boundary is specified much like a **BCINT1** boundary, except that all of the blocks involved with a particular interface are specified in a table on both sides of the interface. A large amount of additional data is required for each **BCINTM** specification.

In the example application, a noncontiguous mesh block interface lies between blocks 1, 2 and blocks 3, 4, 5. A single pair of **BCINTM** specifications is all that is required to completely couple the mesh blocks along this interface, in spite of the fact that 5 mesh blocks are involved in the overall boundary definition. The key to this compact specification is that each **BCINTM** specification includes tables of data which specify which blocks lie along the receiving side of the interface (where the boundary data is being applied) and which blocks lie along the sending side of the interface (where the boundary data is derived).

Immediately following the **BCINTM** boundary specification line is a series of multi-line segments which define the details of the boundary coupling. The first segment consists of 4 lines, and describes some general characteristics of the interpolation along the noncontiguous boundary. The second segment is the table describing the “sending” blocks from which the boundary data is extracted. The third segment is a table describing the “receiving” blocks where the boundary data is eventually interpolated and applied.

The second line in a **BCINTM** specification is a comment line, normally labeling the variables **INTDIR1** and **INTDIR2**. The third line defines the variables **INTDIR1** and **INTDIR2** as either *I*, *J*, or *K*, depending on the direction of interpolation for the receiving and sending blocks, respectively. One mesh restriction to note is that **BCINTM** allows only one interpolation direction for each side of the interface. The fourth line is a comment line normally labeling the variables **ISHFTDR** and **DSHIFT**. The fifth line defines the values for the variables **ISHFTDR** and **DSHIFT**. These variables provide a mechanism for shifting the boundary in one of the three coordinate directions (*x, y, z* for Cartesian flows, or *x, r, θ* for cylindrical flows). Refer to **BCINT1** for descriptions on specifying **ISHFTDR** and **DSHIFT**.

The sixth line in the specification is a comment normally labeling the variable **NBLINT2**, and the seventh line specifies the number of blocks associated with the **LBLOCK2** side of the interface (the “sending” blocks). The eighth line is again a comment normally labeling the variables **NBLDAT**, **LFACE2**, **LDIR2**, **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1**, and **N2LIM2**. The next **NBLINT2** lines define the table containing the limits, directions, and faces for each of the **LBLOCK2** blocks. For each block in this table, **LFACE2** defines the coordinate face upon which the interface lies (**I**, **J**, or **K**), and **LDIR2** defines the direction (**P** for plus, or **M** for minus). **L2LIM** defines the value of the **LFACE2** coordinate upon which the surface is located, and **M2LIM1**, **M2LIM2**, and **N2LIM1**, and **N2LIM2** define the extent of the remaining coordinates for each of the **NBLINT2** blocks in their “natural” order.

Following the table for the **LBLOCK2** side of the interface, there is a comment line normally labeling the variable **NBLINT1**, followed by a line specifying the number of blocks on the **LBLOCK1** side of the interface. Next a comment line labeling the variables **L1LIM**, **M1LIM1**, **M1LIM2**, **N1LIM1**, and **N1LIM2** is given. Finally, a table consisting of **NBLINT1** lines defining the **LBLOCK1** side (“receiving” blocks) information similar to the **LBLOCK2** (“sending” blocks) table is specified.

BCINTM creates a single interpolation stencil from all of the blocks in the **LBLOCK2**

table. This stencil must be monotonic in the **INTDIR2** direction. Thus, the blocks in the **LBLOCK2** must be specified in the order they occur physically, and the limits must be specified so that they form a continuous line. The block numbers and extents identified in the first line of the **BCINTM** specification should match the first entry in each of the respective **LBLOCK** tables.

As with **BCINT1**, the search routine which determines the interpolation stencil assumes that the mating grid lines are piecewise linear approximations to the same curve in the interpolation direction. Additional details of the search algorithm are included with **BCINT1**.

Restrictions/Limitations

The **BCINTM** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap). Generally, endpoints of the interpolated region in the two blocks should be coincident. As with **BCINT1**, there is at least one exception to this rule based on the above description of the search routine. In the case of concentric O-grids, the endpoints of the two blocks may be misaligned (see the **BCINT1** description for details). The interpolation routine will find the appropriate stencil for each point because the grids are periodic.

The *COARSEN* program cannot properly handle subdivision of mesh boundaries involving either a **BCINT1** or **BCINTM** specification. The **BCINTM** condition reduces to a **PATCH** condition if the mating blocks are actually contiguous. However, due to the linear interpolation used in **BCINTM**, the scheme does not maintain either global or local conservation of flow variables across a non-contiguous mesh interface. The **BCINTM** condition is the only non-contiguous patching routine for multiple blocks and will run in either serial or parallel *ADPAC* calculations.

Common Errors

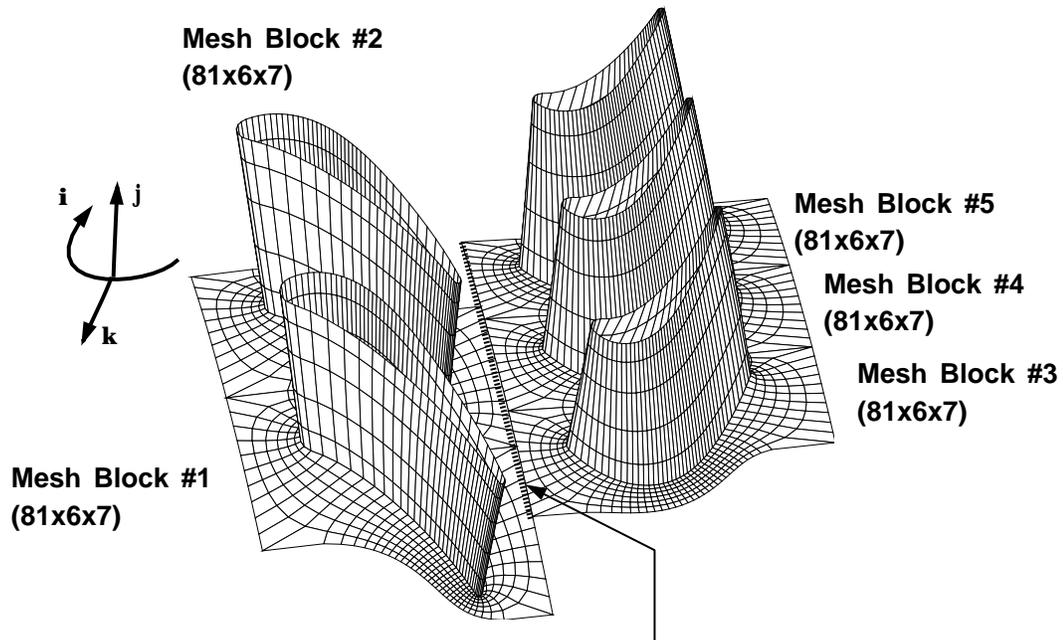
- Failure to provide 2 **BCINTM** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1**, **M2LIM2**, **N2LIM1**, **N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Incorrectly specified index for the interpolation direction for **LBLOCK1** or **LBLOCK2**.
- Attempt to use **BCINTM** for a boundary which has 2 misaligned coordinates.
- Attempt to use **BCINTM** for boundaries which are not monotonic along the interpolated index.
- Incorrect ordering of the **LBLOCK1** or **LBLOCK2** table of data.
- Attempt to use **BCINTM** for interfaces with multiple interpolation directions on the same side of the interface.
- Attempt to use **BCINTM** for interfaces with multiple **LFACE** or **LDIR** requirements in the **LBLOCK1** table of data.

BOUNDATA KEYWORDS

- Attempt to use *COARSEN* for a problem involving a **BCINTM** boundary specification.

BCPRM

Boundary Condition Procedure for Patched Relatively Rotating Mesh Blocks with Multiple Specifications



Relatively Rotating Mesh Block Interface Between Grids in Adjacent Blade Rows Can Employ a **BCPRM** Specification (illustrated in Boundary Data File Format statements below)

Application

The **BCPRM** specification is used in application involving neighboring relatively rotating mesh blocks, such as in rotor/stator interaction problems.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BCPRM** boundary condition and a simple outline of the mesh topography are given below. Note that blocks 1 and 2 require multiple **BCPRM** entries in the data tables due to the location of the O-grid cut line. The topography below depicts a multiple passage 3-D O-grid system for a turbine stage.

```
BCPRM      1  3  K  K  M  M  I  J  7  7  1  6  1  6  36  46  1  6
THPER
          0.41887903
```

BOUNDATA KEYWORDS

NBCPRR - NUMBER OF BLOCKS IN LBLOCK2 TABLE OF BCPRM SPECIFICATION

3

LBLOCK2B	LFACE2B	LDIR2B	L2LIMB	M2LIM1B	M2LIM2B	N2LIM1B	N2LIM2B
3	K	M	7	36	46	1	6
4	K	M	7	36	46	1	6
5	K	M	7	36	46	1	6

NRRDAT - NUMBER OF BLOCKS IN LBLOCK1 TABLE OF BCPRM SPECIFICATION

4

LBLOCK1B	LFACE1B	LDIR1B	L1LIMB	M1LIM1B	M1LIM2B	N1LIM1B	N1LIM2B
1	K	M	7	6	1	1	6
1	K	M	7	81	76	1	6
2	K	M	7	6	1	1	6
2	K	M	7	81	76	1	6

BCPRM 3 1 K K M M I J 7 7 36 46 1 6 6 1 1 6
THPER

0.41887903

NBCPRR - NUMBER OF BLOCKS IN LBLOCK2 TABLE OF BCPRM SPECIFICATION

4

LBLOCK2B	LFACE2B	LDIR2B	L2LIMB	M2LIM1B	M2LIM2B	N2LIM1B	N2LIM2B
1	K	M	7	6	1	1	6
1	K	M	7	81	76	1	6
2	K	M	7	6	1	1	6
2	K	M	7	81	76	1	6

NRRDAT - NUMBER OF BLOCKS IN LBLOCK1 TABLE OF BCPRM SPECIFICATION

3

LBLOCK1B	LFACE1B	LDIR1B	L1LIMB	M1LIM1B	M1LIM2B	N1LIM1B	N1LIM2B
3	K	M	7	36	46	1	6
4	K	M	7	36	46	1	6
5	K	M	7	36	46	1	6

Note that a complete **BCPRM** specification generally requires at least two **BCPRM** statement lines in the boundary data file. In the example above, the first specification provides the inter-block communication for the meshes representing blade row 1 from the meshes representing blade row 2, and the second specification provides the communication for the meshes representing blade row 2 from the meshes representing blade row 1. It is a common error to under-specify a **BCPRM** boundary by only providing a single line per interface.

Description

The **BCPRM** statement is an extension of the **BCPRR** statement to include the specification of multiple LBLOCK1 patches. As with **BCPRR**, the **BCPRM** statement specifies that a time-space interpolation utilizing data from several neighboring mesh blocks is to be performed to determine the boundary data for the LBLOCK1 patches. See the discussion of **BCPRR** for details about specifying the LBLOCK2 table of data and restrictions on the use of **BCPRM**.

BCPRM differs from **BCPRR** only in the following way: an additional table of values allows multiple **LBLLOCK1** patches to be specified. One advantage of **BCPRM** is clearly visible in the above example: only two boundary specifications are required to patch the two blade rows together, compared to seven specifications using **BCPRR**. Another, less obvious advantage is that **BCPRM** executes much faster than **BCPRR** in a parallel computing environment. Any **BCPRM** specification can be equally represented as a series of **BCPRR** specifications.

The additional table of data associated with the **LBLLOCK1** patches in a **BCPRM** statement is essentially the same as the table for the **LBLLOCK2** patches (see **BCPRR** for additional details). A comment line is followed by a line containing the number of patches in the **LBLLOCK1** row. Another comment line is followed by the specification of the limits on each **LBLLOCK1** patch. One restriction on the use of **BCPRM** is that all of the **LBLLOCK1** patches must share a common **LFACE** and **LDIR**. This requirement can be met by the use of multiple **BCPRM** or **BCPRR** specifications.

It should be noted that **BCPRM** is limited to similarly oriented row 1 blocks. In particular, the face and direction of all blocks in row 1 are taken to be the same as those specified for **LBLLOCK1**. **BCPRR** may be used in cases requiring more generality.

Restrictions/Limitations

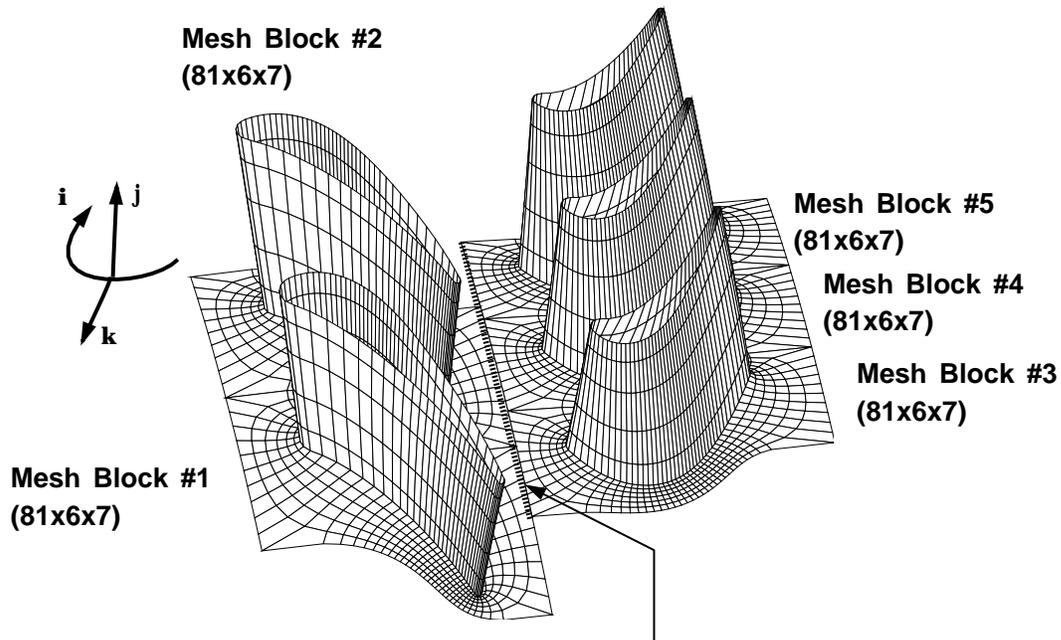
The **BCPRM** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap), and have common axial and radial mesh coordinates. The mesh must satisfy the coordinate restrictions listed in the table above. The **LBLLOCK1** table of patches must share a common face and direction as noted above. The **BCPRM** procedure is only applicable to 3-D mesh systems.

Common Errors

- Failure to provide 2 **BCPRM** statements for each interface
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1B**, **M2LIM2B**, **N2LIM1B**, **N2LIM2B** do not correctly define the interface extents on blocks **LBLLOCK1** and **LBLLOCK2B**)
- Attempt to use **BCPRM** on a 2-D mesh block.
- Attempt to use **BCPRM** at an interface between two Cartesian solution meshes.
- Meshes do not satisfy coordinate restrictions listed above.
- Meshes have dissimilar axial and radial coordinates at the interface.
- Neighboring blade row 1 segments not listed in increasing theta coordinate.
- Application of **BCPRM** to mesh interfaces which do not share a common surface or which have excess overlap.

BCPRR

Boundary Condition Procedure for Patched Relatively Rotating Mesh Blocks



Relatively Rotating Mesh Block Interface Between Grids in Adjacent Blade Rows Requires a **BCPRR** Specification (illustrated in Boundary Data File Format statements below)

Application

The **BCPRR** specification is used in application involving neighboring relatively rotating mesh blocks, such as in rotor/stator interaction problems.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BCPRR** boundary condition and a simple outline of the mesh topography are given below. Note that blocks 1 and 2 require multiple **BCPRR** specifications due to the location of the O-grid cut line.

```
BCPRR      1  3  K  K  M  M  I  J  7  7  1  6  1  6  36  46  1  6
THPER
```

```
0.41887903
```

```
NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION
```

3
 LBLOCK2B LFACE2B LDIR2B L2LIMB M2LIM1B M2LIM2B N2LIM1B N2LIM2B
 3 K M 7 36 46 1 6
 4 K M 7 36 46 1 6
 5 K M 7 36 46 1 6

BCPRR 1 3 K K M M I J 7 7 76 81 1 6 36 46 1 6
 THPER

0.41887903

NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION

3
 LBLOCK2B LFACE2B LDIR2B L2LIMB M2LIM1B M2LIM2B N2LIM1B N2LIM2B
 3 K M 7 36 46 1 6
 4 K M 7 36 46 1 6
 5 K M 7 36 46 1 6

BCPRR 2 3 K K M M I J 7 7 1 6 1 6 36 46 1 6
 THPER

0.41887903

NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION

3
 LBLOCK2B LFACE2B LDIR2B L2LIMB M2LIM1B M2LIM2B N2LIM1B N2LIM2B
 3 K M 7 36 46 1 6
 4 K M 7 36 46 1 6
 5 K M 7 36 46 1 6

BCPRR 2 3 K K M M I J 7 7 76 81 1 6 36 46 1 6
 THPER

0.41887903

NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION

3
 LBLOCK2B LFACE2B LDIR2B L2LIMB M2LIM1B M2LIM2B N2LIM1B N2LIM2B
 3 K M 7 36 46 1 6
 4 K M 7 36 46 1 6
 5 K M 7 36 46 1 6

BCPRR 3 1 K K M M I J 7 7 36 46 1 6 6 1 1 6
 THPER

0.41887903

NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION

4
 LBLOCK2B LFACE2B LDIR2B L2LIMB M2LIM1B M2LIM2B N2LIM1B N2LIM2B
 1 K M 7 6 1 1 6
 1 K M 7 81 76 1 6
 2 K M 7 6 1 1 6
 2 K M 7 81 76 1 6

BCPRR 4 1 K K M M I J 7 7 36 46 1 6 6 1 1 6

BOUNDATA KEYWORDS

THPER

0.41887903

NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION

4

LBLOCK2B	LFACE2B	LDIR2B	L2LIMB	M2LIM1B	M2LIM2B	N2LIM1B	N2LIM2B
1	K	M	7	6	1	1	6
1	K	M	7	81	76	1	6
2	K	M	7	6	1	1	6
2	K	M	7	81	76	1	6

BCPRR 5 1 K K M M I J 7 7 36 46 1 6 6 1 1 6

THPER

0.41887903

NBCPRR - NUMBER OF BLOCKS IN BCPRR SPECIFICATION

4

LBLOCK2B	LFACE2B	LDIR2B	L2LIMB	M2LIM1B	M2LIM2B	N2LIM1B	N2LIM2B
1	K	M	7	6	1	1	6
1	K	M	7	81	76	1	6
2	K	M	7	6	1	1	6
2	K	M	7	81	76	1	6

Note that a complete **BCPRR** specification generally requires at least two **BCPRR** statement lines in the boundary data file. In the example above, the first four specifications provide the inter-block communication for the meshes representing blade row 1 from the meshes representing blade row 2, and the final three specifications provide the communication for the meshes representing blade row 2 from the meshes representing blade row 1. It is a common error to under-specify a **BCPRR** boundary by only providing a single line per interface.

Description

The **BCPRR** statement specifies that a time-space interpolation utilizing data from several neighboring mesh blocks is to be performed to determine the boundary data for block **LBLOCK1**. This time-space interpolation provides the computational means of performing time-dependent predictions of the flow through multiple blade row turbomachines. In order to perform this type of calculation, several conditions must be satisfied. For calculations involving blade rows with dissimilar blade counts, it is necessary to model several blade passages per blade row. The number of blade passages modeled should be chosen such that the overall circumferential span of each blade row is identical. This restriction implies that the blade counts should be reducible to simple integer ratios (1:2, 3:4, etc.) to avoid the need for modeling an excessive number of blade passages.

In the illustrative graphic above, if we seek a solution for a single stage turbomachine involving two blade rows with blade counts of 30 and 45, respectively (reduced blade ratio of 2:3), then the simulation would require 2 blade passages for the first blade row and 3 passages from the second blade row, such that the overall circumferential pitch for either blade row is $\frac{2\pi}{15}$ (the number 15 chosen as the largest common factor in the blade counts 30 and 45).

The second restriction is that the interface separating two adjacent blade rows be a surface of revolution, and that meshes along this interface have common axial and radial grid distributions. This restriction simplifies the time-space interpolation provided by the **BCPRR** specification.

This boundary condition requires the specification of additional data, as shown in the format descriptor above. The variable following the label **THPER** defines the total circumferential span of the neighboring blade row's mesh representation in radians. Using the blade counts given in the previous example, the circumferential span represented in each blade row is determined by $\frac{2\pi}{15}$, and therefore **THPER** should be 0.41887903. The variable following the next label, **NBCPRR**, indicates the number of mesh blocks through which the time-space interpolation is to be performed.

In the example above, if we are applying the **BCPRR** specification to the first blade row, then **NBCPRR** should be 3, since there are 3 mesh block surfaces in the neighboring blade row defining the circumferential extent of the first blade row. The numbers immediately following the labels **LBLOCK2B**, **LFACE2B**, **LDIR2B**, **L2LIMB**, **M2LIM1B**, **M2LIM2N**, **N2LIM1B**, and **N2LIM2B** represent the values of **LBLOCK2**, **LFACE2**, **LDIR2**, **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1**, and **N2LIM2** for *each* of the individual **NBCPRR** segments used in the construction of the circumferential data array. The **NBCPRR** segments and their respective circumferential direction indices (either **M2LIM1B**, **M2LIM2B** or **N2LIM1B**, **N2LIM2B** *must* be listed in order of increasing theta coordinate). Due to the complex nature of the circumferential interpolation operator, this boundary condition is restricted to specific mesh configurations. The following chart describes the permitted mesh configurations for the **BCPRR** specification:

LFACE1 (Block #1 Face)	LFACE2 (Block #2 Face)	Circumferential Coordinate Direction	Grids Must be Aligned in this Coordinate
-----	-----	-----	-----
I	I or K	K or I	J
J	J only	K	I
K	I or K	K or I	J

In the example described above, if block numbers 1 and 2 are the block numbers for the first blade row, and block numbers 3, 4, and 5 are the block numbers for the second blade row, then the **BCPRR** specification for each of the first blade row blocks would set **THPER** = 0.41887903, **NBCPRR** = 2, and **LBLOCK2B** = 3, 4, 5. In a similar manner, the specification for each of the blocks in the second blade row would set **THPER** = 0.41887903, **NBCPRR** = 4 (due to the use of the O-type mesh for each airfoil, the extent of the interface between the two blade rows requires 2 mesh surfaces from each of the blade row 1 airfoil meshes), and **LBLOCK2B** = 1, 1, 2, 2. It should be mentioned that this specification is somewhat unique in that more than one block is involved in the boundary specification, therefore the variable **LBLOCK2** is essentially ignored; however, since the blocks specified by the **LBLOCK2B** variable are assumed to be essentially duplicate representations of neighboring blade passages, the variables **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1**, and **N2LIM2** are also ignored.

The time-space interpolation is constructed to permit the relative rotation of blocks

representing neighboring blade rows and therefore cannot be applied to Cartesian solution meshes. The simulation is initiated from the relative position of the blocks at the start of the calculation $t=0$. The interpolation scheme is area weighted to maintain a conservative property across the interface between the relatively rotating mesh blocks.

Restrictions/Limitations

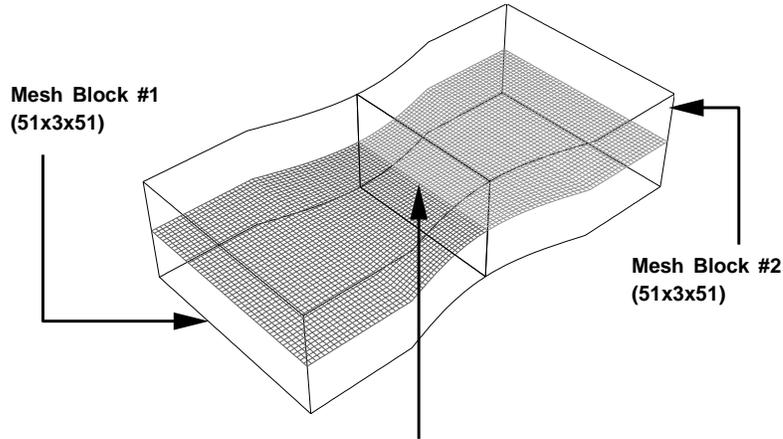
The **BCPRR** boundary specification is restricted to mesh interfaces which lie on a common surface, and have common axial and radial mesh coordinates. The mesh must satisfy the coordinate restrictions listed in the table above. The **BCPRR** procedure is only applicable to 3-D mesh systems.

Common Errors

- Failure to provide 2 **BCPRR** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1B**, **M2LIM2B**, **N2LIM1B**, **N2LIM2B** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2B**).
- Attempt to use **BCPRR** on a 2-D mesh block.
- Meshes do not satisfy coordinate restrictions listed above.
- Meshes have dissimilar axial and radial coordinates at the interface.
- Neighboring blade row segments not listed in increasing theta coordinate.
- Application of **BCPRR** to mesh interfaces which do not share a common surface, or which have excessive overlap.
- **BCPRR** runs very slowly on multiple processors - use **BCPRM** instead.

BDATIN

File Read In Mesh Interface Patching Scheme



**BDATIN/BDATOU Combination Used
to Provide Disk Read/Write of Boundary
Data for Interblock Communication
Between Blocks #1 and #2**

Application

The **BDATIN** specification is used to read in boundary data from an external file. This file may be either be created by an external program, or by the *ADPAC* boundary specification **BDATOU**. The application illustrated above indicates an application of the **BDATIN/BDATOU** combination for a two block nozzle flow case. The **BDATIN/BDATOU** combination is applied to the interface between the two mesh blocks and is equivalent to a **PATCH** specification, except that the inter-block communication is accomplished through disk read/write rather than shared memory communication.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BDATIN** boundary condition are given below:

```
BDATIN 1 2 I I M P J K 51 1 1 3 1 51 1 3 1 51
FILENAME
bc.12.data
```

```
BDATIN 2 1 I I P M J K 1 51 1 3 1 51 1 3 1 51
FILENAME
bc.21.data
```

Note that a complete **BDATIN** specification requires the specification of a filename from which the boundary data is read.

Description

The **BDATIN** statement is utilized to provide boundary data for a mesh surface through external file specification. During the application of a **BDATIN** specification, an external file is opened, and phantom cell boundary data are read in for the appropriate computational cells. The external file data may be created by an external program, or through the application of a **BDATOU** specification. A coupled set of **BDATIN/BDATOU** specifications can be effectively used to replace a **PATCH** boundary specification. In this case, inter-block communication would be achieved through external file read/write rather than shared memory. If the **BDATIN/BDATOU** combination is used to replace an equivalent **PATCH** condition, it should be noted that both the **BDATIN** and **BDATOU** specifications should be written in the same manner as the **PATCH** statement. In other words, the **BDATIN** data is read in to the **LBLOCK1** block on the mesh cells defined by **L1LIM**, **M1LIM1**, **M1LIM2**, **N1LIM1** and **N1LIM2**, and the **BDATOU** data is written out from the **LBLOCK2** block on the mesh cells defined by **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1** and **N1LIM2**. The **BDATIN/BDATOU** routines were developed in conjunction with early parallelization studies for the *ADPAC* to permit inter-block communication via shared disk file read/write operations. The routines are now considered useful for coupling the *ADPAC* code with other codes capable of providing or using specified boundary data.

A **BDATIN** specification requires two additional lines in addition to the normal boundary data file descriptor, as shown above. The first additional line is simply a label, while the second line indicates the file name relative to the current directory from which data will be read in for this particular boundary condition. **BDATIN** reads the file at every stage of the Runge-Kutta solver and separate files are required for the coarser grids when using multi-grid.

Restrictions/Limitations

The **BDATIN/BDATOU** coupling scheme is restricted to mesh interfaces which have a one-to-one mesh point correspondence. Other restrictions appropriate for the **PATCH** boundary condition also apply to mesh coupling using the **BDATIN/BDATOU** scheme. Data provided in the external file for the **BDATIN** specification must represent cell-centered data and must be normalized consistently with the *ADPAC* flow variable nondimensionalization procedure.

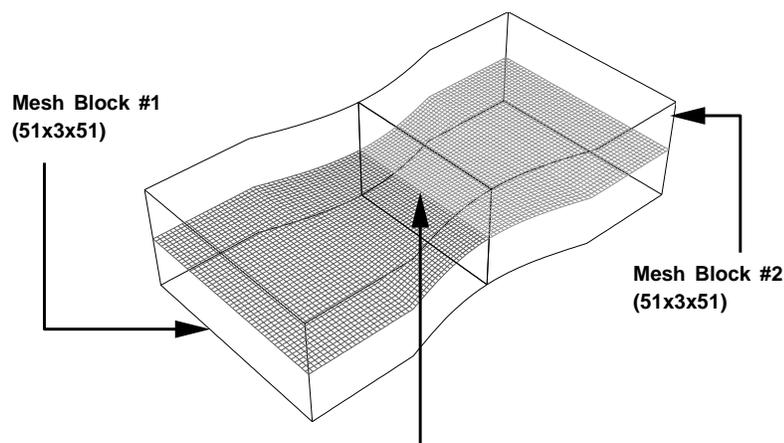
The **BDATIN/BDATOU** boundary conditions should not be run using multi-grid and allowing *ADPAC* to automatically generate the coarse grid **BDATIN/BDATOU** statements, because *ADPAC* will repeat the same filename when creating the coarser levels of multi-grid. Therefore during the multi-grid cycle, the files containing the boundary data would be overwritten by different levels of mesh size. In such a case the coarse mesh would read only a portion of the fine mesh data, or vice versa, the fine mesh would run out of data if it tried to read the coarse level file. This can be avoided by either running *ADPAC* without multi-grid or by “hand-writing” the coarser boundary conditions (see input keyword **FRDMUL**).

Common Errors

- Failure to provide file name for **BDATIN** boundary data file specification.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1**, **M2LIM2**, **N2LIM1**, **N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- **BDATIN/BDATOU** coupling scheme desired, but only one of the **BDATIN/BDATOU** specifications provided.
- **BDATIN/BDATOU** coupling scheme boundary specification for a periodic boundary is applied to a non-periodic mesh.
- **BDATIN/BDATOU** coupling scheme boundary specification applied to a spatially periodic Cartesian geometry using the cylindrical coordinate solution scheme or vice versa (results in incorrect spatial periodicity relationships). The **BDATIN/BDATOU** coupling scheme boundary specifications for Cartesian geometries must use the Cartesian solution algorithm in *ADPAC* (see input variable **FCART**).
- Using multi-grid and letting *ADPAC* automatically generate the coarse grid **BDATIN** statements. In this case the coarse mesh would read only a portion of the fine mesh data.

BDATOU

File Write Out Mesh Interface Patching Scheme



**BDATIN/BDATOU Combination Used
to Provide Disk Read/Write of Boundary
Data for Interblock Communication
Between Blocks #1 and #2**

Application

The **BDATOU** specification is used to write out boundary data to an external file. This file may either be utilized by an external program, or by the *ADPAC* boundary specification **BDATIN**. The application illustrated above indicates an application of the **BDATIN/BDATOU** combination for a two block nozzle flow case. The **BDATIN/BDATOU** combination is applied to the interface between the two mesh blocks and is equivalent to a **PATCH** specification, except that the inter-block communication is accomplished through disk read/write rather than shared memory communication.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **BDATOU** boundary condition are given below:

```
BDATOU 1 2 I I M P J K 51 1 1 3 1 51 1 3 1 51
FILENAME
bc.12.data
```

```
BDATOU 2 1 I I P M J K 1 51 1 3 1 51 1 3 1 51
FILENAME
bc.21.data
```

Note that a complete **BDATOU** specification requires the specification of a filename from which the boundary data is read.

Description

The **BDATOU** statement is utilized to export boundary data for a mesh surface through external file specification. During the application of a **BDATOU** specification, an external file is opened, and near boundary cell-centered data are written in for the appropriate computational cells. The external file data may then be utilized by an external program, or through the application of a **BDATIN** specification. Refer to **BDATIN** for additional detail in coupling these two boundary conditions.

A **BDATOU** specification requires two additional lines in addition to the normal boundary data file descriptor, as shown above. The first additional line is simply a label, while the second line indicates the file name relative to the current directory to which data will be written out for this particular boundary condition. **BDATOU** writes the file at every stage of the Runge-Kutta solver and separate files are required for the coarser grids when using multi-grid.

Restrictions/Limitations

The **BDATIN/BDATOU** coupling scheme is restricted to mesh interfaces which have a one-to-one mesh point correspondence. Other restrictions appropriate for the **PATCH** boundary condition also apply to mesh coupling using the **BDATIN/BDATOU** scheme. Data provided in the external file for the **BDATOU** specification represents near-boundary cell-centered data and is normalized consistently with the *ADPAC* flow variable nondimensionalization procedure.

The **BDATIN/BDATOU** boundary conditions should not be run using multi-grid and allowing *ADPAC* to automatically generate the coarse grid **BDATIN/BDATOU** statements, because *ADPAC* will repeat the same filename when creating the coarser levels of multi-grid. Therefore during the multi-grid cycle, the files containing the boundary data would be overwritten by different levels of mesh size. In such a case the coarse mesh would read only a portion of the fine mesh data, or vice versa, the fine mesh would run out of data if it tried to read the coarse level file. This can be avoided by either running *ADPAC* without multi-grid or by “hand-writing” the coarser boundary conditions (see input keyword **FRDMUL**).

Common Errors

- Failure to provide file name for **BDATOU** boundary data file specification.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1**, **M2LIM2**, **N2LIM1**, **N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- **BDATIN/BDATOU** coupling scheme desired, but only one of the **BDATIN/BDATOU** specifications provided.

- **BDATIN/BDATOU** coupling scheme boundary specification for a periodic boundary is applied to a non-periodic mesh.
- **BDATIN/BDATOU** coupling scheme boundary specification applied to a spatially periodic Cartesian geometry using the cylindrical coordinate solution scheme or vice versa (results in incorrect spatial periodicity relationships) The **BDATIN/BDATOU** coupling scheme boundary specifications for Cartesian geometries must use the Cartesian solution algorithm in *ADPAC* (see input variable **FCART**).
- Using multi-grid and letting *ADPAC* automatically generate the coarse grid **BDATOU** statements. In this case the coarse mesh would write only a portion of the fine mesh data.

ENDDATA

Boundary Data File Read Terminator

Application

The **ENDDATA** statement causes the *ADPAC* boundary data file read utility to discontinue reading lines in the boundary data file and proceeds with normal code processing. Any lines following an **ENDDATA** statement in a boundary data file are ignored.

Boundary Data File Format

The boundary data file specification for an **ENDDATA** statement is given below:

```
ENDDATA
```

Note that the **ENDDATA** statement does not require the accompanying values (i.e., *LBLOCK1*, *LBLOCK2*, *LFACE1*, etc.) as do all other boundary data file keywords.

Description

The **ENDDATA** statement is utilized to provide a terminator for the boundary data file read sequence in the *ADPAC* code. Under normal operating conditions, the boundary data file is read in one line at a time and parsed to determine if a boundary data file keyword is present and uncommented on each line. When the end of the file is reached, the boundary data file read sequence stops, and normal processing continues as usual. In some cases, it may be desirable to terminate the boundary data file read sequence before the end of the file, and the **ENDDATA** statement is provided for this purpose. Whenever an **ENDDATA** statement is reached, the boundary data file read sequence is terminated, and all remaining lines in the boundary data file are ignored. The **ENDDATA** keyword is useful for debugging boundary condition problems, as whole portions of the boundary data file can be effectively eliminated by simply preceding the section with an **ENDDATA** statement.

Restrictions/Limitations

The **ENDDATA** keyword has no restrictions.

Common Errors

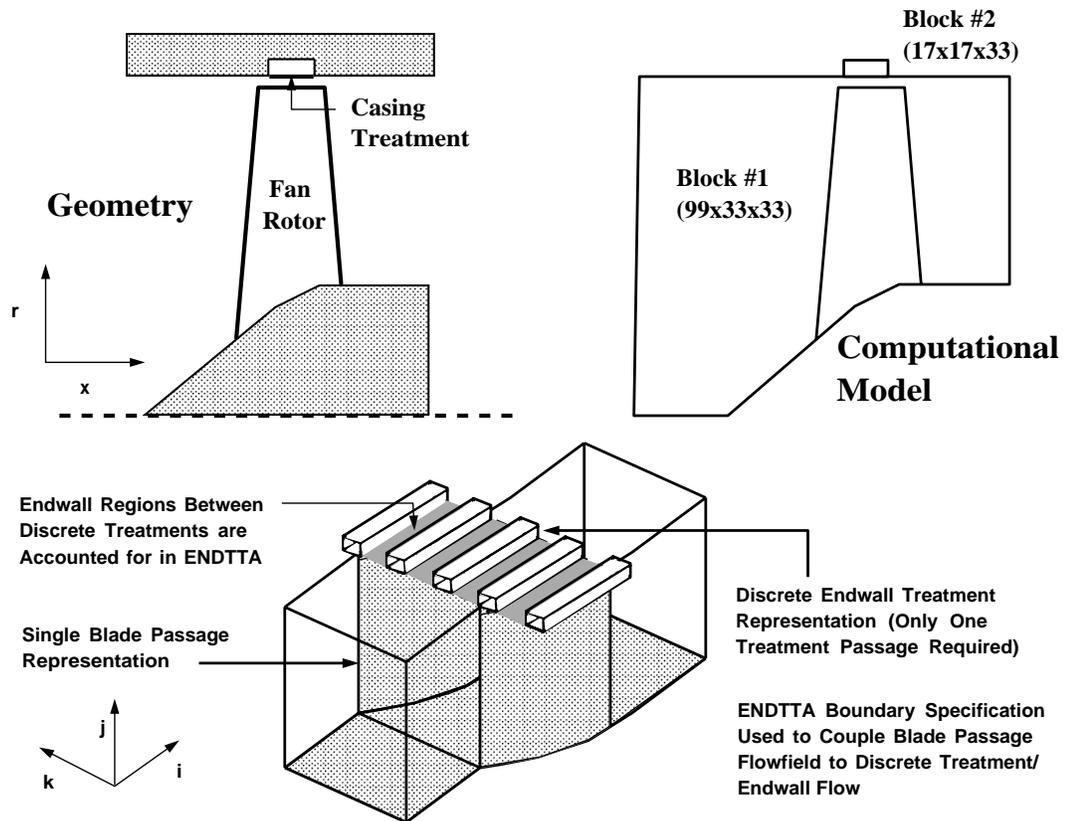
- Desired boundary conditions specifications following an **ENDDATA** statement are ignored.
- *ADPAC* complains because an insufficient number of boundary conditions were

BOUNDATA KEYWORDS

provided for the external boundaries of each mesh block (see input keyword **FBCWARN**).

ENDTTA

Endwall Treatment Time-Average Mesh Block Interface Patching Scheme



Application

The **ENDTTA** boundary specification was developed specifically to permit numerical prediction of turbomachinery airfoil blade row flows employing endwall treatments such as slots, grooves, or embedded bladed passages in a time-averaged fashion. The example graphic above illustrates a 3-D blocked mesh system for a turbofan engine fan rotor employing an axial slot casing treatment. The **ENDTTA** boundary specification employs a time-averaging operator (circumferential average of flow variables) between adjacent rotating and non-rotating mesh blocks to simulate the effects of the blade row/endwall treatment interaction. As such, it is possible to perform steady-state (representative of a time average) numerical analysis of turbomachinery blade passages and endwall treatments which have arbitrary blade passage/treatment passage count ratios.

Boundary Data File Format

BOUNDATA KEYWORDS

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **ENDTTA** boundary condition are given below:

```
ENDTTA    1  2  J  J  M  P  L  L  49   1  49  81   1  33   1  33   1  17
NTREAT RPMWALL TWALL
113      0.0      0.0
```

```
MBCAVG    2  1  J  J  P  M  I  K   1  49   1  33   1  17  49  81   1  33
NSEGS
1
BLK  LFACE2 LDIR2  L2LIM  M2LIM1 M2LIM2 N2LIM1 N2LIM2
1      J      M      49      49      81      1      33
```

Note that a complete **ENDTTA** specification generally requires a companion **MBCAVG** specification to complete the blade passage mesh/treatment passage mesh interface specification. In the example above, the first specification provides the inter-block communication for block 1 (blade passage mesh) from block 2 (treatment passage mesh) which ultimately accounts for the influence of the true endwall in the boundary specification. The second specification (**MBCAVG**) is applied to the treatment passage mesh boundary to simulate the time-average (circumferential average) of the neighboring blade passage mesh. It is a common error to under-specify an **ENDTTA** boundary by neglecting to specify the companion **MBCAVG**.

Description

The **ENDTTA** boundary statement provides a means for block-to-block communication for the prediction of the time-averaged flow for turbomachinery blade rows employing endwall treatments such as discrete slots, grooves, or embedded bladed passages. The boundary condition is restricted to j =constant mesh surfaces *only* and must possess aligned coordinates in the i direction, but have misaligned mesh points and extents in the circumferential (k) direction. The **ENDTTA** boundary specification is valid for 3-D cylindrical solution mesh blocks only.

The **ENDTTA** specification requires the specification of additional data, as shown in the format descriptor above. The first additional line following an **ENDTTA** specification is assumed to be a label for the variables NTREAT, RPMWALL, and TWALL. The next line contains the values for the variables NTREAT, RPMWALL, and TWALL. The variable NTREAT represents the total number of discrete treatments for the *entire* rotor. The analysis is normally performed for a single rotor blade passage and a single treatment blade passage, and the value of NTREAT is used to effectively set the circumferential spacing between discrete treatment passages. The next variable, RPMWALL, sets the value of the rotational speed of the endwall regions which separate the discrete treatments. Naturally, this also implies the rotational speed of the treatments themselves, and the value of RPMWALL in this context must be consistent with the value of RPM specified in the input file for the treatment passage mesh block. The third variable, TWALL, sets the thermal boundary condition for the wall segments separating the discrete treatment passages (see **SSVI**).

Restrictions/Limitations

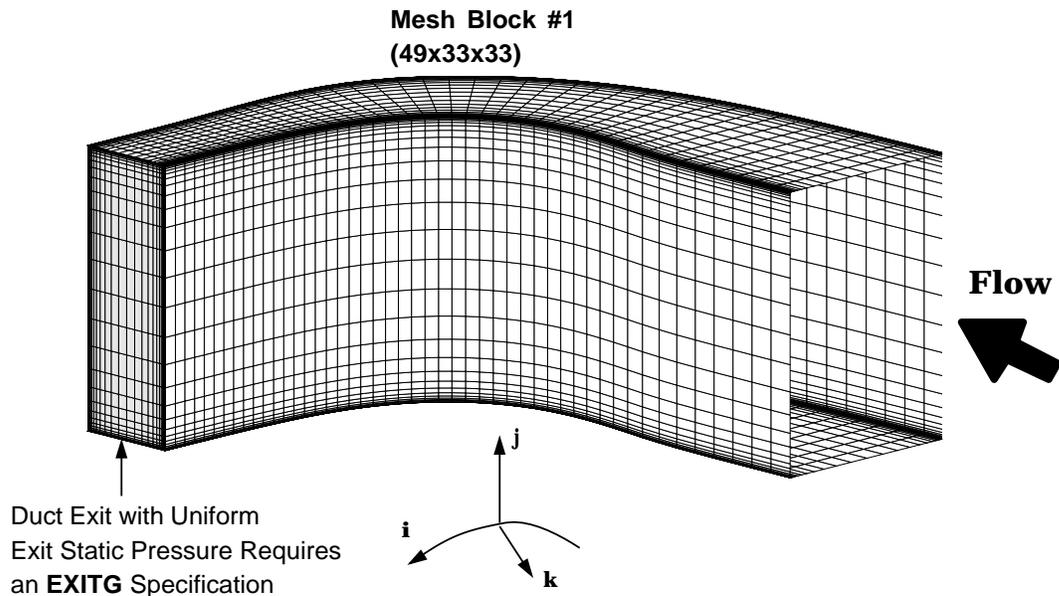
The **ENDTTA** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap). The **ENDTTA** procedure permits only that the k coordinates between adjacent mesh surfaces are misaligned. The **ENDTTA** procedure is only valid if applied to j =constant mesh surfaces. **ENDTTA** will *not* run across multiple processors in a parallel computing environment if the blocks are across separate processors. In the above example, blocks 1 and 2 would need to be on the same processor.

Common Errors

- Failure to provide a coupled pair of **ENDTTA** and **MBCAVG** statements for each interface.
- Failure to properly specify the values for **RPM**, **TWALL** and/or **NTREAT**.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1**, **M2LIM2**, **N2LIM1**, **N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Attempt to use **ENDTTA** for an i or k constant boundary.
- Attempt to use **ENDTTA** for a Cartesian solution mesh.
- Attempt to use **ENDTTA** for a boundary which has 2 misaligned coordinates.
- Attempt to use **ENDTTA** between blocks on different processors.

EXITG

Generic Outflow Boundary Condition



Application

The **EXITG** specification is used to impose a generic subsonic outflow boundary condition with a uniform exit static pressure. The example graphic above depicts a simple duct flow using a Cartesian-based H-grid, where the exit boundary plane is controlled by an **EXITG** specification.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITG** boundary condition is given below:

```
EXITG 1 1 I I M M J K 49 49 1 33 1 33 1 33 1 33
PEXIT
0.625
```

or the alternate specification including the mass flow specification:

```
EXITG 1 1 I I M M J K 49 49 1 33 1 33 1 33 1 33
PEXIT EMDOT PRELAX
0.625 40.0 0.001
```

Note that a complete **EXITG** specification requires two additional lines following the

EXITG boundary data file specification line. Failure to properly specify the data in these additional lines is a common **EXITG** specification error.

Description

The **EXITG** statement specifies that a generic, subsonic, uniform static pressure exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **EXITG** boundary condition should be applied for those cases where any other “specialized” exit boundary condition (i.e., **EXITT**, **EXITP**) does not apply. The **EXITG** boundary condition is also likely to be somewhat more efficient computationally than the other exit boundary condition procedures, at the expense of some physical simplification. **EXITG** may be used on any mesh face (*i*, *j*, or *k* constant) for either cylindrical or Cartesian-based solution schemes (see input variable **FCART**).

The **EXITG** procedure utilizes a Reimann invariant formulation to compute exit velocities based on a specified constant exit static pressure. Included in the **EXITG** procedure is a special correction scheme which forces the flow to pass out of the flow domain at the boundary. In other words, if the computed velocities result in a local inflow at the **EXITG** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point. This boundary condition requires the specification of additional data. The first additional line following the **EXITG** specification is assumed to be a label. The next line contains the value imposed for the variables **PEXIT** which represents the downstream exit static pressure ratio used in the **EXITG** characteristic solution sequence. The value of the **PEXIT** variable is the desired normalized downstream static pressure computed as:

$$PEXIT = \frac{P_{static,desired}}{P_{ref}}$$

where the variable P_{ref} is specified by the input variable **PREF**. It should be mentioned that for most geometries, the value of **PEXIT**, in combination with any inlet flow boundary conditions, will normally govern the resulting solution mass flow rate. Values of **PEXIT** < 0.0 are not permitted. As the value of **EXITP** is reduced, the flow through the boundary will ultimately choke, and further reductions of **EXITP** will no longer increase the mass flow through the boundary. Naturally, poor convergence or solution divergence can occur if **PEXIT** is too high or too low when compared to the rest of the flow domain. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

An alternate specification is provided for the **EXITG** boundary specification as shown in the sample application above. In this case, three values are included following the original boundary specification line. The alternate specification is provided as a means of achieving a desired mass flow rate through the bounding surface using the **EXITG** algorithm. The desired mass flow rate is achieved iteratively by incrementally adjusting the exit static pressure specification until the desired flow rate is achieved. Therefore, in this specification, the variable **PEXIT** described in detail above is the initial exit static pressure used in the iterative process, **EMDOT** represents the desired mass flow rate through the bounding surface in pounds mass, and **PRELAX** is a relaxation factor to stabilize the iterative process (values may range from 0.0 to 1.0, though poor convergence is likely for values larger than 0.1).

For applications to 2-D surface boundaries, the exit plane is projected to create a exit area over which to calculate the mass flow. For Cartesian flow calculations a unit depth (1.0 in mesh coordinates) is assumed for the third coordinate direction to determine the mass flow rate. For cylindrical flow calculations, the geometry is assumed to be axisymmetric and a multiple of 2π is used in the mass flow integration (the mass flow is computed as if the full circumference of the axisymmetric geometry were employed).

This procedure is not foolproof, and suffers from the fact that when a job is restarted, if an updated exit pressure is not inserted in the boundary data file, then the pressure-mass flow iterative process will start over. The *ADPAC* code will automatically determine when to employ the iterative process by detecting the presence of the additional boundary specification variables.

Restrictions/Limitations

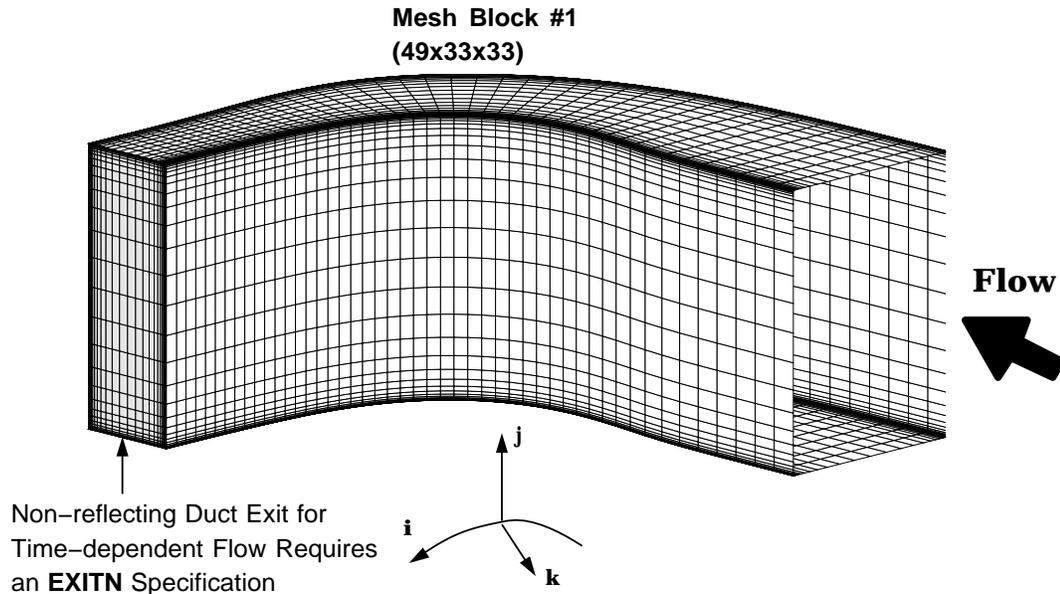
The **EXITG** boundary specification can be applied to both 3-D and 2-D mesh surfaces.

Common Errors

- Failure to specify the additional data value PEXIT.
- Improper specification of the alternate (mass flow) boundary scheme.
- Reductions in the value of PEXIT do not increase the mass flow rate because of flow choking.
- Value of PEXIT is too high (flow cannot get started).

EXITN

Non-Reflecting Outflow Boundary Condition



Application

The **EXITN** specification is used to impose a non-reflecting outflow boundary condition for time-dependent flow calculations where spurious numerical reflections normally imposed by other boundary procedures are undesirable. The example graphic above depicts a simple duct flow using a Cartesian-based H-grid, where the exit boundary plane is controlled by an **EXITN** specification. This boundary condition should be used for time-dependent flow calculations only.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITN** boundary condition is given below:

```
EXITN 1 1 I I M M J K 49 49 1 33 1 33 1 33 1 33
```

Description

The **EXITN** statement specifies that a non-reflecting exit flow boundary condition is to be applied to the mesh surface specified by LFACE1 on the block specified by LBLOCK1. The **EXITN** boundary condition should be used for time-dependent flow calculations where spurious numerical reflections which normally occur for other exit boundary conditions are

undesirable. **EXITN** may be used on any mesh face (i , j , or k constant) for either cylindrical or Cartesian-based solution schemes (see input variable **FCART**). The **EXITN** procedure utilizes a characteristic-based formulation described in Appendix A.

Restrictions/Limitations

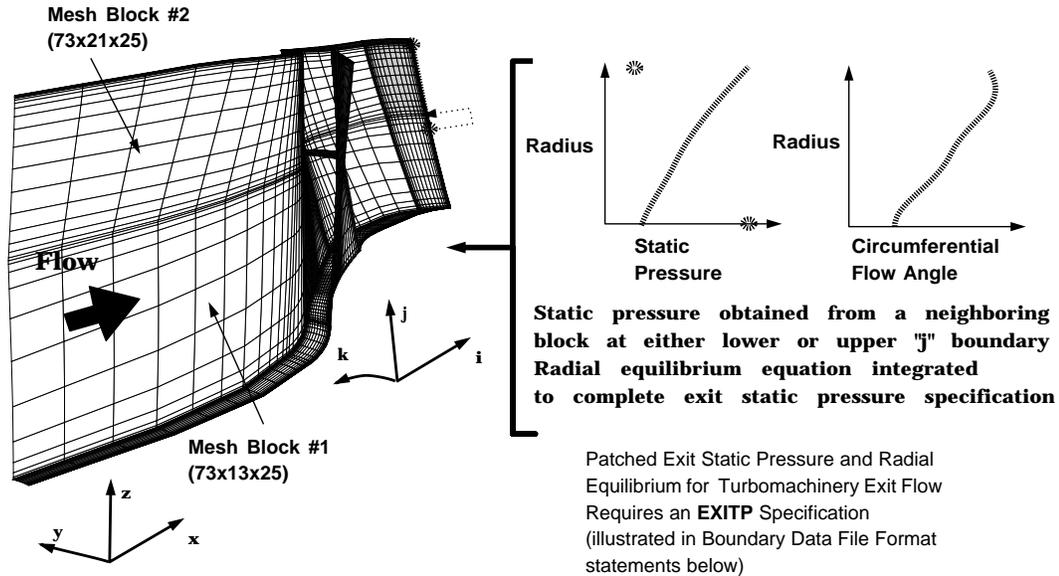
Because no pressure is explicitly specified using **EXITN**, it relies on whatever pressure is initially in the phantom cell. It is best to run without the **EXITN** boundary condition to set up the pressure field and then turn on the **EXITN** boundary with a restart so feasible values of pressure are set in the phantom cells. If no restart is used, the cells will be initialized to **PREF** which may not work well in some cases. The **EXITN** boundary specification should only be applied for time-dependent flow calculations restarted from a steady-state restart file. Steady-state problems can usually use the **EXT2DG**, **EXITX**, or **EXITT** boundary specification.

Common Errors

- Application of **EXITN** for a steady-state solution.

EXITP

Patched Turbomachinery Exit Boundary Condition



Application

The **EXITP** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium for mesh systems employing multiple blocks radially across the exit plane. The example graphic illustrates a two block 3-D mesh system used to predict the flow through a blade passage of a turbomachinery fan rotor with a part span shroud. The blocks are divided radially by the part span shroud, and as a result, the exit boundary plane consists of two mesh boundary segments. In order to employ a turbomachinery-based radial equilibrium exit flow boundary condition for this case, the **EXITT** specification is applied to the inner block 1 and the **EXITP** boundary condition is used for the outer block 2 to complete the inner to outer integration of the radial equilibrium equation across the mesh block interface.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITP** boundary condition is given below:

```
EXITP 2 1 I I M M L H 73 73 1 21 1 25 13 13 1 25
```

Note that the M2LIM1, M2LIM2 variables in the **EXITP** specification define a single *j* mesh line in mesh block LBLOCK2. Failure to properly regard this requirement is a common **EXITP** specification error. It should also be mentioned that **EXITP** also requires proper

specification of the **LSPEC1** variable for proper execution.

Description

The **EXITP** keyword specifies that a turbomachinery-based radial equilibrium patched exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **EXITP** boundary condition was specifically designed as an exit flow boundary procedure for axial and mixed flow turbomachinery geometries employing multiple, stacked mesh blocks (radially) at an exit boundary plane. The **EXITP** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the static pressure field at all points on the boundary surface. The initial static pressure specification used to initiate the radial equilibrium integration process is obtained from a neighboring mesh block.

As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation may be performed along the j coordinate direction of the mesh. Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process. The direction of integration, and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. Refer to **EXITT** for details of specifying **LSPEC1**.

The remaining flow variables on the **EXITP** boundary are updated by a Reimann invariant formulation based on the resulting local static pressure field. Included in the **EXITP** procedure is a special correction scheme which forces the flow to pass out of the flow domain. In other words, if the computed velocities result in a local inflow at the **EXITP** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point.

Restrictions/Limitations

The **EXITP** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a $j = \text{constant}$ mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. The **EXITP** boundary specification is restricted to 3-D mesh surfaces (2-D z - r mesh surfaces should use the **EXT2DP** boundary specification).

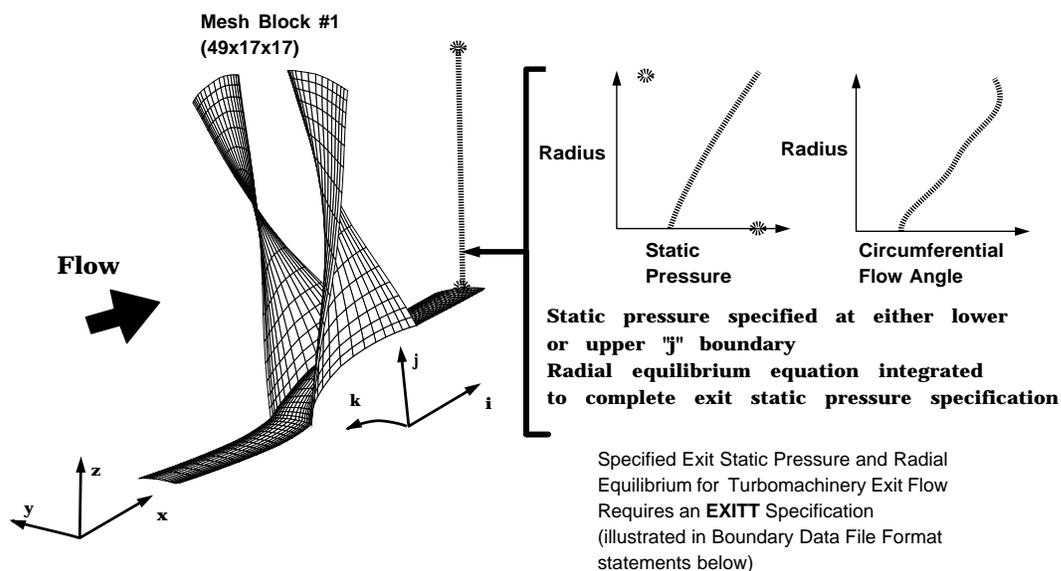
Common Errors

- Application of **EXITP** to a 2-D z - r mesh system.
- Failure to properly specify the **LSPEC2** variable.
- **M2LIM1** and **M2LIM2** differ.

- Radial-like direction of the mesh is not the j coordinate.
- Failure to properly specify the LSPEC1 variable on the boundary data file specification line.

EXITT

Turbomachinery Exit Boundary Condition



Application

The **EXITT** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium. The illustrative graphic above depicts an application of the **EXITT** outflow boundary condition for an H-type mesh for a turbomachinery fan rotor blade passage. The **EXITT** specification provides the radial variation of flow properties at the outflow boundary resulting from the application of a simplified form of the radial equilibrium equation.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITT** boundary condition is given below:

```
EXITT 1 1 I I M M L L 49 49 1 17 1 17 1 17 1 17
PEXIT
1.105
```

or the alternate specification including the mass flow specification:

```
EXITT 1 1 I I M M L L 49 49 1 17 1 17 1 17 1 17
PEXIT EMDOT PRELAX
1.105 13.7 0.001
```

Note that a complete **EXITT** specification requires two additional lines following the **EXITT** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **EXITT** specification error. It should also be mentioned that **EXITT** also requires proper specification of the **LSPEC1** variable for proper execution.

Description

The **EXITT** keyword specifies that a turbomachinery-based radial equilibrium exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **EXITT** boundary condition was specifically designed as an exit flow boundary procedure for axial and mixed flow turbomachinery geometries (pure radial flow turbomachinery exit flow boundaries usually use the **EXITG** boundary condition). The **EXITT** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the static pressure field at all points on the boundary surface.

As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation may be performed along the j coordinate direction of the mesh. Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process.

The direction of integration and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. If **LSPEC1** = L, for LOW, then **PEXIT** is applied to the lower (smallest value) of the j index, and the radial equilibrium equation is integrated outward (increasing j direction). If **LSPEC1** = H, for HIGH, then **PEXIT** is applied to the upper (largest value) of the j index, and the radial equilibrium equation is integrated inward (decreasing j direction). The remaining flow variables on the **EXITT** boundary are updated by a Reimann invariant formulation based on the resulting local static pressure field. Included in the **EXITT** procedure is a special correction scheme which forces the flow to pass out of the flow domain. In other words, if the computed velocities result in a local inflow at the **EXITT** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **EXITT** specification is assumed to be a label. The line following the **PEXIT** label contains the value of specified non-dimensional exit static pressure used to initiate the radial equilibrium integration procedure. Refer to **EXITG** for guidelines on **PEXIT** specification.

An alternate specification is provided for the **EXITT** boundary specification as shown in the sample application above. In this case, three values are included following the original boundary specification line. The alternate specification is provided as a means of achieving a desired mass flow rate through the bounding surface using the **EXITT** algorithm. Again, refer to **EXITG** for guidelines on using the mass flow specification option.

Restrictions/Limitations

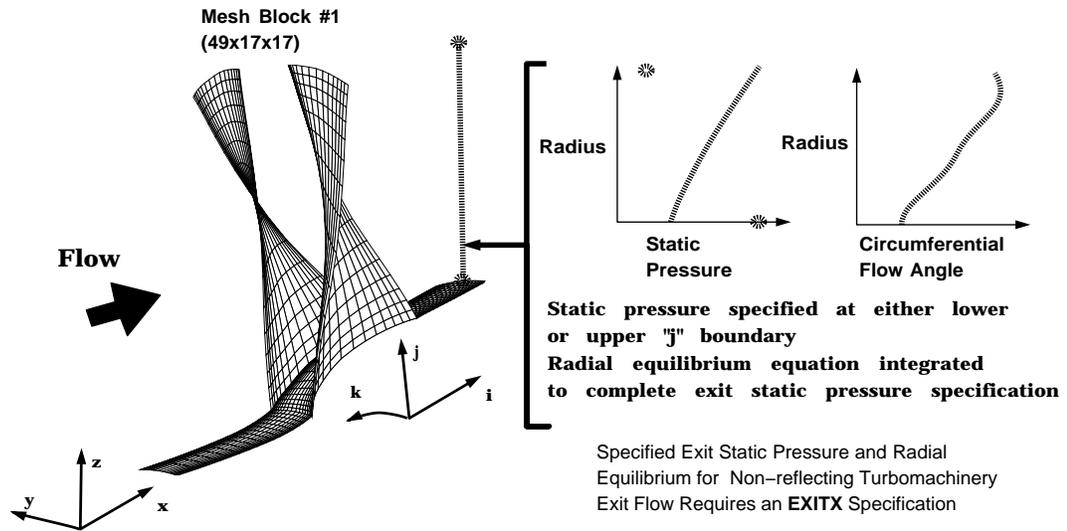
The **EXITT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a $j = \text{constant}$ mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. Examples of this type of mesh system can be found in the chapter defining standard configurations. The **EXITT** boundary specification is restricted to 3-D mesh surfaces (2-D z - r mesh surfaces should use the **EXT2DT** boundary specification).

Common Errors

- Application of **EXITT** to a 2-D z - r mesh system.
- Failure to specify the additional data value **PEXIT**.
- Improper specification of the alternate (mass flow) iterative scheme.
- Radial-like direction of the mesh is not the j coordinate.
- Mesh does not possess circumferential symmetry (axial, radial mesh coordinates vary in the circumferential coordinate direction).
- Failure to properly specify the **LSPEC1** variable on the boundary data file specification line.
- Value of **PEXIT** is too high (flow cannot get started).

EXITX

Non-Reflecting Steady State Turbomachinery Exit Boundary Condition



Application

The **EXITX** specification is used to impose a non-reflecting turbomachinery-based exit boundary condition based on radial equilibrium for steady flow calculations. The illustrative graphic above depicts an application of the **EXITX** outflow boundary condition for an H-type mesh for a turbomachinery fan rotor blade passage. The **EXITX** specification provides the radial variation of flow properties at the outflow boundary resulting from the application of a simplified form of the radial equilibrium equation.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXITX** boundary condition is given below:

```
EXITX 1 1 I I M M L L 49 49 1 17 1 17 1 17 1 17
PEXIT
1.105
```

Note that a complete **EXITX** specification requires two additional lines following the **EXITX** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **EXITX** specification error. It should also be mentioned that **EXITX** also requires proper specification of the LSPEC1 variable for proper execution.

Description

The **EXITX** keyword specifies that a non-reflecting turbomachinery-based radial equilibrium exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **EXITX** boundary condition was specifically designed as a non-reflecting exit flow boundary procedure for steady state analysis of axial and mixed flow turbomachinery geometries. The **EXITX** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the circumferentially averaged static pressure field at all points on the boundary surface. The circumferentially-averaged pressure at the exit boundary is forced to match the imposed static pressure. The use of the circumferential average permits flow properties to vary across the exit plane, while still maintaining the desired overall flow.

As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation is performed along the j coordinate direction of the mesh. Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process.

The direction of integration and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. If **LSPEC1** = L, for LOW, then **PEXIT** is applied to the lower (smallest value) of the j index, and the radial equilibrium equation is integrated outward (increasing j direction). If **LSPEC1** = H, for HIGH, then **PEXIT** is applied to the upper (largest value) of the j index, and the radial equilibrium equation is integrated inward (decreasing j direction). The remaining flow variables on the **EXITX** boundary are updated by a Reimann invariant formulation based on the resulting local static pressure field. Included in the **EXITX** procedure is a special correction scheme which forces the flow to pass out of the flow domain. In other words, if the computed velocities result in a local inflow at the **EXITX** boundary, no matter how small the magnitude of the inflow, the velocities are reset to zero at that point. This may result in a degradation of the non-reflective nature of this boundary condition in favor of a more stable boundary specification.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **EXITX** specification is assumed to be a label. The line following the **PEXIT** label contains the value of specified non-dimensional exit static pressure used to initiate the radial equilibrium integration procedure. Refer to **EXITG** for guidelines in specifying **PEXIT**.

Restrictions/Limitations

The **EXITX** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a $j = \text{constant}$ mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. The **EXITX** boundary specification is restricted to 3-D mesh surfaces. The **EXITX** boundary condition

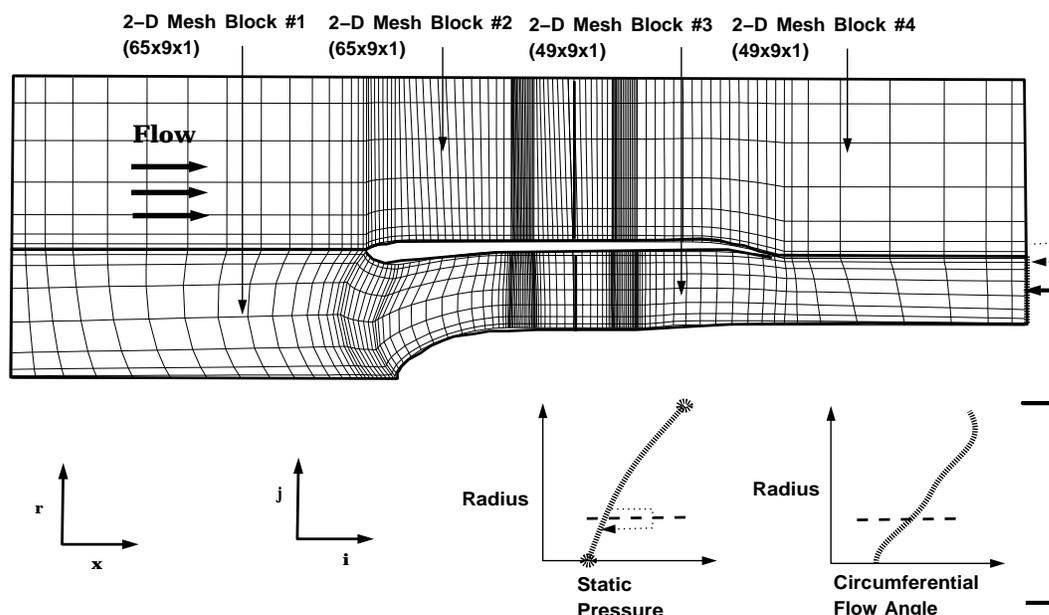
is only valid for steady-state flow calculations.

Common Errors

- Application of **EXITX** to a 2-D z - r mesh system.
- Failure to specify the additional data value **PEXIT**.
- Radial-like direction of the mesh is not the j coordinate.
- Mesh does not possess circumferential symmetry (axial, radial mesh coordinates vary in the circumferential coordinate direction).
- Failure to properly specify the **LSPEC1** variable on the boundary data file specification line.
- Value of **PEXIT** is too high (flow cannot get started).
- Application of **EXITX** for time-dependent flow calculations.

EXT2DP

2-D Patched Turbomachinery Exit Boundary Condition



Patched Exit Static Pressure and Radial Equilibrium for 2-D Turbomachinery Exit Flow Requires an **EXT2DP** Specification (illustrated in Boundary Data File Format statements below)

Static pressure specified at either lower or upper "j" boundary
Radial equilibrium equation integrated to complete exit static pressure specification

Application

The **EXT2DP** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium for 2-D z - r mesh systems employing multiple blocks radially across the exit plane. The example graphic above illustrates a four block mesh system used to predict the axisymmetric flow through a high bypass ratio turbofan engine geometry. The solution utilizes a specified freestream static pressure at the outer boundary of block 4, and an **EXT2DT** specification to integrate the radial equilibrium equation equation inward radially along the outflow boundary. In order to continue the radial equilibrium integration process across the block boundary between blocks 3 and 4, an **EXT2DP** specification is used to patch the two blocks.

Boundary Data File Format

The boundary data file specification for the 2-D mesh surface indicated in the illustrative graphic for the **EXT2DP** boundary condition is given below:

EXT2DP 4 3 I I M M L H 49 49 1 9 1 2 9 9 1 2

Note that the M2LIM1, M2LIM2 variables in the **EXT2DP** specification define a single j mesh line in mesh block LBLOCK2. Failure to properly regard this requirement is a common **EXT2DP** specification error. It should also be mentioned that **EXT2DP** also requires proper specification of the LSPEC1 variable for proper execution.

Description

The **EXT2DP** keyword specifies that a turbomachinery-based radial equilibrium patched exit flow boundary condition is to be applied to the mesh surface specified by LFACE1 on the 2-D block specified by LBLOCK1. The **EXT2DP** boundary condition was specifically designed as an exit flow boundary procedure for axial and mixed flow turbomachinery geometries employing multiple, stacked 2-D mesh blocks (radially) at an exit boundary plane. The **EXT2DP** boundary condition is the 2-D version of **EXITP**. The description of **EXITP** should be referenced for details in using **EXT2DP**.

Restrictions/Limitations

The **EXT2DP** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a $j = \text{constant}$ mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. The **EXT2DP** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **EXITP** boundary specification). By default, it is important that this type of boundary condition be carefully specified and the final solution carefully examined to ensure that the desired mesh patching be adequately satisfied. It is a common error to patch to the wrong grid, or the wrong end of the correct grid, and still obtain a converged solution.

Common Errors

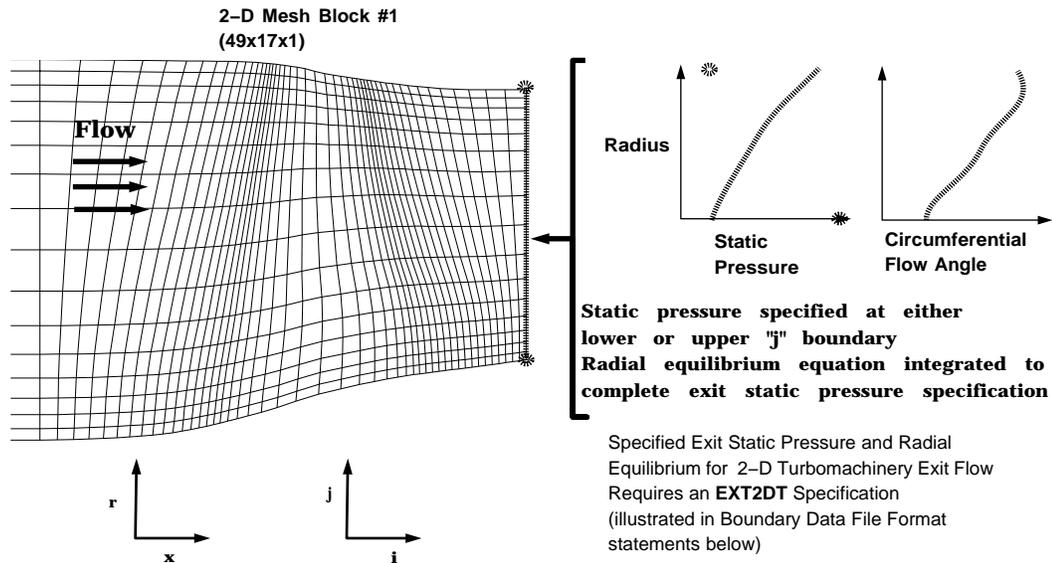
- Application of **EXT2DP** to a 3-D mesh system.
- Failure to properly specify the LSPEC1, LSPEC2 variables.
- M2LIM1 and M2LIM2 differ.
- Radial-like direction of the mesh is not the j coordinate.
- Failure to properly specify the LSPEC1 variable on the boundary data file specification line.
- **EXT2DP** specification patched to the wrong grid.
- **EXT2DP** specification patched to the wrong end of the correct grid.

BOUNDATA KEYWORDS

- **EXT2DP** boundary condition used but no **EXT2DT** or **EXITT** boundary condition specified.
- **EXT2DP** boundary condition called before **EXT2DT** (not required, but could cause problems).

EXT2DT

2-D Turbomachinery Exit Boundary Condition



Application

The **EXT2DT** specification is used to impose a turbomachinery-based exit boundary condition based on radial equilibrium for 2-D mesh blocks. The example graphic illustrated above depicts an **EXT2DT** specification for a 2-D (axisymmetric) flow solution for a turbomachinery blade row.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **EXT2DT** boundary condition is given below:

```
EXT2DT 1 1 I I M M L L 49 49 1 17 1 2 1 17 1 2
PEXIT
1.105
```

or the alternate specification including the mass flow specification:

```
EXT2DT 1 1 I I M M L L 49 49 1 17 1 2 1 17 1 2
PEXIT EMDOT PRELAX
1.105 13.7 0.001
```

Note that a complete **EXT2DT** specification requires two additional lines following the **EXT2DT** boundary data file specification line. Failure to properly specify the data in

these additional lines is a common **EXT2DT** specification error. It should also be mentioned that **EXT2DT** also requires proper specification of the **LSPEC1** variable for proper execution.

Description

The **EXT2DT** keyword specifies that a turbomachinery-based radial equilibrium exit flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the 2-D mesh block specified by **LBLCK1**. The **EXT2DT** boundary condition was specifically designed as an exit flow boundary procedure for 2-D axial and mixed flow turbomachinery geometries. Pure radial flow turbomachinery exit flow boundaries usually use the **EXT2DG** boundary condition. Due to the form of the radial equilibrium equation utilized in the **EXT2DG** routine, only cylindrical coordinate solution meshes are permitted to use this routine. The **EXT2DT** boundary condition procedure utilizes a combination static pressure specification and integration of the radial equilibrium equation to define the static pressure field at all points on the boundary surface. As a result of the complexity of this procedure, several mesh restrictions were imposed to simplify the application of this approach. The primary assumption is that the integration of the radial equilibrium equation may be performed along the j coordinate direction of the mesh. Hence, the j coordinate should be the radial-like direction. A single specification of static pressure is required at either the maximum or minimum extreme of the j coordinate of the boundary surface in order to initiate the integration process. The direction of integration, and location of application of the specified exit static pressure are determined by the **LSPEC1** variable in the calling sequence. Refer to **EXITT** and **EXITG** for details on specifying the integration direction and **PEXIT**, respectively.

An alternate specification is provided for the **EXDT2DT** boundary specification as shown in the sample application above. In this case, three values are included following the original boundary specification line. The alternate specification is provided as a means of achieving a desired mass flow rate through the bounding surface using the **EXT2DT** algorithm. Refer to **EXITG** for guidelines in correctly specifying the exit mass flow.

Restrictions/Limitations

The **EXT2DT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate, and is therefore not valid on a $j = \text{constant}$ mesh plane. This is required in order to properly integrate the radial equilibrium equation to complete the exit static pressure specification. The **EXT2DT** boundary specification is restricted to 2-D mesh surfaces.

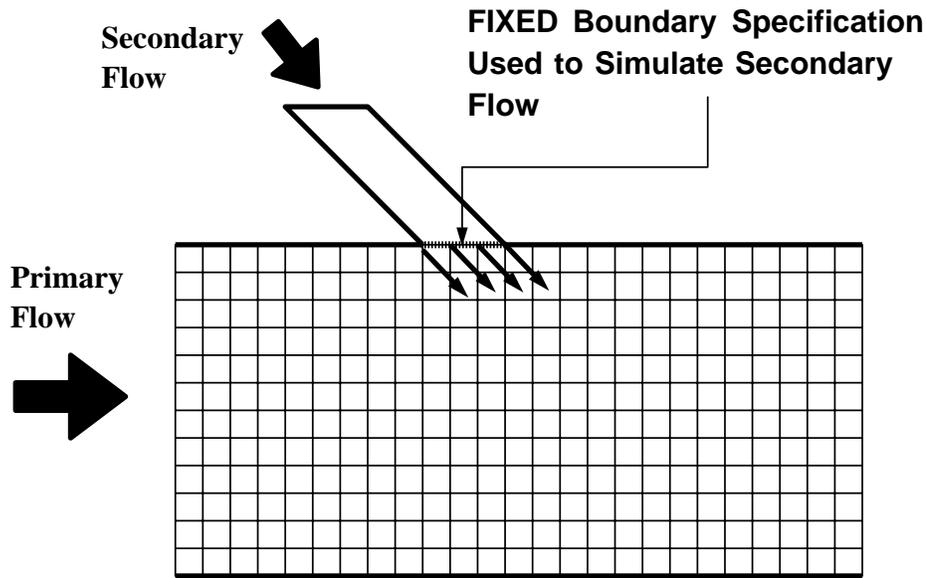
Common Errors

- Application of **EXT2DT** to a 3-D mesh system.
- Failure to specify the additional data value **PEXIT**.
- Improper specification of the alternate (mass flow) iterative scheme.

- Radial-like direction of the mesh is not the j coordinate.
- **FCART** set to 1.0 (**EXT2DT** requires cylindrical solution procedure to be selected, **FCART=0.0**).
- Failure to properly specify the **LSPEC1** variable on the boundary data file specification line.
- Value of **PEXIT** is too high (flow cannot get started).

FIXED

Fixed Flow Boundary Specification



Application

The **FIXED** specification is used as a “last resort” boundary specification which hardwires flow properties into the numerical solution. The application illustrated above indicates an application of the **FIXED** boundary specification to provide a direct implementation of the flow properties of an injection jet into a simple duct flow. The same jet could have been modeled more effectively using alternate boundary conditions, or through the addition of an additional grid to simulate the jet flow passage; however, for the purposes of demonstration, and to obtain a solution of this type quickly, the **FIXED** specification was used instead.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **FIXED** boundary condition are given below:

```

FIXED 1 1 J J P P I K 1 1 11 21 1 11 11 21 1 11
RO U V W TTOT
0.002 100.0 100.0 0.0 600.0

```

Note that a complete **FIXED** specification requires the specification of additional data beyond the standard boundary specification line.

Description

The **FIXED** statement is used to provide a fixed specification of boundary flow data in the absence of any other appropriate boundary condition. This routine was provided for those cases where other boundary conditions either cannot provide the boundary specifications desired, or in those cases where a fixed boundary specification is deemed appropriate. In most cases, the **FIXED** specification is undesirable because the boundary condition itself is perfectly reflecting, and will therefore inhibit solution convergence. In addition, the **FIXED** specification does not permit interaction between the boundary flow and the interior flow, which runs contrary to the normal fluid dynamics behavior.

A **FIXED** specification requires two additional lines in addition to the normal boundary data file descriptor. The first additional line simply contains the labels for the additional flow variable **R0**, **U**, **V**, **W**, and **TTOT**. The next line contains the actual values for the flow variable specifications. The variable **R0** defines the fluid density in slugs per cubic foot. The variables **U**, **V**, and **W** contain the fluid velocity components in feet per second for the x , y , and z coordinate directions for a Cartesian solution mesh block, and the x , r , and θ coordinate directions for a cylindrical solution mesh block, respectively. Finally, **TTOT** represents the fluid total temperature in degrees Rankine for the boundary specification. During the application of a **FIXED** specification, phantom boundary cell data are set according to the data provided in the extra lines following the boundary data specification line as shown above. As a result, the data is not necessarily applied *at* the boundary, but the influence of the data is felt just outside the boundary. This phenomenon is consistent with the behavior of a finite volume solution algorithm.

Restrictions/Limitations

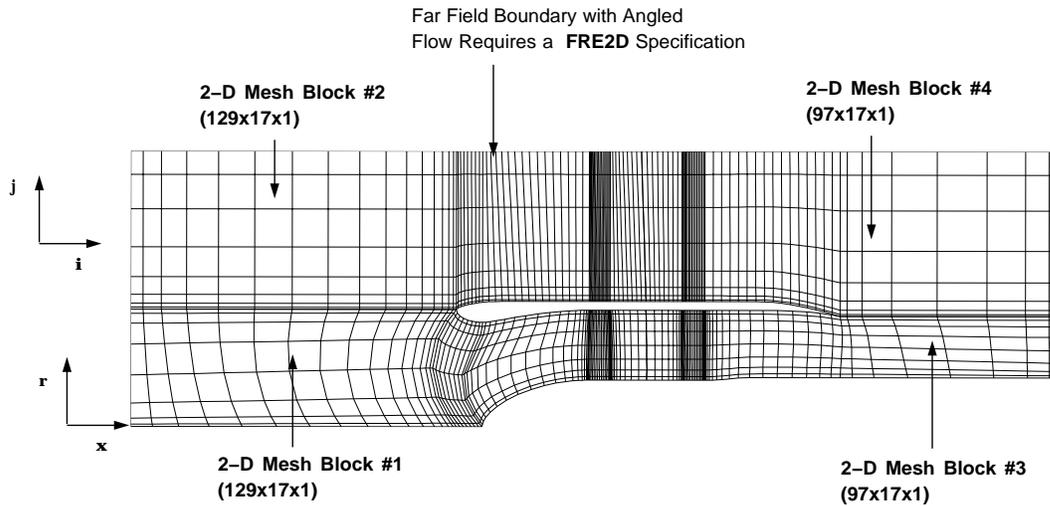
Data provided in the **FIXED** specification should represent phantom cell-centered data and must be dimensionalized as described above.

Common Errors

- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1**, **M2LIM2**, **N2LIM1**, **N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Failure to provide additional data for **FIXED** specification.
- **FIXED** boundary specifications for cylindrical solution mesh blocks must use the cylindrical velocity components.
- **FIXED** boundary specifications for Cartesian solution mesh blocks must use the Cartesian velocity components.

FRE2D

2-D Far Field Flow Boundary Condition



Application

The **FRE2D** specification is used to impose a 2-D far field boundary condition with uniform far field flow properties. The example graphic above illustrates a four-block mesh system used to predict the axisymmetric flow through a high bypass ducted fan. The two outer blocks (2 and 4) require a far-field boundary condition at the outer boundary ($j=17$). The **FRE2D** boundary specification is used to satisfy the far-field flow requirement.

Boundary Data File Format

The boundary data file specification for the mesh interface indicated in the illustrative graphic for the **FRE2D** boundary condition is given below:

```

FRE2D 2 2 J J M M I K 17 17 1 129 1 2 1 129 1 2 Block 2
PTOT TTOT EMINF ALPHA CHI/AKIN ARIN
1.0 1.0 0.75 0.0 1.0 0.0001

FRE2D 4 4 J J M M I K 17 17 1 97 1 2 1 97 1 2 Block 4
PTOT TTOT EMINF ALPHA CHI/AKIN ARIN
1.0 1.0 0.75 0.0 1.0 0.0001
    
```

Note that a complete **FRE2D** specification requires two additional lines following the **FRE2D** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **FRE2D** specification error.

Description

The **FRE2D** statement specifies that an external, free flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the 2-D block specified by **LBLCK1**. The **FRE2D** boundary condition is primarily used for external flow problems at a far field boundary to simulate the effects of the atmosphere or other large reservoir with known properties. The **FRE2D** procedure utilizes a Reimann invariant formulation to compute the local flow quantities, and permits both inflow and outflow through the bounding surface based on the nature of the local flow with respect to the known far field conditions.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **FRE2D** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, **EMINF**, and **ALPHA** be used. The next line contains the values imposed for the variables **PTOT**, **TTOT**, **EMINF**, and **ALPHA**, which represent the far field nondimensional reservoir total pressure and total temperature, along with the Mach number and Cartesian flow angle, respectively, used in the **FRE2D** characteristic solution sequence. The value of the **PTOT** and **TTOT** are the desired normalized far field total pressure and total temperature, respectively, computed as described in **INLETG**.

The variable **EMINF** represents the far field Mach number. The far field flow is *always* assumed to progress along the positive x axis, and therefore mesh systems should be generated with this in mind. The variable **ALPHA** represents the far-field Cartesian flow angle, in degrees, relative to the x axis, with positive angles resulting in far field velocity components in the y coordinate direction. Naturally, poor convergence or solution divergence can occur if **PTOT**, **TTOT**, **EMINF**, or **ALPHA** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

The additional specifications **CHI/AKIN** and **ARIN** are used with the one-equation or two-equation turbulence model and their specification is described in detail in **INLETG**.

Restrictions/Limitations

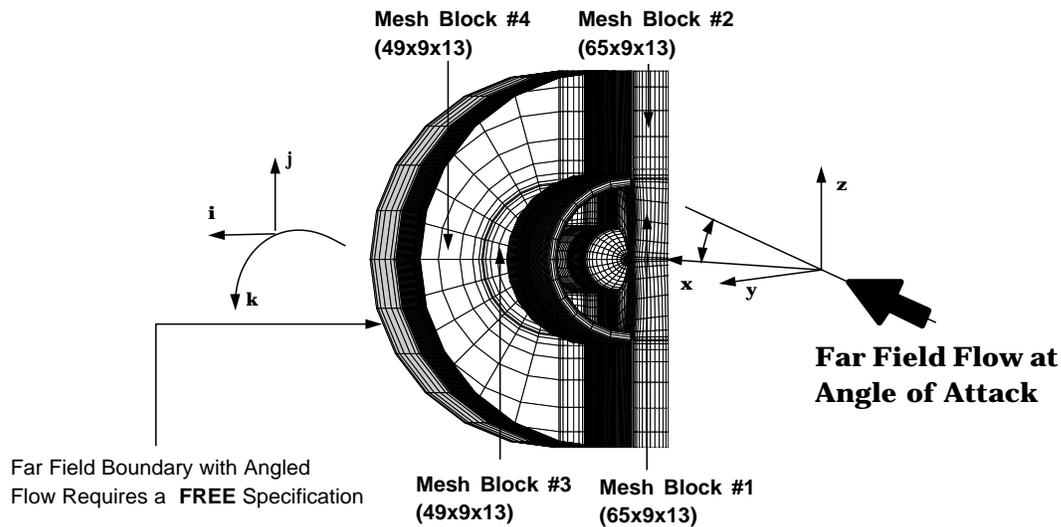
The **FRE2D** boundary specification is not restricted to 2-D mesh surfaces, although for consistency 3-D mesh surfaces may use the **FREE** boundary specification.

Common Errors

- Failure to specify the additional data values **PTOT**, **TTOT**, **EMINF**, or **ALPHA**.
- Failure to generate the mesh with $+x$ as the primary flow direction.

FREE

Far Field Flow Boundary Condition



Application

The **FREE** specification is used to impose a far field boundary condition with uniform far field flow properties. The example graphic above illustrates a four block mesh system used to predict the 3-D flow through a high bypass ducted fan. The two outer blocks 2 and 4 require a far-field boundary condition at the outer boundary ($j=9$). The **FREE** boundary specification is used to satisfy the far-field flow requirement.

Boundary Data File Format

The boundary data file specification for the mesh interfaces indicated in the illustrative graphic for the **FREE** boundary condition are given below:

```

FREE 2 2 J J M M I K 9 9 1 65 1 13 1 65 1 13
PTOT TTOT EMINF ALPHA CHI/AKIN ARIN
1.0 1.0 0.75 10.0 1.00 0.0001

FREE 4 4 J J M M I K 9 9 1 49 1 13 1 49 1 13
PTOT TTOT EMINF ALPHA CHI/AKIN ARIN
1.0 1.0 0.75 10.0 1.00 0.0001
    
```

Note that a complete **FREE** specification requires two additional lines following the **FREE** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **FREE** specification error.

Description

The **FREE** statement specifies that an external, free flow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **FREE** boundary condition is primarily used for external flow problems at a far field boundary to simulate the effects of the atmosphere or other large reservoir with known properties. The **FREE** procedure utilizes a Riemann invariant formulation to compute the local flow quantities, and permits both inflow and outflow through the bounding surface based on the nature of the local flow with respect to the known far field conditions.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **FREE** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, **EMINF**, and **ALPHA** be used. The next line contains the values imposed for the variables **PTOT**, **TTOT**, **EMINF**, and **ALPHA**, which represent the far field nondimensional reservoir total pressure and total temperature, along with the Mach number and Cartesian flow angle, respectively, used in the **FREE** characteristic solution sequence. The value of the **PTOT** and **TTOT** are the desired normalized far field total pressure and total temperature, respectively, computed as described in **INLETG**.

The variable **EMINF** represents the far field Mach number. The far field flow is *always* assumed to progress primarily along the positive x axis, and therefore mesh systems should be generated with this in mind. The variable **ALPHA** represents the far-field Cartesian flow angle, in degrees, relative to the x axis, with positive angles resulting in far field velocity components in the z coordinate direction. The flow angle velocities are *always* in the x - z plane and the velocity components in the y coordinate direction are *always* zero. If there is outflow along the **FREE** boundary, then some small y component velocities may occur as a result of extrapolation from the near field flow. Naturally, poor convergence or solution divergence can occur if **PTOT**, **TTOT**, **EMINF**, or **ALPHA** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

The additional specifications **CHI/AKIN** and **ARIN** are used with the one-equation or two-equation turbulence model and their specification is described in detail in **INLETG**.

Restrictions/Limitations

The **FREE** boundary specification is not restricted to 3-D mesh surfaces, although 2-D mesh surfaces may use the **FRE2D** boundary specification for consistency. The far field flow angle must be specified relative to the x axis, and produces additional velocity components in the z coordinate direction only. Imposed far-field velocity components in the y coordinate direction will always be zero for 3-D meshes.

Common Errors

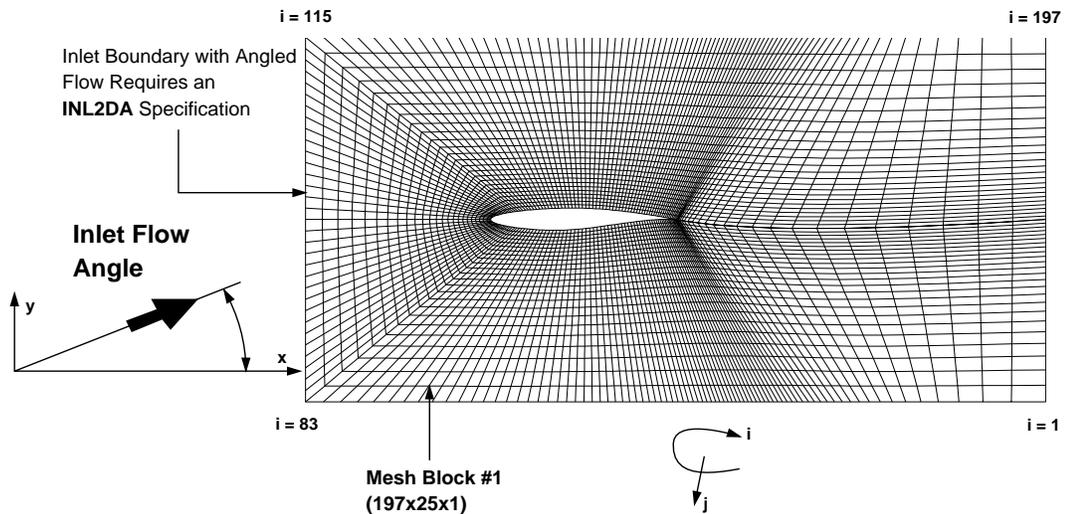
- Application of **FREE** to a boundary for which far field y coordinate direction velocity components are required.

BOUNDATA KEYWORDS

- Failure to specify the additional data values PTOT, TTOT, EMINF, or ALPHA.
- Failure to generate the mesh with $+x$ as the downstream flow direction.

INL2DA

Cartesian Flow Angle Inflow Boundary Condition Procedure



Application

The **INL2DA** specification is used to impose a Cartesian flow angle inflow boundary condition with uniform flow properties at a local mesh surface. The illustrative graphic above depicts a single-block mesh system for an isolated airfoil geometry. The **INL2DA** specifier is utilized at the inlet of mesh block 1 to set the angled inflow necessary to simulate angle of attack.

Boundary Data File Format

The boundary data file specification for the mesh boundaries indicated in the illustrative graphic for the **INL2DA** boundary condition are given below:

```
INL2DA 1 1 J J M M I K 25 25 83 115 1 2 83 115 1 2
PTOT TTOT ALPHA CHI/AKIN ARIN
1.0 1.0 20.0 1.00 0.0001
```

Note that a complete **INL2DA** specification requires two additional lines following the **INL2DA** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INL2DA** specification error.

Description

The **INL2DA** keyword specifies that a uniform property flow angle inflow boundary

condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBL0CK1**. **INL2DA** is valid for Cartesian solution meshes. The **INL2DA** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified upstream reservoir total pressure and total temperature, and a single Cartesian flow angle as shown in the illustrative graphic, above. Included in the **INL2DA** procedure is a special correction scheme which forces the flow to pass into the flow domain.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INL2DA** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, and **ALPHA** be used. The next line contains the values imposed for the variables **PTOT**, **TTOT**, and **ALPHA** which represent the upstream reservoir total pressure, total temperature, and Cartesian flow angle, respectively, used in the **INL2DA** characteristic solution sequence. The value of the **PTOT** and **TTOT** are the desired normalized far field total pressure and total temperature, respectively, computed as described in **INLETG**.

The variable **ALPHA** represents the flow angle in degrees referenced to the x axis. For 2-D applications, positive flow angles generate components of the flow in the positive y direction, and inlet velocity component in the z direction are set to zero. Values of **ALPHA** must lie between ± 90 degrees. Naturally, poor convergence or solution divergence can occur if **PTOT** or **TTOT** suggest boundary values which are significantly different from the remainder of the flowfield, or if **ALPHA** is very large. In cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

The additional specifications **CHI/AKIN** and **ARIN** are used with the one-equation or two-equation turbulence model and their specification is described in detail in **INLETG**.

Restrictions/Limitations

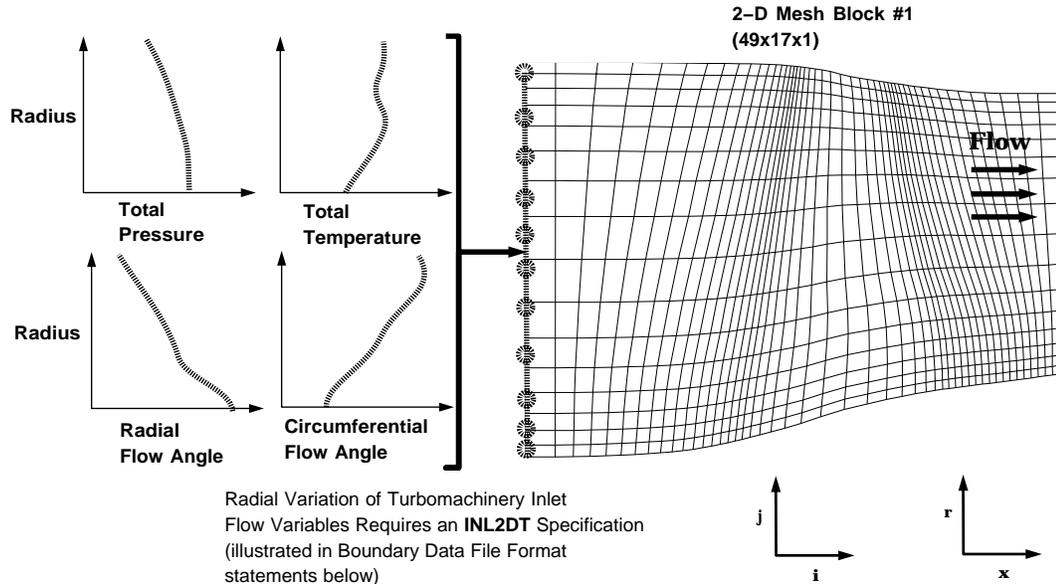
The **INL2DA** boundary specification is applied to 2-D mesh surface. The angle specified by **ALPHA** is assumed to be in the positive y coordinate. An example of this type of application is for a planar cascade flow where x is the primary flow direction, and the inflow is at an angle of 20 degrees relative to the x axis, thus resulting in a y component of velocity (2-D only). The angle of the velocity components specified by the **INL2DA** procedure must always be referenced to the x coordinate axis, and it is left to the user to generate a mesh which is consistent with this feature.

Common Errors

- Application of **INL2DA** to a cylindrical mesh system.
- Application of **INL2DA** to a 2-D mesh boundary for which non-zero z component velocities are required.
- Failure to specify the additional data values **PTOT**, **TTOT**, or **ALPHA**.

INL2DT

2-D Turbomachinery Inflow Boundary Condition



Application

The **INL2DT** specification is used to impose an inflow boundary condition with radially varying flow properties for 2-D axisymmetric mesh systems. The example graphic illustrated above depicts an **EXT2DT** specification for a 2-D (axisymmetric) flow solution for a turbomachinery blade row.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **INL2DT** boundary condition is given below:

```

INL2DT 1 1 I I P P J K 1 1 1 17 1 2 1 17 1 2
NDATA
7
RAD PTOT TTOT BETAR BETAT CHI
0.20 1.01 0.98 5.0 5.1 1.0
0.25 1.01 0.99 4.0 5.7 1.0
0.30 1.00 1.00 3.0 6.3 1.0
0.35 0.99 1.01 2.5 6.8 1.0
0.40 0.97 1.00 2.0 7.4 1.0
0.45 0.96 1.01 1.0 8.0 2.0
0.50 0.95 1.01 0.0 7.7 5.0
    
```

A complete **INL2DT** specification requires at least six additional lines (defining at least 3 points on the inlet data distribution) following the **INL2DT** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INL2DT** specification error.

Description

The **INL2DT** statement specifies that a turbomachinery-based radially varying inflow boundary condition is to be applied to the 2-D mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INL2DT** boundary condition was specifically designed as an inflow boundary procedure for axial and mixed flow axisymmetric turbomachinery geometries. The **INL2DT** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified radial variation in flow properties (upstream reservoir total pressure, total temperature, radial flow angle, and circumferential flow angle). Included in the **INL2DT** procedure is a special correction scheme which forces the flow to pass into the flow domain.

This boundary condition requires the specification of additional data. The details of creating the inlet radial distribution table is outlined in **INLETT**.

Restrictions/Limitations

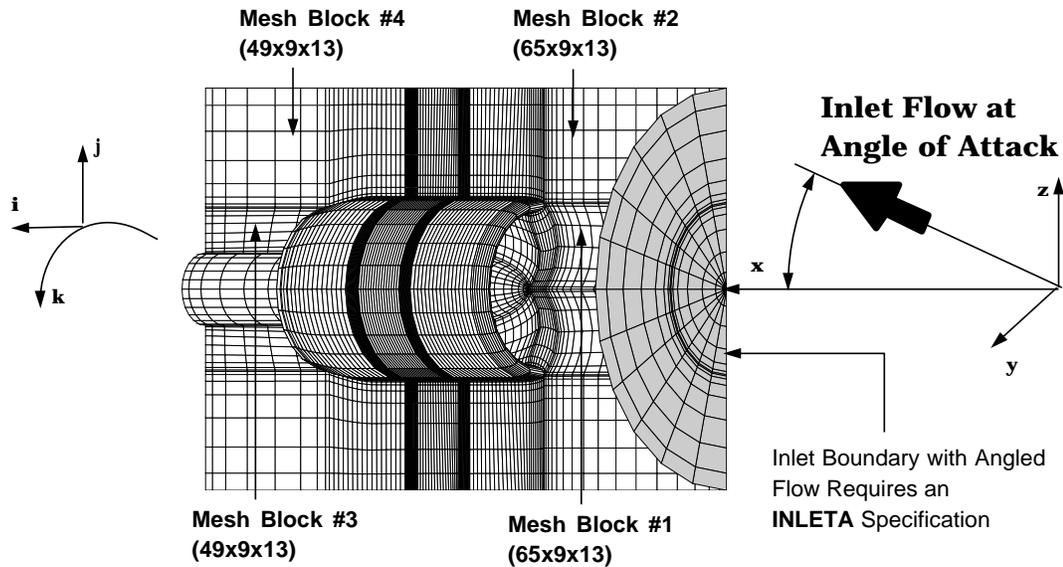
The **INL2DT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate. The **INL2DT** boundary specification is restricted to 2-D mesh surfaces.

Common Errors

- Application of **INL2DT** to a 3-D mesh system.
- Failure to specify the additional data values **NDATA**, **PTOT**, **TTOT**, **BETAR**, or **BETAT**.
- Radial-like direction of the mesh is not the j coordinate.
- **NDATA** less than 3, resulting in job termination.
- **BETAR** and/or **BETAT** orientation incorrectly interpreted.
- **RAD**, **PTOT**, and/or **TTOT** improperly normalized.
- Mesh/geometry not defined with the x axis as the centerline.

INLETA

Cartesian Flow Angle Inflow Boundary Condition Procedure



Application

The **INLETA** specification is used to impose a Cartesian flow angle inflow boundary condition with uniform flow properties at a local mesh surface. The illustrative graphic above depicts a four block mesh system for a turbfan engine geometry. The **INLETA** specifier is utilized at the inlet of mesh blocks 1 and 2 to set the angled inflow necessary to simulate angle of attack.

Boundary Data File Format

The boundary data file specification for the mesh boundaries indicated in the illustrative graphic for the **INLETA** boundary condition are given below:

```

INLETA 1 1 I I P P J K 1 1 1 9 1 13 1 9 1 13
PTOT TTOT ALPHA CHI/AKIN ARIN
1.0 1.0 20.0 1.00 0.0001

INLETA 2 2 I I P P J K 1 1 1 9 1 13 1 9 1 13
PTOT TTOT ALPHA CHI/AKIN ARIN
1.0 1.0 20.0 1.00 0.0001
    
```

Note that a complete **INLETA** specification requires two additional lines following the

INLETA boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETA** specification error.

Description

The **INLETA** keyword specifies that a uniform property flow angle inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. **INLETA** is valid for both cylindrical and Cartesian solution meshes (see the description of the input variable **FCART**). The **INLETA** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified upstream reservoir total pressure and total temperature, and a single Cartesian flow angle as shown in the illustrative graphic, above. Included in the **INLETA** procedure is a special correction scheme which forces the flow to pass into the flow domain.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INLETA** specification is assumed to be a label and may contain any information; however, for consistency it is recommended that the labels **PTOT**, **TTOT**, and **ALPHA** be used. The next line contains the values imposed for the variables **PTOT**, **TTOT** and **ALPHA** which represent the upstream reservoir total pressure, total temperature, and Cartesian flow angle, respectively, used in the **INLETA** characteristic solution sequence. The value of the **PTOT** and **TTOT** are the desired normalized far field total pressure and total temperature, respectively, computed as described in **INLETG**.

The variable **ALPHA** represents the flow angle in degrees referenced to the x axis. For 3-D applications, positive flow angles generate components of the flow in the positive z direction, and inlet velocity component in the y direction are set to zero. For 2-D applications, positive flow angles generate components of the flow in the positive y direction, and inlet velocity component in the z direction are set to zero. Values of **ALPHA** must lie between +/- 90 degrees. Naturally, poor convergence or solution divergence can occur if **PTOT** or **TTOT** suggest boundary values which are significantly different from the remainder of the flowfield, or if **ALPHA** is very large. In cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

The additional specifications **CHI/AKIN** and **ARIN** are used with the one-equation or two-equation turbulence model and their specification is described in detail in **INLETG**.

Restrictions/Limitations

The **INLETA** boundary specification may be applied to either 3-D or 2-D mesh surfaces. For 2-D applications, the angle specified by **ALPHA** is assumed to be in the positive y coordinate. The angle of the velocity components specified by the **INLETA** procedure must always be referenced to the x coordinate axis, and it is left to the user to generate a mesh which is consistent with this feature.

Common Errors

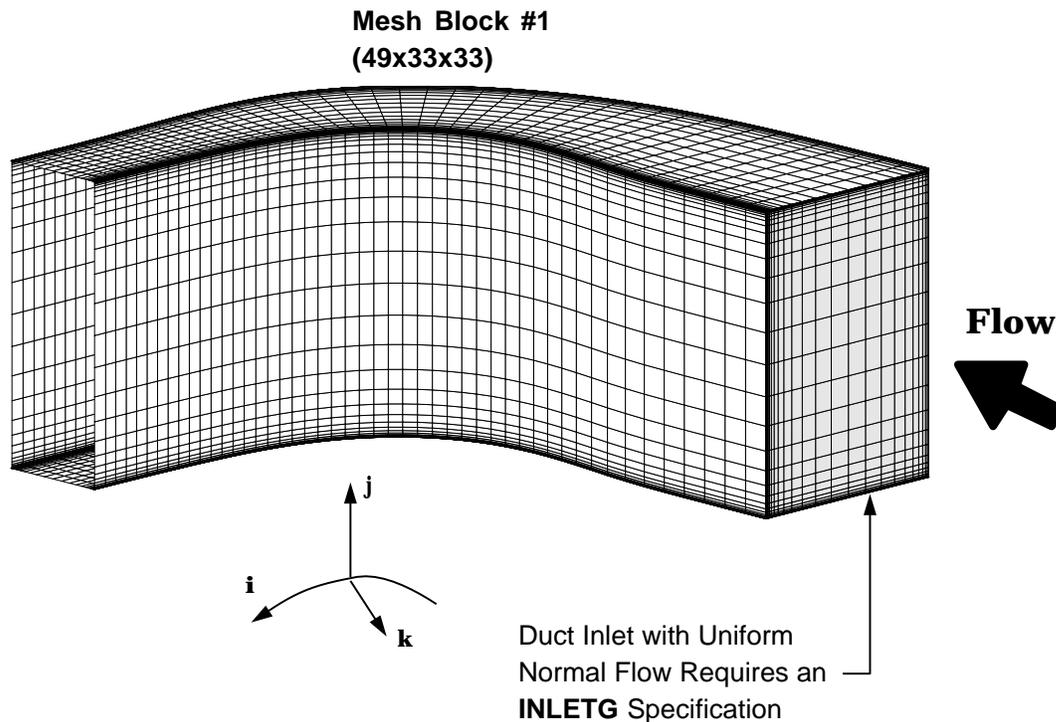
- Application of **INLETA** to a 3-D mesh boundary for which non-zero y component

velocities are required.

- Application of **INLETA** to a 2-D mesh boundary for which non-zero z component velocities are required.
- Failure to specify the additional data values **PTOT**, **TTOT**, or **ALPHA**.

INLETG

Generic Inflow Boundary Condition



Application

The **INLETG** specification is used to impose a generic inflow boundary condition with uniform flow properties where the default inflow velocity is normal to the local mesh surface. Angle specification can be added to modify the injection angle of the inflow.

Boundary Data File Format

The boundary data file specification for the mesh interface indicated in the illustrative graphic for the **INLETG** boundary condition is given below:

```
INLETG 1 1 I I P P J K 1 1 1 33 1 33 1 33 1 33
PTOT TTOT CHI/AKIN ARIN ANGLE1 ANGLE2 EMDOT PRELAX
1.0 1.0 1.00 0.0001 0.0 -20.0 5.34 0.1
```

Note that a complete **INLETG** specification requires two additional lines following the **INLETG** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETG** specification error. Of the additional variables specified, only PTOT and TTOT are required.

Description

The **INLETG** statement specifies that a generic, uniform normal inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INLETG** boundary condition should be applied for those cases where any other “specialized” inflow boundary condition (i.e, **INLETR**, **INLETT**) does not apply. The **INLETG** boundary condition is also likely to be somewhat more efficient computationally than the other inflow boundary condition procedures, at the expense of some physical simplification. **INLETG** may be utilized on either cylindrical or Cartesian solution meshes (see the description of the input variable **FCART**). The **INLETG** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified upstream reservoir total pressure and total temperature. The velocity components at an **INLETG** boundary are always computed to be normal (no transverse velocity components) to the local cell face at which the procedure is applied. Included in the **INLETG** procedure is a special correction scheme which forces the flow to pass into the flow domain.

This boundary condition requires the specification of additional data. The first additional line following the **INLETG** specification is assumed to be a label. The next line contains the values imposed for the variables **PTOT**, **TTOT**, **CHI/AKIN**, and **ARIN**, which represent the upstream reservoir total pressure, total temperature, and turbulence model quantities, respectively. The value of the **PTOT** variable is the desired normalized upstream total pressure computed as:

$$PTOT = \frac{P_{total,desired}}{P_{ref}}$$

and the value of the **TTOT** variable is the desired normalized upstream total temperature computed as:

$$TTOT = \frac{T_{total,desired}}{T_{ref}}$$

The variables P_{ref} and T_{ref} are specified by the input variables **PREF** and **TREF**. Values of **PTOT** and **TTOT** < 0.0 are not permitted. Naturally, poor convergence or solution divergence can occur if **PTOT** or **TTOT** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

The variable **CHI** is only used when the one-equation turbulence model is enabled (see **F1EQ**). When enabled, the variable **CHI** specifies the nondimensional inlet turbulence level (χ) used in the Spalart-Allmaras model. Details of this model are included in Appendix A. Normally this value should be set to 1.0 for general fully turbulent flow; Larger values of **CHI** can be used to simulate higher turbulence levels.

The variables **AKIN** and **ARIN** are only used when the one-equation ($k - \mathcal{R}$) turbulence model is enabled (see **F2EQ**). When enabled, the variable **AKIN** represents the value of the nondimensional freestream turbulence kinetic energy defined by k/V_{ref}^2 where k is the freestream turbulent kinetic energy and V_{ref} is the reference velocity defined by $\sqrt{R_{ref}T_{ref}}$. Here R_{ref} is the gas constant. The variable **ARIN** represent the so-called freestream turbulence Reynolds number and is calculated as $\mathcal{R}/V_{ref}L_{ref}$, where L_{ref} is

the reference length defined by the input variable **DIAM**. Additional details covering the two-equation model are included in Appendix A.

By default, the inflow is constrained to be normal to the boundary; however, this injection angle can be specified using the **ANGLE1** and **ANGLE2** variables. These angle specifications are relative to the *local* mesh index direction and not to the global coordinate system. **ANGLE1** and **ANGLE2** prescribe the angles with respect to the remaining two mesh indices in “natural” order, respectively (e.g., if the **INLETG** surface is a *j* constant grid plane, then **ANGLE1** and **ANGLE2** would be in the *i* and *k* directions, respectively.). The default values for both the angle specifications is 0.0 degrees (normal to the surface). This feature is extremely useful in modeling injected cooling flow along an airfoil surface.

A mass flow specification similar to that used in **EXITG** is also available with **INLETG**. Whereas the **EXITG** boundary condition iteratively altered the exit *static* pressure to regulate mass flow, the **INLETG** will alter the inlet *total* pressure each iteration until the prescribed mass flow is reached. **EMDOT** represents the desired mass flow rate through the bounding surface in pounds mass, and **PRELAX** is a relaxation factor to stabilize the iterative process (values may range from 0.0 to 1.0, though poor convergence is likely for values larger than 0.1). In order to use the mass flow specification without employing the angled flow injection, the user needs to remember to assign place holders for **ANGLE1** and **ANGLE2** equal to 0.0 prior to specifying the mass flow value and relaxation parameter.

Restrictions/Limitations

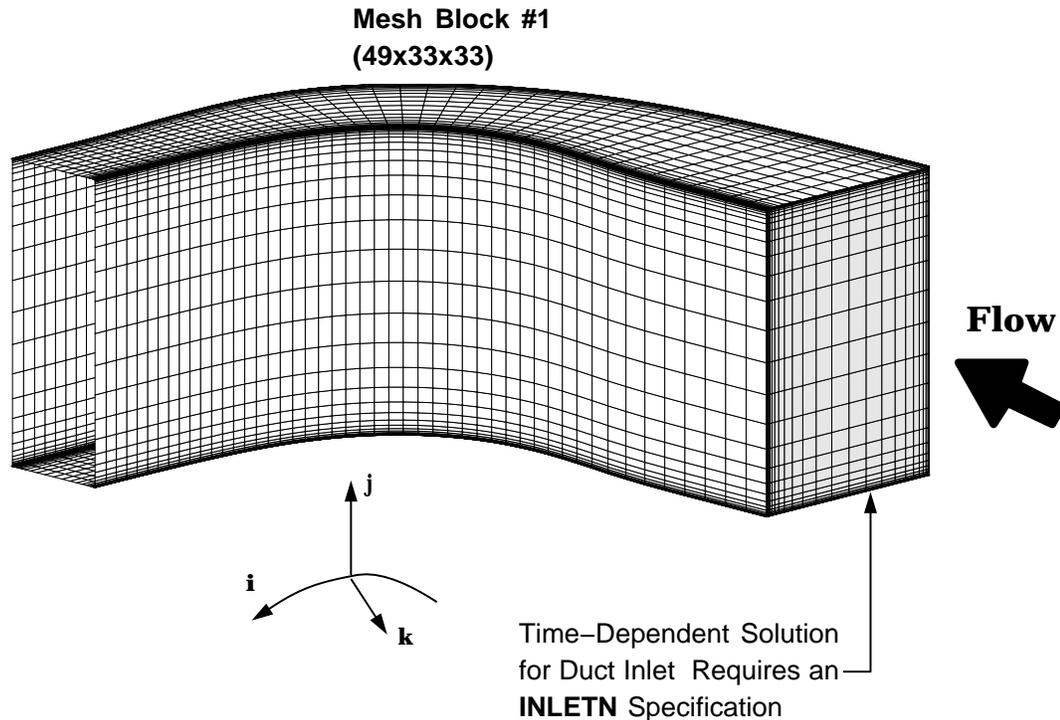
The **INLETG** boundary specification is applicable 2-D and 3-D mesh surfaces.

Common Errors

- Application of **INLETG** to a boundary for which transverse inflow velocity components are required.
- Failure to specify the additional data values **PTOT** or **TTOT**.

INLETN

Non-Reflecting Unsteady Inflow Boundary Condition



Application

The **INLETN** specification is used to impose a non-reflecting inflow boundary condition for time-dependent flow calculations. This boundary condition is utilized for time-dependent flows where spurious numerical reflections from other boundary algorithms are undesirable.

Boundary Data File Format

The boundary data file specification for the mesh interface indicated in the illustrative graphic for the **INLETN** boundary condition is given below:

```
INLETN 1 1 I I P P J K 1 1 1 33 1 33 1 33 1 33
```

Description

The **INLETN** statement specifies that a generic, non-reflecting inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by

LBLOCK1. The **INLETN** boundary condition should only be applied for time-dependent flow calculations where spurious numerical reflections such as those normally expected from other boundary specifications are undesirable. **INLETN** may be utilized on either cylindrical or Cartesian solution meshes. The **INLETN** procedure utilizes a characteristic-based formulation to compute inflow velocities based on local flow conditions only.

Restrictions/Limitations

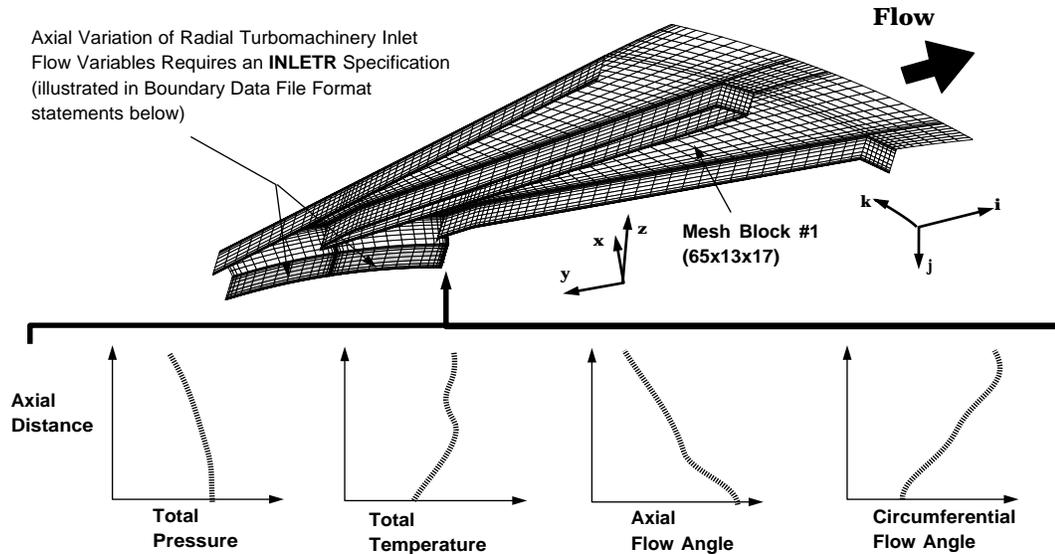
The **INLETN** boundary specification is not restricted to 3-D mesh surfaces but should only be applied for time-dependent flow calculations where a non-reflecting inflow boundary is desired.

Common Errors

- Application of **INLETN** for a steady-state solution.

INLETR

Radial Flow Turbomachinery Inflow Boundary Condition



Application

The **INLETR** specification is used to impose an inflow boundary condition with axially varying flow properties for radial flow turbomachinery. The example graphic above illustrates adjacent passages of a mesh system designed to predict the flow through a radial diffuser. The inlet boundary is a radial surface of revolution with properties which vary in the *axial* direction, and therefore **INLETR** is used to supply the desired flow characteristics at this boundary.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **INLETR** boundary condition is given below:

```
INLETR 1 1 I I P P J K 1 1 1 13 1 17 1 13 1 17
NDATA
4
AXIAL PTOT TTOT BETAX BETAT
0.1 0.99 0.99 5.0 -73.3
0.2 0.98 1.01 4.0 -75.8
0.3 0.97 1.00 3.0 -77.2
0.4 0.96 1.01 2.0 -79.0
```

A complete **INLETR** specification requires at least six additional lines (defining at least 3 points on the inlet data distribution) following the **INLETR** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETR** specification error.

Description

The **INLETR** statement specifies that a radial flow turbomachinery inlet flow boundary condition with axially varying flow properties is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INLETR** boundary condition was specifically designed as an inflow boundary procedure for pure radial flow turbomachinery geometries. The **INLETR** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified axial variation in flow properties (upstream reservoir total pressure, total temperature, axial flow angle, and circumferential flow angle). Included in the **INLETR** procedure is a special correction scheme which forces the flow to pass into the flow domain.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INLETR** specification is assumed to be a label. The line following the **NDATA** label contains the number of axial data points which will be used to specify the desired axial variation of properties at the inflow boundary. At least 3 axial data locations must be specified to use the **INLETR** boundary condition. The third line following the **INLETR** specifier is again a label which outlines the variables **AXIAL**, **PTOT**, **TTOT**, **BETAX** and **BETAT**. The remaining **NDATA** lines contain the numeric information which defines the axial variation of the flow properties specified by these variables.

The variable **AXIAL** is the axial coordinate (x) at which the data is specified. This value should be nondimensionalized in the same manner as the mesh is nondimensionalized. This implies that the **AXIAL** variable, when multiplied by the input variable **DIAM** will result in the true geometric measurement in feet. Due to the interpolation procedures which will ultimately be performed on the **NDATA** lines of radial inflow data, it is essential that the axial locations be specified in a monotonic (constantly increasing) fashion. The variables **PTOT** and **TTOT** represent the local upstream reservoir total pressure and total temperature, respectively, used in the **INLETR** characteristic solution sequence. The value of the **PTOT** and **TTOT** are the desired normalized far field total pressure and total temperature, respectively, computed as described in **INLETG**. The variables **BETAX** and **BETAT** represent the local axial and circumferential flow angles expressed in degrees according to the coordinate orientation defined in Figure 3.12.

Naturally, poor convergence or solution divergence can occur if any of the values of **PTOT**, **TTOT**, **BETAX**, or **BETAT** suggest boundary values which are significantly different from the remainder of the flowfield, or if the axial variation of these values is excessively large. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

The **INLETR** boundary condition assumes that the mesh is oriented in such a fashion that

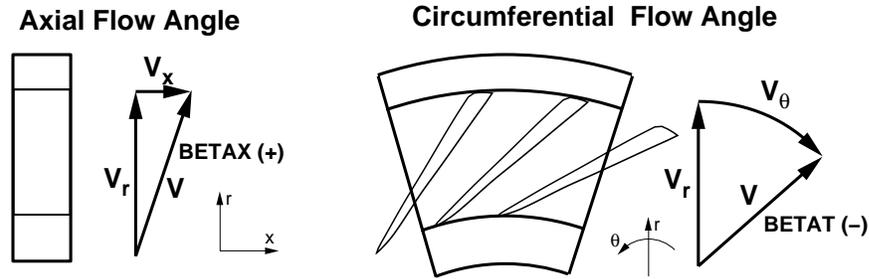


Figure 3.12: ADPAC INLETR boundary specification flow angle reference

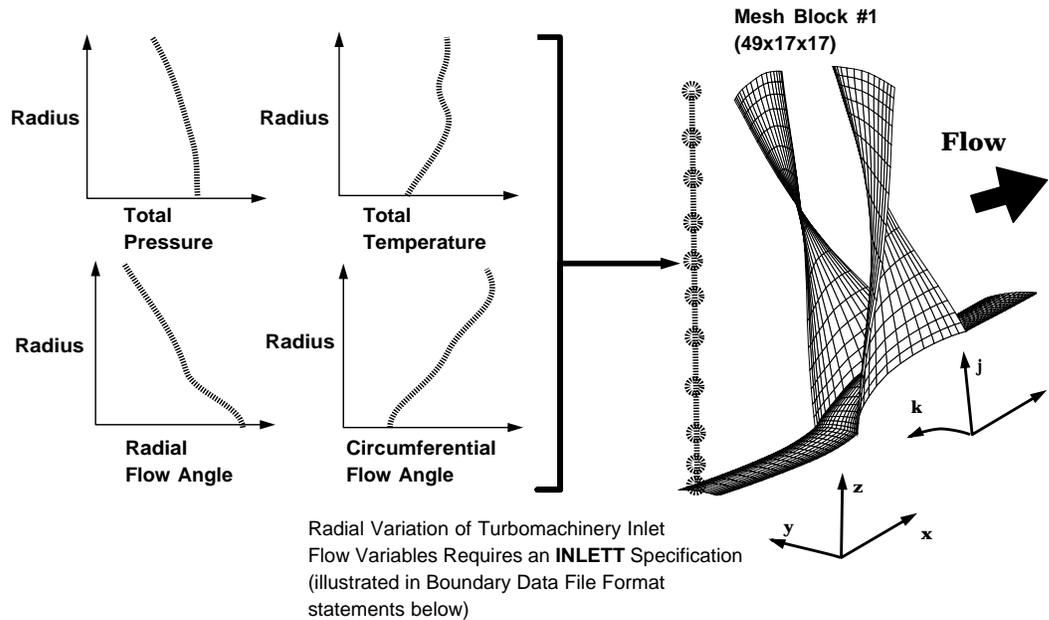
the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For radial flow turbomachinery, this implies that the axis of rotation (or the centerline) coincides with the x axis. It is also required that the axial-like direction of the mesh be defined by the j coordinate.

Common Errors

- Failure to specify the additional data values NDATA, AXIAL, PTOT, TTOT, BETAX, or BETAT.
- Axial-like direction of the mesh is not the j coordinate.
- NDATA less than 3, resulting in job termination.
- BETAX and/or BETAT orientation incorrectly interpreted.
- AXIAL, PTOT, and/or TTOT improperly normalized.
- Mesh/geometry not defined with the x axis as the centerline.

INLETT

Turbomachinery Inflow Boundary Condition



Application

The **INLETT** specification is used to impose an inflow boundary condition with radially varying flow properties. The illustrative graphic above depicts an application of the **INLETT** inflow boundary condition for an H-type mesh for a turbomachinery fan rotor blade passage. The **INLETT** specification provides the radial variation of flow properties at the inflow boundary resulting from experimental conditions, upstream blade rows, or other known inlet property variation.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **INLETT** boundary condition is given below:

```

INLETT 1 1 I I P P J K 1 1 1 17 1 17 1 17 1 17
NDATA
7
RAD PTOT TTOT BETAR BETAT CHI
0.20 1.01 0.98 5.0 5.1 1.0
0.25 1.01 0.99 4.0 5.7 1.0
0.30 1.00 1.00 3.0 6.3 1.0
0.35 0.99 1.01 2.5 6.8 1.0
    
```

0.40	0.97	1.00	2.0	7.4	1.0
0.45	0.96	1.01	1.0	8.0	2.0
0.50	0.95	1.01	0.0	7.7	5.0

A complete **INLETT** specification requires six or more additional lines (defining at least 3 points on the inlet data distribution) following the **INLETT** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETT** specification error.

Description

The **INLETT** statement specifies that a turbomachinery-based radially varying inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INLETT** boundary condition was specifically designed as an inflow boundary procedure for axial and mixed flow turbomachinery geometries. The **INLETT** boundary procedure is only valid on mesh systems employing the cylindrical solution algorithm (see input variable **FCART**). The **INLETT** procedure utilizes a Reimann invariant formulation to compute inflow velocities based on a specified radial variation in flow properties (upstream reservoir total pressure, total temperature, radial flow angle, and circumferential flow angle). Included in the **INLETT** procedure is a special correction scheme which forces the flow to pass into the flow domain.

This boundary condition requires the specification of additional data, as shown in the boundary data format descriptor above. The first additional line following the **INLETT** specification is assumed to be a label. The line following the **NDATA** label contains the number of radial data points which will be used to specify the desired radial variation of properties at the inflow boundary. At least 3 radial data locations must be specified to use the **INLETT** boundary condition. The third line following the **INLETT** specifier is again a label which outlines the variables **RAD**, **PTOT**, **TTOT**, **BETAR**, **BETAT**, and optionally **CHI**. The remaining **NDATA** lines contain the numeric information which defines the radial variation of the flow properties specified by these variables.

The variable **RAD** is the radius at which the data is specified. This value should be nondimensionalized in the same manner as the mesh is nondimensionalized. This implies that the **RAD** variable, when multiplied by the input variable **DIAM** will result in the true geometric measurement in feet. Due to the interpolation procedures which will ultimately be performed on the **NDATA** lines of radial inflow data, it is essential that the radial variations be specified from the inner to the outer radius in a monotonic (constantly increasing) fashion. The variables **PTOT** and **TTOT** represent the local upstream reservoir total pressure and total temperature, respectively, used in the **INLETT** characteristic solution sequence. The value of the **PTOT** and **TTOT** are the desired normalized far field total pressure and total temperature, respectively, computed as described in **INLETTG**. The variables **BETAR** and **BETAT** represent the local radial and circumferential flow angles expressed in degrees according to the coordinate orientation defined in Figure 3.13.

Naturally, poor convergence or solution divergence can occur if any of the values of **PTOT**, **TTOT**, **BETAR**, or **BETAT** suggest boundary values which are significantly different from the remainder of the flowfield, or if the radial variation of these values is excessively large. In cases where this occurs, it is recommended that the solution be started with

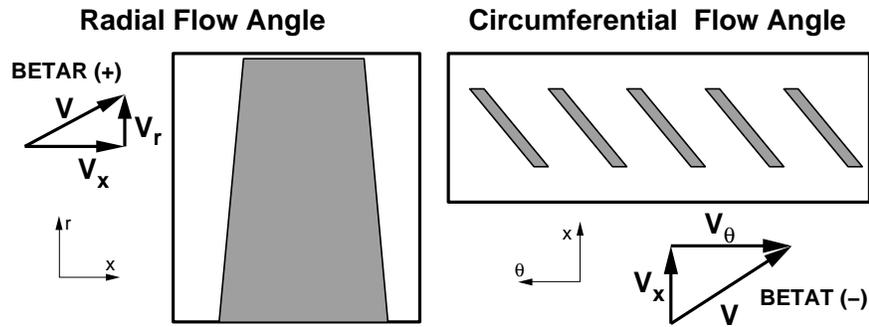


Figure 3.13: ADPAC INLETT boundary specification flow angle reference

more conservative boundary values, and then restarted using the final boundary values.

The additional specification CHI is used with the one-equation turbulence model and allows for a radial profile of turbulence to be specified. The specifics of how CHI is used is described in detail in **INLETTG**.

Restrictions/Limitations

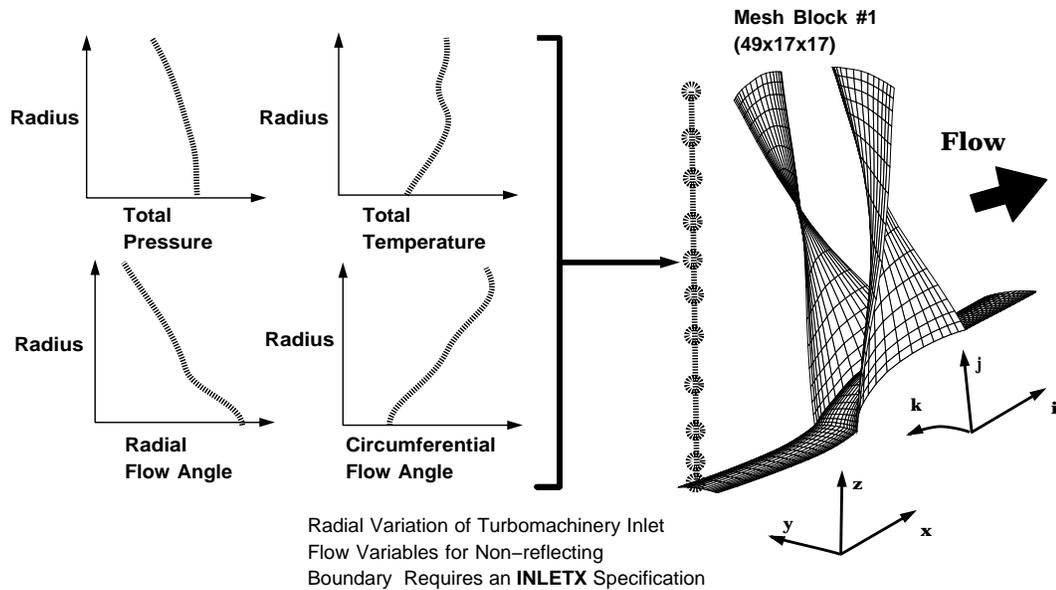
The **INLETT** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate. The **INLETT** boundary specification is restricted to 3-D mesh surfaces.

Common Errors

- Application of **INLETT** to a 2-D mesh system.
- Failure to specify the additional data values NDATA, PTOT, TTOT, BETAR, or BETAT.
- Radial-like direction of the mesh is not the j coordinate.
- NDATA less than 3, resulting in job termination.
- BETAR and/or BETAT orientation incorrectly interpreted.
- RAD, PTOT, and/or TTOT improperly normalized.
- Mesh/geometry not defined with the x axis as the centerline.

INLETX

Non-Reflecting Steady State Turbomachinery Inflow Boundary Condition



Application

The **INLETX** specification is used to impose a non-reflecting turbomachinery inflow boundary condition with radially varying flow properties. The illustrative graphic above depicts an application of the **INLETX** inflow boundary condition for an H-type mesh for a turbomachinery fan rotor blade passage. The **INLETX** specification provides the radial variation of flow properties at the inflow boundary resulting from experimental conditions, upstream blade rows, or other known inlet property variation.

Boundary Data File Format

The boundary data file specification for the mesh surface indicated in the illustrative graphic for the **INLETX** boundary condition is given below:

```
INLETX 1 1 I I P P J K 1 1 1 17 1 17 1 17 1 17
NDATA
7
RAD PTOT TTOT BETAR BETAT CHI
0.20 1.01 0.98 5.0 5.1 1.0
0.25 1.01 0.99 4.0 5.7 1.0
0.30 1.00 1.00 3.0 6.3 1.0
```

0.35	0.99	1.01	2.5	6.8	1.0
0.40	0.97	1.00	2.0	7.4	1.0
0.45	0.96	1.01	1.0	8.0	1.0
0.50	0.95	1.01	0.0	7.7	1.0

A complete **INLETX** specification requires six or more additional lines (defining at least 3 points on the inlet data distribution) following the **INLETX** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **INLETX** specification error.

Description

The **INLETX** statement specifies that a non-reflecting turbomachinery-based radially varying inflow boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **INLETX** boundary condition was specifically designed as a non-reflecting inflow boundary procedure for steady-state analysis of axial and mixed flow turbomachinery geometries. As such, the **INLETX** boundary procedure is only valid on mesh systems employing the cylindrical solution algorithm (see input variable **FCART**). The **INLETX** procedure utilizes a characteristic-based flow decomposition to compute inflow velocities based on a specified radial variation in flow properties (upstream reservoir total pressure, total temperature, radial flow angle, and circumferential flow angle). Due to the non-reflective nature of the boundary procedure, the circumferential average of the numerical solution at each radial station matches the specifications imposed by the **INLETX** specification (rather than a point-by-point matching). This feature permits circumferential variation of flow properties across the boundaries, where other boundary procedures do not.

This boundary condition requires the specification of additional data. The details of creating the inlet radial distribution table is outlined in **INLETT**, except that the variable **CHI** is not currently supported under **INLETX**.

Restrictions/Limitations

The **INLETX** boundary condition assumes that the mesh is oriented in such a fashion that the radial coordinate is defined as $r = \sqrt{y^2 + z^2}$. For axial flow turbomachinery, this implies that the axis of rotation (centerline) coincides with the x axis. It is also required that the radial-like direction of the mesh be defined by the j coordinate. This implies that **INLETX** can only be applied to either constant i or constant k index mesh surfaces. **INLETX** is only valid for steady-state solutions (the **INLETN** boundary procedure is available for time-dependent non-reflecting inflow boundaries).

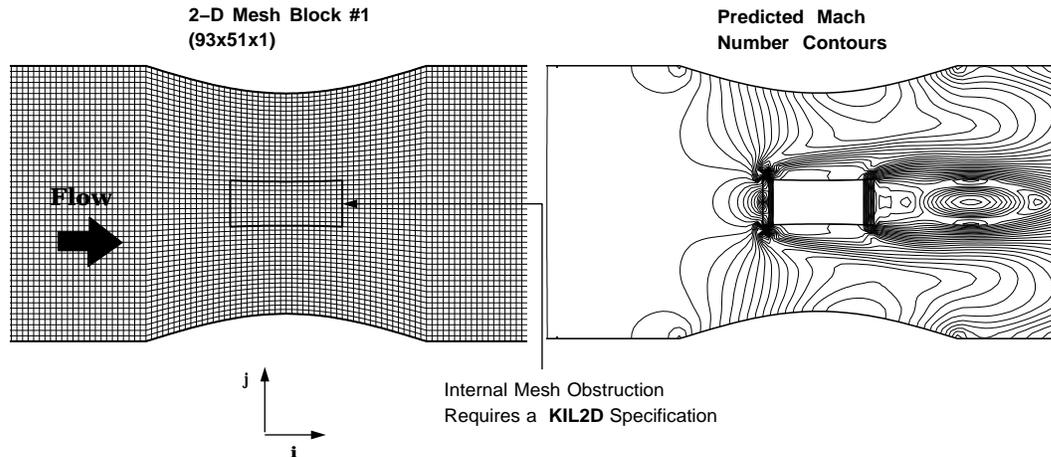
Common Errors

- Application of **INLETX** to a Cartesian mesh solution.
- Failure to specify the additional data values **NDATA**, **PTOT**, **TTOT**, **BETAR**, or **BETAT**.
- Radial-like direction of the mesh is not the j coordinate.

- Application of **INLETX** to a constant j mesh surface.
- **NDATA** less than 3, resulting in job termination.
- **BETAR** and/or **BETAT** orientation incorrectly interpreted.
- **RAD**, **PTOT**, and/or **TTOT** improperly normalized.
- Mesh/geometry not defined with the x axis as the centerline.
- Application of **INLETX** for a time-dependent solution.
- Specification of **CHI** field has no effect currently.

KIL2D

2-D Solution Kill Routine



Application

The **KIL2D** keyword is a tool to effectively neutralize or “kill” the time-marching solution over a segment of the computational domain for a two-dimensional mesh. The example graphic above illustrates a single block 2-D mesh system used to predict the flow through a converging/diverging nozzle system with a square-edged obstruction. Rather than construct a multiple block mesh system to treat this case (whereby the obstruction is essentially gridded as block boundaries), the **KIL2D** specification is used to neutralize the advancing solution within the obstruction, and boundary conditions are applied along the surface of the obstruction to predict this flow.

Boundary Data File Format

The boundary data file specification for the mesh interface indicated in the illustrative graphic for the **KIL2D** boundary condition is given below:

```
KIL2D  1  1  I  I  M  M  L  L  40  60  21  31  1  2  21  31  1  2
      LSTART  LEND
           40    60
```

Note that a complete **KIL2D** specification requires two additional lines following the **KIL2D** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **KIL2D** specification error.

Description

In cases where a portion of a 2-D mesh does not represent a valid flow region, the **KIL2D** specification can be used, in conjunction with boundary conditions specified about the region to be “killed”, to effectively remove a portion of a given mesh block from the computational domain. The figure depicts a single block mesh for the flow through a simple nozzle. Suppose that for whatever reason, the user wished to remove an internal rectangular portion of the mesh (as if there were an obstruction placed in the flowpath). This could be accomplished by subdividing the original mesh into several smaller pieces, and applying the appropriate boundary conditions along the outer boundaries of each block. This same configuration could also be modeled using the original mesh by invoking the **KIL2D** specification for the points inside the obstruction, followed by an application of the proper boundary specifications along the obstruction internally on the single-block mesh.

This boundary condition requires the specification of additional data. The variable following the label **LSTART** indicates the starting index of the **LFACE1** coordinate direction (in the example above, this would be the i coordinate direction) for the region to be “killed”. The variable following the label **LEND** indicates the final index in the **LFACE1** coordinate direction (again, the i coordinate in the example above) for the region to be “killed”. The remaining coordinate indices for the region to be “killed” are determined by the variables **M1LIM1**, **M1LIM2** for the j coordinate direction and **N1LIM1**, and **N1LIM2** for the k coordinate direction. The additional specification of the **LSTART**, **LEND** variables imply that the variables **L1LIM**, **L2LIM** are not used in this specification.

The **KIL2D** routine functions by constantly resetting the flow variables inside the region to be killed to the initial values specified by the **RMACH** input variable. So, in effect, the solution is still being performed in the region to be killed, but the updated results are constantly reset to a uniform flow value. This routine is not without drawbacks. First of all, although the mesh points are effectively neutralized by the **KIL2D** specification, other routines such as the residual smoothing algorithm are unaltered, and under certain circumstances, this may cause poor convergence. It is also possible that divergence may occur within the “killed” cells in spite of the resetting procedure. The best advice is to manipulate block structures to eliminate the need for the use of the **KIL2D** routine, but the user should be aware that under dire circumstances this facility is available. The **KIL2D** specification should be given prior to any other boundary conditions to avoid over-writing previously specified boundary specifications.

Restrictions/Limitations

The **KIL2D** boundary specification is restricted to 2-D mesh surfaces (3-D mesh surfaces should use the **KILL** boundary specification).

Common Errors

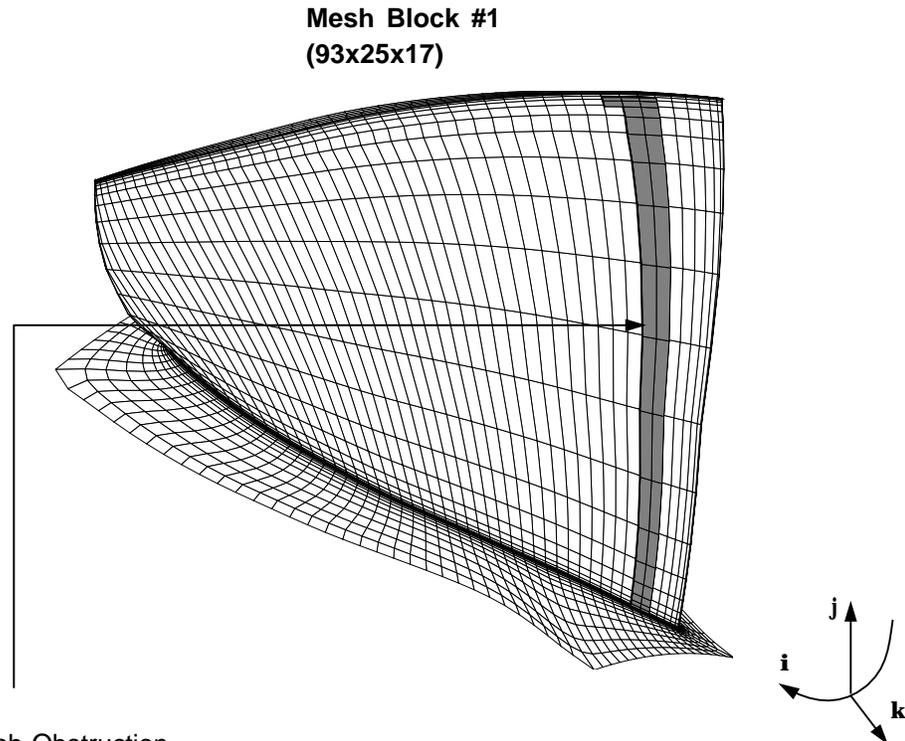
- Application of **KIL2D** to a 3-D mesh system.
- Poor convergence due to residual smoothing across a “killed” region (The residual smoothing operator can be turned off through the **RESID** input variable, although the time step must be restricted (see variable **CFL**) to maintain numerical stability).

BOUNDATA KEYWORDS

- Failure to specify the additional data values LSTART, LEND.

KILL

Solution Kill Routine



Internal Mesh Obstruction
Requires a **KILL** Specification

Application

The **KILL** keyword is a tool to effectively neutralize or “kill” the time-marching solution over a segment of the computational domain for a three-dimensional mesh. The example graphic above illustrates a single block 3-D O-type mesh system used to predict the flow through a turbomachinery compressor rotor blade passage with a surface-mounted square-edged obstruction. Rather than construct a multiple block mesh system to treat this case (whereby the obstruction is essentially gridded as block boundaries), the **KILL** specification is used to neutralize the advancing solution within the obstruction, and boundary conditions are applied along the surface of the obstruction to predict this flow.

Boundary Data File Format

The boundary data file specification for the mesh system indicated in the illustrative graphic for the **KILL** boundary condition is given below:

```
KILL 1 1 I I P P L L 49 49 1 19 1 5 1 19 1 5
```

```
LSTART  LEND
      49   51

KILL  1  1  I  I  P  P  L  L  49  49  19  21  1  5  19  21  1  5
      LSTART  LEND
      49   52
```

Note that a complete **KILL** specification requires two additional lines following the **KILL** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **KILL** specification error.

Description

In cases where a portion of a 3-D mesh does not represent a valid flow region, the **KILL** specification can be used, in conjunction with boundary conditions specified about the region to be “killed”, to effectively remove a portion of a given mesh block from the computational domain. The figure depicts a single mesh block for the flow through a high speed rotor passage upon which surface instrumentation is mounted. The blockage associated with the surface instrumentation is incorporated into the solution through the application of the appropriate boundary conditions on the surface of the instrumentation, and by applying the **KILL** procedure to negate the flow variables of the cells within the instrumentation itself. It should be noted that this effect could be accomplished by subdividing the original mesh into several smaller pieces, and applying the appropriate boundary conditions along the outer boundaries of each block.

This boundary condition requires the specification of additional data. The variable following the label **LSTART** indicates the starting index in the **LFACE1** coordinate direction (in the example above, this would be the *i* coordinate direction) for the region to be “killed”. The variable following the label **LEND** indicates the final index in the **LFACE1** coordinate direction (again, the *i* coordinate in the example above) for the region to be “killed”. The remaining coordinate indices for the region to be “killed” are determined by the variables **M1LIM1**, **M1LIM2** for the *j* coordinate direction and **N1LIM1**, and **N1LIM2** for the *k* coordinate direction. The additional specification of the **LSTART**, **LEND** variables imply that the variables **L1LIM**, **L2LIM** are not used in this specification.

The **KILL** routine functions by constantly resetting the flow variables inside the region to be killed to the initial values specified by the **RMACH** input variable. So, in effect, the solution is still being performed in the region to be killed, but the updated results are constantly reset to a uniform flow value. This routine is not without drawbacks. First of all, although the mesh points are effectively neutralized by the **KILL** specification, other routines such as the residual smoothing algorithm are unaltered, and under certain circumstances, this may cause poor convergence. It is also possible that divergence may occur within the “killed” cells in spite of the resetting procedure. The best advice is to manipulate block structures to eliminate the need for the use of the **KILL** routine, but the user should be aware that under dire circumstances this facility is available. The **KILL** specification should be given prior to any other boundary conditions to avoid over-writing previously specified boundary specifications.

Restrictions/Limitations

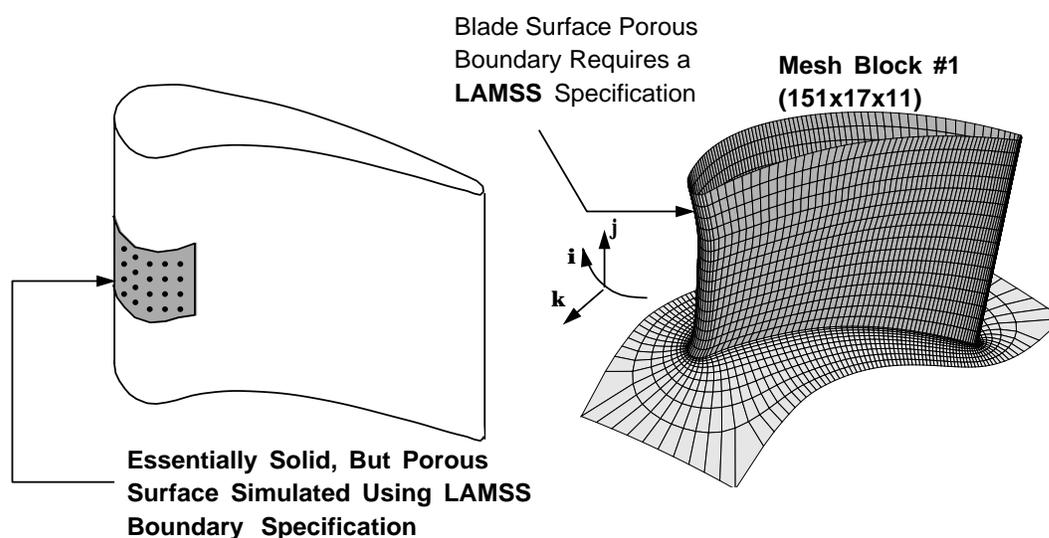
The **KILL** boundary specification is not restricted to 3-D mesh surfaces (although for consistency, 2-D mesh surfaces may use the **KIL2D** boundary specification).

Common Errors

- Application of **KILL** to a 2-D mesh system.
- Failure to specify **KILL** boundary condition prior to boundary conditions defining edges of “killed” region.
- Poor convergence due to residual smoothing across a “killed” region (The residual smoothing operator can be turned off through the **RESID** input variable, although the time step must be restricted (see variable **CFL**) to maintain numerical stability).
- Failure to specify the additional data values **LSTART**, **LEND**.

LAMSS

Porous Solid Surface Viscous No-Slip Boundary Condition



Application

The **LAMSS** specification is used to impose a porous injection, no-slip boundary condition for solid surfaces used in a viscous flow solution. The graphic above illustrates a 3-D body-centered O-type mesh system for a turbine vane cascade. The **LAMSS** specification is used to simulate the effects of a fine array of discrete cooling holes (porous injection) which are too small to be individually gridded. The **LAMSS** provides a “smeared out” normal injection which essentially simulates the global effects of the individual cooling sites.

Boundary Data File Format

The boundary data file specifications for the hub and blade surfaces in the application described above and indicated in the illustrative graphic for the **LAMSS** boundary condition are given below:

```
LAMSS 1 1 K K P P I K 1 1 1 151 1 11 1 151 1 11
PT TT RPMLOC TWALL ARATIO
1.1 0.70 0.0 0.00 0.10
```

Note that a complete **LAMSS** specification requires two additional lines following the **LAMSS** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **LAMSS** specification error.

Description

The **LAMSS** statement specifies that a solid surface viscous (no-slip) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **LAMSS** boundary condition may be applied to either a rotating or non-rotating surface and may indicate a rotational speed which is different than the rotational speed of the mesh (**RPM**) to which the boundary condition is applied (the most common example of this type of application is a mesh embedded in a rotating blade passage with an endwall which is non-rotating).

The **LAMSS** boundary condition is a blend of the **INLETG** and **SSVI** boundary conditions. For each **LAMSS** statement both a complete **INLETG** and **SSVI** surface condition are calculated and then are combined through a weighted sum based on the **ARATIO** specification. This means that if **ARATIO** is set to 0.0, the **LAMSS** boundary condition will behave exactly like a **SSVI** surface, and if **ARATIO** is set to 1.0, the surface will simulate an **INLETG** condition.

Because of this blend of boundary conditions, **LAMSS** requires the specification of additional data as found in **INLETG** and **SSVI**. The first additional line following the **LAMSS** specification is assumed to be a label. The next line contains the values imposed for the variables **PTOT**, **TTOT**, **RPMLOC**, **TWALL**, and **ARATIO**. The value of the **PTOT** and **TTOT** are the desired normalized total pressure and total temperature, respectively, of the injected flow computed as described in **INLETG**. The value of the **RPMWALL** variable is the desired solid wall dimensional rotational speed in revolutions per minute. This value is sign dependent and follows the orientation for rotation as described in Figure 3.6. The variable **TWALL** determines which type of temperature condition is applied to the surface as described in **SSVI**.

Naturally, poor convergence or solution divergence can occur if **RPMWALL** or **TWALL** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values. The value of the variable **ARATIO** represents the geometric “porosity” of the surface in the form of the ratio of open (injection) surface area to total surface area for the boundary segment being defined. In other words, if the porous surface has an injection area of 0.01 square inch per square inch of total surface, then **ARATIO** would be 0.01. **ARATIO** values less than zero or greater than 1.0 are non-physical and not permitted.

Restrictions/Limitations

The boundary rotational speed imposed by the **LAMSS** boundary condition can only be non-zero when using the cylindrical coordinate solution algorithm in the *ADPAC* code. When using the Cartesian coordinate solution algorithm **FCART** and/or **FCARB**= 1.0, the boundary rotational speed must be zero (**RPMWALL**= 0.0 when **FCART** or **FCARB**= 1.0). The injection process modeled by **LAMSS** is always *normal* to the local surface topography. Arbitrary injection angle specification is not currently possible.

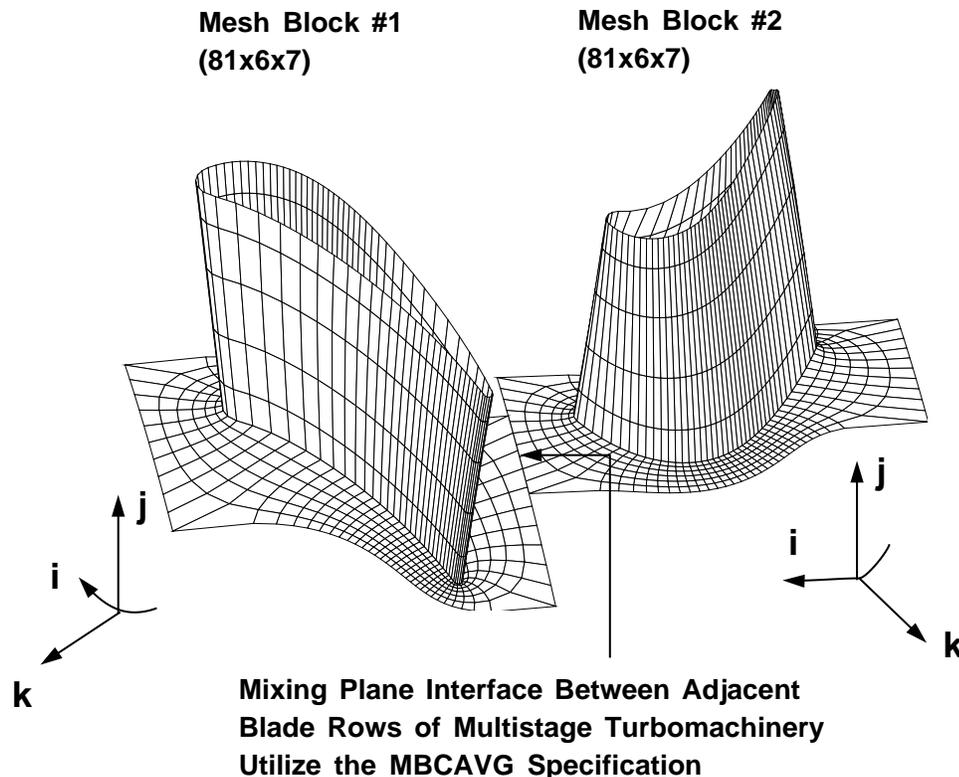
Common Errors

BOUNDATA KEYWORDS

- Incorrect sign for value of boundary rotational speed **RPMWALL**.
- Attempt to utilize a non-zero boundary rotational speed with the Cartesian coordinate solution algorithm.
- **ARATIO** value less than 0.0 or greater than 1.0.
- **PTOT**, **TTOT**, and/or **TWALL** values are significantly different than freestream.

MBCAVG

Multiple Block Circumferential Averaging Routine for Multiple Blade Row Turbomachines



Application

The **MBCAVG** specification is used in applications involving neighboring relatively rotating blade rows which may consist of one or more mesh blocks. The **MBCAVG** specification permits time-averaged interconnection between these adjacent, blade row local mesh systems based on the “mixing plane” approximation. The example illustrates the application of the **MBCAVG** boundary condition for the case of a single-stage turbine, whereby a single mesh block is used to represent a single blade passage for each blade row in the turbine stage, and the **MBCAVG** boundary routine is used to perform the mixing plane (circumferential/time-averaged) coupling of the relatively rotating blade rows.

Boundary Data File Format

The boundary data file specifications for the mesh interfaces indicated in the illustrative graphic for the **MBCAVG** boundary condition are given below. Note that block 1 requires multiple specifications due to the location of the O-grid cut line.

BOUNDATA KEYWORDS

```

MBCAVG      1  2  K  K  M  M  I  J  7  7  1  6  1  6  36  46  1  6
NSEGS
      1
LBLOCK2B    LFACE2B  LDIR2B    L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
2           K           M           7         36         46         1         6

MBCAVG      1  2  K  K  M  M  I  J  7  7  76  81  1  6  36  46  1  6
NSEGS
      1
LBLOCK2B    LFACE2B  LDIR2B    L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
2           K           M           7         36         46         1         6

MBCAVG      2  1  K  K  M  M  I  J  7  7  36  46  1  6  1  6  1  6
NSEGS
      2
LBLOCK2B    LFACE2B  LDIR2B    L2LIMB  M2LIM1B  M2LIM2B  N2LIM1B  N2LIM2B
1           K           M           7         1         6         1         6
1           K           M           7         76         81         1         6

```

Note that a complete **MBCAVG** specification generally requires at least two **MBCAVG** statement lines in the boundary data file for each mesh interface. In the example above, the first two specifications provide the inter-block communication for block 1 from block 2, and the third specification provides the communication for block 2 from block 1. It is a common error to under-specify an **MBCAVG** boundary by only providing a single line per interface.

Description

The **MBCAVG** specification provides a “circumferential mixing plane” mesh block communication scheme for steady-state (time-averaged) analysis of multiple blade row turbomachines. The **MBCAVG** operator permits the specification of multiple neighboring blocks upon which the circumferential averaging is applied to provide boundary data for the current block of interest. This multiple block averaging scheme permits the use of **MBCAVG** for body-centered mesh systems and also for multiple blade passage representations of neighboring blade rows. Due to the complex nature of the circumferential averaging operator, this boundary condition is restricted to specific mesh configurations. The following chart describes the permitted mesh configurations for the **MBCAVG** specification:

LFACE1 (Block #1 Face)	LFACE2 (Block #2 Face)	Circumferential Coordinate Direction	Grids Must be Aligned in this Coordinate
-----	-----	-----	-----
I	I or K	K or I	J
J	J only	K	I
K	I or K	K or I	J

A second mesh restriction is that the interface separating two adjacent blade rows

must be a surface of revolution, and that meshes along this interface have common axial and radial grid distributions. This restriction simplifies the averaging scheme provided by the **MBCAVG** specification.

The **MBCAVG** boundary condition requires the specification of additional data. The variable following the label **NSEGS** defines the number of neighboring mesh block surfaces from which the circumferentially averaged data is obtained. In the example, this value is simply 1 for the upstream inter-blade row boundaries, but is 2 for the downstream inter-blade row boundary because of the fact that the matching boundary of the upstream blade row is composed of two distinct mesh segments even though it is taken from a single mesh block. The next line following the **NSEGS** variable is a label indicating the variables which must be input for each of the **NSEGS** segments in the mixing plane. The variables **LBLOCK2B**, **LFACE2B**, **LDIR2B**, **L2LIMB**, **M2LIM1B**, **M2LIM2B**, **N2LIM1B**, and **N2LIM2B** represent the values of **LBLOCK2**, **LFACE2**, **LDIR2**, **L2LIM**, **M2LIM1**, **M2LIM2**, **N2LIM1**, and **N2LIM2** for *each* of the individual **NSEGS** segments used in the mixing plane construction. The segments may be specified in any order.

Restrictions/Limitations

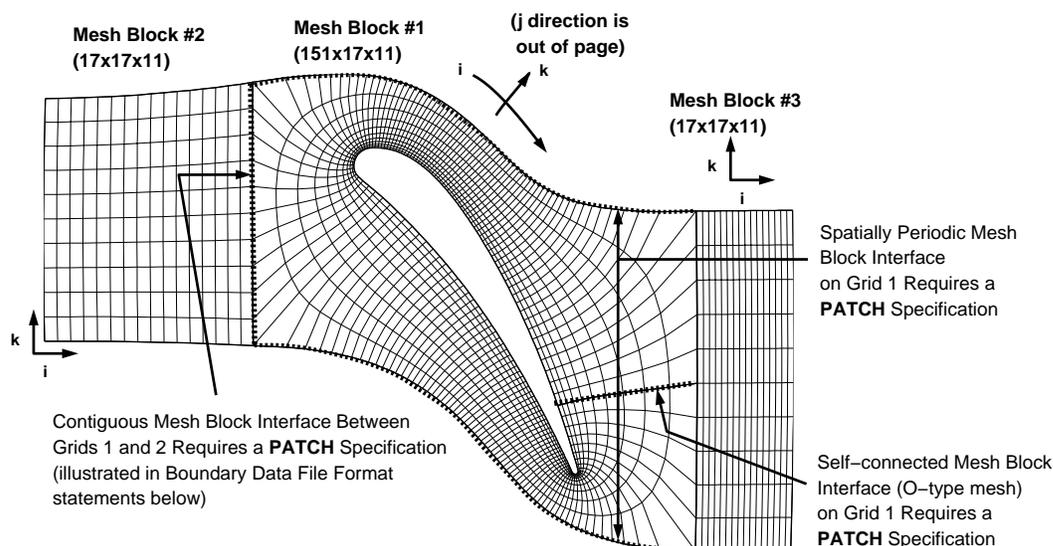
The **MBCAVG** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap), and have common axial and radial mesh coordinates. The mesh must obey the coordinate restrictions outlined in the description above.

Common Errors

- Failure to provide 2 or more **MBCAVG** statements for each inter-blade row interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1B**, **M2LIM2B**, **N2LIM1B**, **N2LIM2B** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2B**)
- Meshes do not obey the mesh coordinate restrictions listed in the description above.
- Meshes have dissimilar axial and radial coordinates at the interface.
- Application of **MBCAVG** to mesh interfaces which do not share a common surface, or which have excess overlap.
- Application of **MBCAVG** to Cartesian solution mesh systems.

PATCH

Contiguous Mesh Block Interface Patching Scheme



Application

The **PATCH** specification is used in any application involving neighboring mesh blocks with a contiguous (common mesh points) interface. The graphic above illustrates a **PATCH** connection between mesh blocks in a combination H-O-H mesh system for a turbine vane cascade. The **PATCH** boundary specification is used to provide block-to-block communication between mesh blocks 1 and 2, and mesh blocks 1 and 3, as well as providing periodic flow boundary conditions for blocks 1, 2, and 3. In addition, the **PATCH** routine is used to provide aerodynamic communication across the O-mesh branch cut for mesh block 1. The **PATCH** boundary condition is perhaps the most common specification resulting from the use of the multiple blocked mesh capabilities of the *ADPAC* code.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the example for the **PATCH** boundary condition are given below:

```
PATCH 1 2 K I M M J I 11 17 71 81 1 17 1 17 1 11 Blks #1-#2
PATCH 2 1 I K M M K J 17 11 1 17 1 11 71 81 1 17 Blks #1-#2

PATCH 1 3 K I M P J I 11 1 1 6 1 17 1 17 6 1 Blks #1-#3
PATCH 3 1 I K P M K J 1 11 1 17 1 6 6 1 1 17 Blks #1-#3
```

```

PATCH 1 3 K I M P J I 11 1 146 151 1 17 1 17 11 6 Blks #1-#3
PATCH 3 1 I K P M K J 1 11 1 17 6 11 151 146 1 17 Blks #1-#3

PATCH 1 1 K K M M I J 11 11 6 71 1 17 146 81 1 17 Blk #1 Per
PATCH 1 1 K K M M I J 11 11 81 146 1 17 71 6 1 17 Blk #1 Per

PATCH 2 2 K K P M I J 1 11 1 17 1 17 1 17 1 17 Blk #2 Per
PATCH 2 2 K K M P I J 11 1 1 17 1 17 1 17 1 17 Blk #2 Per

PATCH 3 3 K K P M I J 1 11 1 17 1 17 1 17 1 17 Blk #3 Per
PATCH 3 3 K K M P I J 11 1 1 17 1 17 1 17 1 17 Blk #3 Per

PATCH 1 1 I I P M J K 1 151 1 17 1 11 1 17 1 11 Blk #1 O-Grid
PATCH 1 1 I I M P J K 151 1 1 17 1 11 1 17 1 11 Blk #1 O-Grid

```

Note that a complete **PATCH** specification generally requires two **PATCH** statement lines in the boundary data file. For any two grid blocks (1 and 2 for example), the first specification provides the inter-block communication for block 1 from block 2, and the second specification provides the communication for block 2 from block 1. It is a common error to under-specify a **PATCH** boundary by only providing a single line per interface.

Description

The **PATCH** statement is utilized to provide direct block-to-block communication between mesh blocks with contiguous grid points. This is perhaps the most common, and most useful of the boundary condition specifications. For many complicated geometries requiring a multiple block mesh system, a common approach is to generate mesh systems with coincident mesh points along all, or at least part of the mesh block interfaces. This property is henceforth referred to as a contiguous mesh block interface (coincident mesh points). By default, the boundary condition specification *must* have a one-to-one correspondence between mesh points in block **LBLOCK1** and mesh points in block **LBLOCK2**. This type of boundary is particularly effective in the finite-volume flow solver due to the fact that local and global conservation of the flow variables can be accomplished without special treatment, by direct substitution of the neighboring block flow variables into the phantom cells of the block of interest.

The **PATCH** boundary condition performs this direct substitution between blocks to provide an aerodynamic communication between neighboring blocks with a contiguous interface. A **PATCH** specification can also be imposed connecting a block to itself. The **PATCH** boundary condition requires no additional data beyond the initial specification line, but does require the proper specification of the variables **LSPEC1** and **LSPEC2**. For boundary conditions involving more than one mesh block, it is possible that the connection between blocks may involve communication between different grid surfaces (i.e., an i =constant mesh face in **LBLOCK1** connects to a j =constant mesh face in **LBLOCK2**) and that the remaining indices in block **LBLOCK2** correspond to different coordinates in block **LBLOCK1**. The specification of the variables **LSPEC1** and **LSPEC2** serve to eliminate any confusion between contiguous boundary patches involving dissimilar mesh coordinates. In every case, when a particular coordinate direction is specified by the variable **LFACE1**, the

remaining coordinate indices defining the extent of the patch on **LFACE1** are specified in their “natural” (i, j, k) order (see the description of **LSPEC1** at the beginning of Section 3.7).

In order to relate the coordinate indices in block **LBLOCK2** with the indices specified in block **LBLOCK1**, the special terms **LSPEC1** and **LSPEC2** are utilized. The variables **LSPEC1** and **LSPEC2** should be defined as either **I**, **J**, or **K**, based on the connection scheme between the two blocks. The **LSPEC1** variable should define the coordinate direction in block **LBLOCK1** which corresponds to the first remaining coordinate in block **LBLOCK2** (whose range is defined by **M2LIM1**, **M2LIM2**), and the **LSPEC2** variable should define the coordinate direction in block **LBLOCK1** which corresponds to the second remaining coordinate in block **LBLOCK2** (whose range is defined by **N2LIM1**, **N2LIM2**). The **PATCH** specification may also be used for two-dimensional meshes as long as the third coordinate direction (k) limits **N1LIM1**, **N1LIM2**, and **N2LIM1**, **N2LIM2** are “1” and “2”, respectively (2-D patches are specified as if the mesh were actually one cell deep in the k direction spanning grid points 1 to 2).

An *ADPAC* utility program called *PATCHFINDER* was developed to greatly reduce the time and effort needed to create an *ADPAC* boundary condition file (*case.boundata*). This utility is described in further detail in Section 6.6. Because the **PATCH** boundary condition is one of most frequently used boundary specifications in *ADPAC*, a more detailed explanation of the three most likely uses of the **PATCH** statement follows.

Connecting Two Blocks

The most obvious use of the **PATCH** statement is allowing communication between two separate but adjacent blocks. The first three pairs of **PATCH** statements in the example deal with this type of communication. For example, the first pair of **PATCH** statements sets up the communication between block 1 (vane mesh) and block 2 (upstream mesh). The first **PATCH** statement specifies that the **K**-face of block 1 with index 11, ranging from $i = 71$ to 81 and $j = 1$ to 17, maps to the **I**-face of block 2 with index 17, ranging from $j = 1$ to 17 and $k = 1$ to 11. The **LSPEC**'s are reversed from natural order in this example (**J I**), because the first set of indices from the second block specifying the j range match with the second set (or **J**) of indices from the first block; likewise, the second set of indices from the second block specifying the k range match with the first set (or **I**) of indices from the first block.

Periodic Boundaries

In several turbomachinery applications, periodic boundary conditions are used to simulate the effect of neighboring blades or repeated geometric features. In the above example, the fourth through the sixth set of **PATCH** statement pairs are examples of this type of use. For example using the fifth pair of **PATCH** statements describing the periodic boundary for block 2, the first **PATCH** statement of the pair specifies that the **K**-face of block 2 with index 1 (k_{min}), ranging from $i = 1$ to 17 and $j = 1$ to 17, maps to the **K**-face of block 2 with index 11 (k_{max}), ranging from $j = 1$ to 17 and $k = 1$ to 17. The two surfaces are swapped to create the complimentary **PATCH** statement completing the pair. Whereas a

similar usage is employed with block 3, block 1 requires some additional considerations such as reversed indices in order to ensure proper alignment.

Connecting One Block to Itself

In some types of mesh topologies (i.e., O-mesh and C-mesh), a single block will wrap back upon itself creating a branch cut. In the above example, the final pair of **PATCH** statements is an example of this type of use. The first **PATCH** statement specifies that the I-face of block 1 with index 1 (i_{min}), ranging from $j = 1$ to 17 and $k = 1$ to 11, maps to the I-face of block 1 with index 151 (i_{max}), ranging from $j = 1$ to 17 and $k = 1$ to 11. These two surfaces are swapped to create the complimentary **PATCH** statement completing the pair.

Restrictions/Limitations

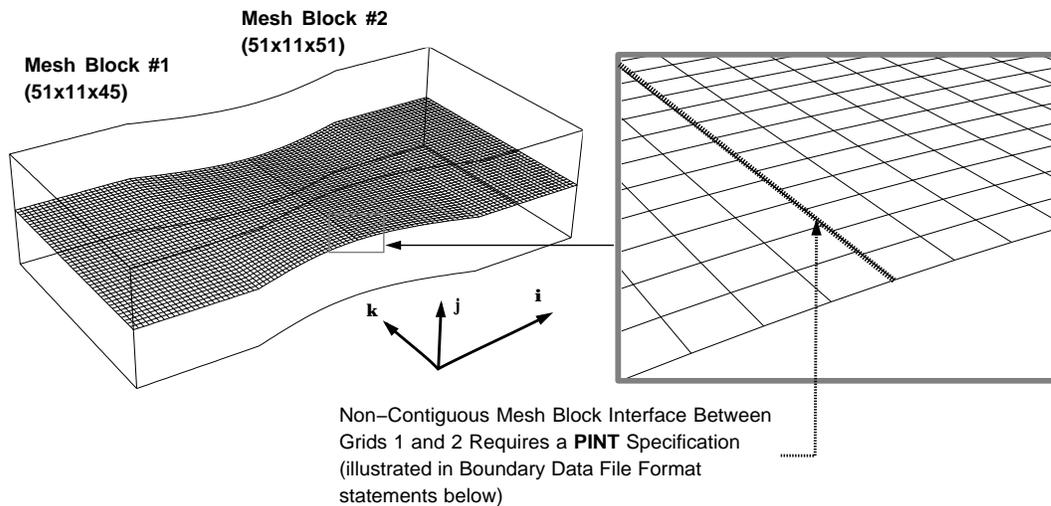
The **PATCH** boundary specification is restricted to mesh interfaces which have a one-to-one mesh point correspondence. To maintain the conservative property of the governing equations, the **PATCH** routine assumes that the mesh points between the two blocks of interest are either contiguous, or share a spatially periodic relationship, and it is left to the user to verify this.

Common Errors

- Failure to provide 2 **PATCH** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2 do not correctly define the interface extents on blocks LBLOCK1 and LBLOCK2).
- Incorrectly specified block coordinate direction relationships (values of LSPEC1, LSPEC2 do not correctly define the coordinate connection scheme between block LBLOCK1 and block LBLOCK2).
- **PATCH** boundary specification for a periodic boundary is applied to a non-periodic mesh.
- **PATCH** boundary specification applied to a spatially periodic Cartesian geometry using the cylindrical coordinate solution scheme or vice versa (results in incorrect spatial periodicity relationships) The **PATCH** boundary specifications for Cartesian geometries must use the Cartesian solution algorithm in *ADPAC* (see input variable **FCART**).

PINT

Non-Contiguous Mesh Block Interface Patching Scheme



Application

The **PINT** specification is used in any application involving neighboring mesh blocks which share a common mating surface (either contiguous or non-contiguous). The example graphic above illustrates a two-dimensional plane of a two block 3-D mesh system used to predict the flow through a converging/diverging nozzle. The mesh points at the interface between the two grids (near the nozzle throat) are non-contiguous, and therefore, the **PINT** specification is used to provide communication between the adjacent mesh blocks.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **PINT** boundary condition are given below:

```
PINT 1 2 I I M P L L 51 1 1 11 1 45 1 11 1 51
PINT 2 1 I I P M L L 1 51 1 11 1 51 1 11 1 45
```

Note that a complete **PINT** specification generally requires two **PINT** statement lines in the boundary data file. In the example above, the first specification provides the inter-block communication for block 1 from block 2, and the second specification provides the communication for block 2 from block 1. It is a common error to under-specify a **PINT** boundary by only providing a single line per interface.

Description

The **PINT** boundary statement provides a means for block-to-block communication for cases involving neighboring meshes which share a common surface, but not necessarily common grid points along a block boundary (meshes with contiguous mesh points should use the **PATCH** boundary specification, meshes with contiguous points in one coordinate direction should use the **BCINT1** boundary specification). The **PINT** specification instructs the *ADPAC* code to perform a weighted interpolation to determine the appropriate flow variables for the phantom cells, based on the non-contiguous data structure of the neighboring mesh. The bounding surfaces of each block should lie on a common surface (no significant overlap). The interpolation scheme used in the **PINT** specification is not conservative, and therefore the solution accuracy can be degraded by this procedure.

During code execution, the first time the **PINT** specification is encountered, the code initiates a search to determine the interpolation stencil for the given array of points in block **LBLOCK1** based on the data in block **LBLOCK2**. This stencil is then saved to eliminate the search routine at every application of **PINT**. In order to provide storage for the interpolation stencil information, a separate array system based on the dimensioning parameter **NRAINT** is utilized. The **PINT** boundary condition requires no additional data beyond the initial specification line, but does require some extra care when used. The primary precaution is that the **PINT** procedure is based entirely on a simplified interpolation scheme, and hence, does not maintain either global or local conservation of flow variables across the mesh interface.

Restrictions/Limitations

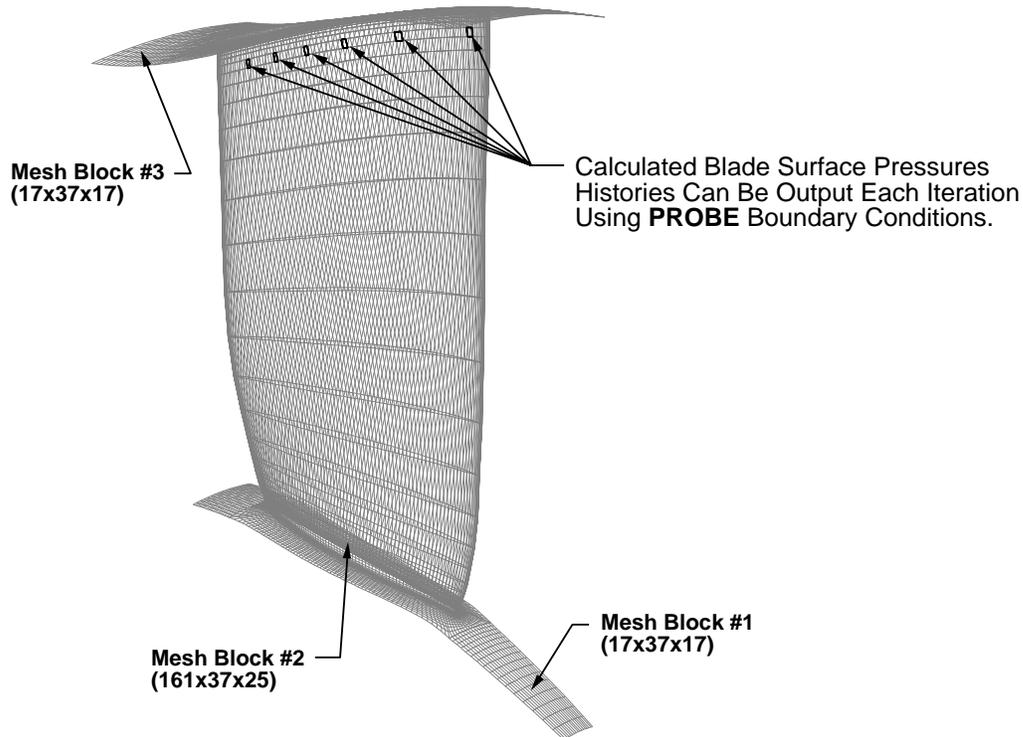
The **PINT** boundary specification is restricted to mesh interfaces which lie on a common surface (no significant overlap). The **PINT** procedure is only applicable to 3-D mesh systems. **PINT** can not be used across multiple processors in a parallel computing environment.

Common Errors

- Failure to provide 2 **PINT** statements for each interface.
- Incorrectly specified or misaligned extents of boundary regions (values of **M1LIM1**, **M1LIM2**, **N1LIM1**, **N1LIM2**, **M2LIM1**, **M2LIM2**, **N2LIM1**, **N2LIM2** do not correctly define the interface extents on blocks **LBLOCK1** and **LBLOCK2**).
- Attempt to use **PINT** for a periodic boundary (no special spatial periodicity arrangement is available in **PINT**).
- Attempt to use **PINT** on a 2-D mesh block.
- Failure to provide enough storage for the **PINT** interpolation stencils via the **NRAINT** parameter.
- Application of **PINT** to mesh interfaces which do not share a common surface, or which have excess overlap.

PROBE

Flow Variable Sampling Area Specification



Application

The **PROBE** specification is used to track the time history of the flow variables at specific locations or areas within the computational domain. The example graphic above illustrates a fan blade with six separate **PROBE** specifications near the tip. This series of boundary conditions will output a separate ASCII file for each **PROBE**.

Boundary Data File Format

The boundary data file specifications for the mesh in the example for the **PROBE** boundary condition are given below:

```

PROBE      2  2  K  K  P  P  M  M      1  1      5  6  25  26      5  6  25  26
PROBE      2  2  K  K  P  P  M  M      1  1     15 16  25  26     15 16  25  26
PROBE      2  2  K  K  P  P  M  M      1  1     25 26  25  26     25 26  25  26
    
```

```

PROBE      2  2 K K P P M M      1  1  35 36  25 26  35 36  25 26
PROBE      2  2 K K P P M M      1  1  45 46  25 26  45 46  25 26
PROBE      2  2 K K P P M M      1  1  55 56  25 26  55 56  25 26

```

Note that a complete **PROBE** specification do not have to align with multi-grid boundaries, as they are only evaluated on the finest mesh level. In addition to single cell specifications, larger areas spanning several cells may be defined.

Description

The **PROBE** boundary statement was developed to act as the numerical equivalent to an experimental test probe. The user may specify several locations of interest within the computational domain, and at each iteration on the fine mesh, averages of flow variables are output to a separate file. These time histories of the flow can then be plotted to analyze the developing flow similar to how experimental data might be used.

The **PROBE** boundary specification consists of a single line without any additional information following. The flow variables are averaged across the cells on the LFACE1 of block LBLOCK1 over the range M1LIM1 to M1LIM2 and N1LIM1 to N1LIM2. One of three types of averaging can be employed using a **PROBE** statement: area averaging, mass averaging, or bi-directional mass averaging. The averaging scheme is determined by the LSPEC1 variable (A for area averaging, M for mass averaging, or B bi-directional mass averaging). As the names imply, area averaging and mass averaging use areas and mass flux to weight the averages, respectively. The bi-directional mass averaging using mass averaging, but output two values for each iteration depending on the flux vector through the cell face (one for positive flow, one for negative flow).

As noted above, for each **PROBE** statement a separate output file is generated. The files are named *case.probe.num*, where *num* is the number of the boundary condition as read into *ADPAC*. This **PROBE** file will contain a few header lines describing the location of the **PROBE** followed by data in multiple column format. After each iteration, the following flow variables are output for a cylindrical mesh block: iteration count, time, mass flow, total pressure, total temperature, static pressure, static temperature, axial velocity, radial velocity, and tangential velocity. For a Cartesian mesh block, the radial and tangential velocities are replaced with y-direction and z-direction velocities, respectively.

Restrictions/Limitations

The **PROBE** boundary conditions are *not* limited by the multi-grid restrictions imposed on all other boundary conditions.

Common Errors

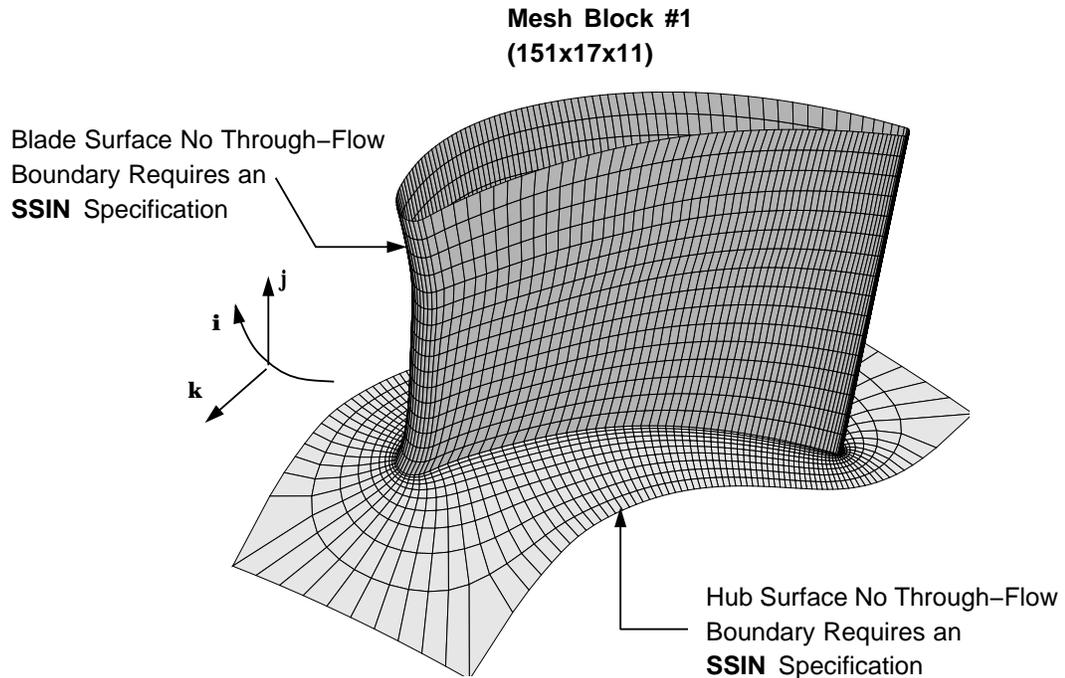
- Incorrectly specified or misaligned extents of boundary region (values of M1LIM1, M1LIM2, N1LIM1, N1LIM2, M2LIM1, M2LIM2, N2LIM1, N2LIM2 do not correctly define the area of interest on blocks LBLOCK1).

BOUNDATA KEYWORDS

- FORTRAN compilers which do not support **APPEND** access to data files may not permit **PROBE** use.

SSIN

Solid Surface Inviscid No-Through-Flow Boundary Condition



Application

The **SSIN** specification is used to impose a no-through-flow inviscid solid surface condition for any solid surface in a solution. The graphic above illustrates a 3-D body-centered O-type mesh system for a turbine vane cascade.

Boundary Data File Format

The boundary data file specification for the mesh boundary indicated in the example for the **SSIN** boundary condition is given below:

```
SSIN  1  1  J  J  P  P  I  K  1  1  1  151  1  11  1  151  1  11
SSIN  1  1  K  K  P  P  I  K  1  1  1  151  1  17  1  151  1  17
```

No additional data beyond the boundary data file descriptor is required.

Description

The **SSIN** statement specifies that a solid surface inviscid (no through-flow) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **SSIN** boundary condition may be applied to either rotating or non-rotating surfaces. The rotational speed of the boundary is irrelevant for an inviscid surface on a properly defined mesh; either the boundary rotates with the mesh, or, in the case where the rotational speeds of the mesh and boundary differ, there is no difference in the application of an inviscid surface boundary condition. The **SSIN** algorithm imposes no flow normal to the local mesh surface, but permits tangential velocity components at the boundary. The current **SSIN** algorithm is based on a loose specification of the local static pressure ($\frac{\partial p}{\partial n} = 0$) and is known to introduce some non-physical loss. However, it has been the authors experience that this formulation provides the best mix of accuracy and reliability for most applications. It should be noted that the **SSIN** boundary condition is also very useful as a method of imposing a symmetry plane in a solution for geometries which possess spatial symmetry. Naturally, the mesh must be generated in a manner which represents this spatial symmetry as well.

Restrictions/Limitations

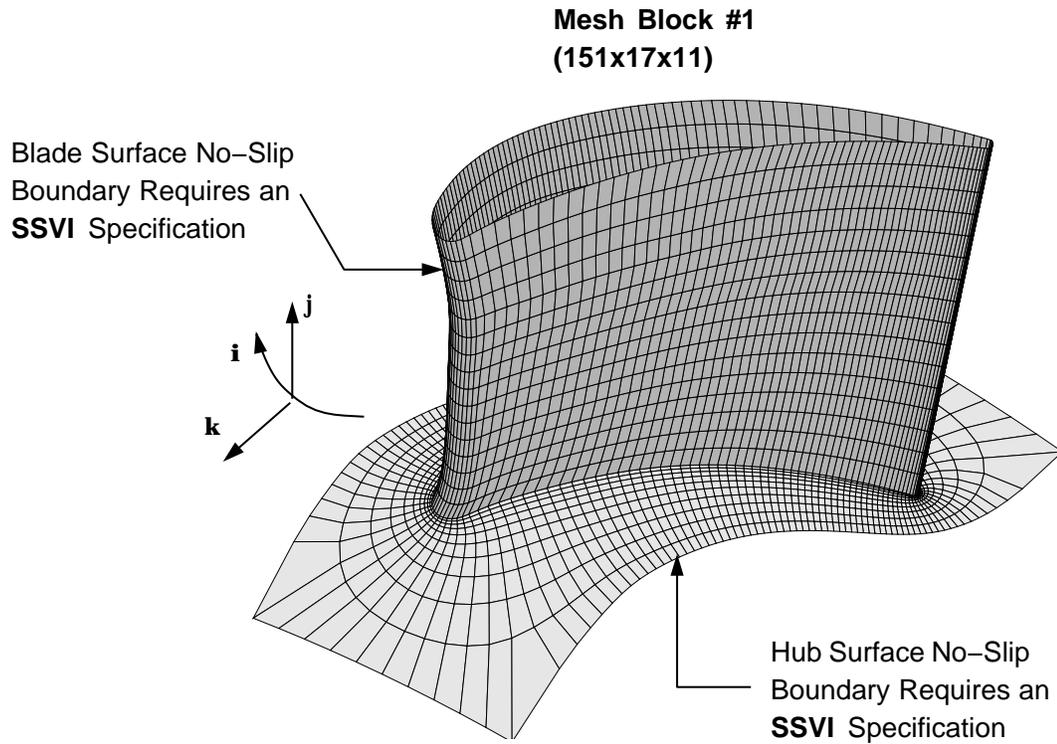
The **SSIN** boundary specification is applicable to 2-D and 3-D mesh surfaces (2-D mesh surfaces could also use the **SS2DIN** boundary specification in the same manner).

Common Errors

- Application of **SSIN** as a symmetry plane condition for a mesh which does not represent a spatially symmetric geometry

SSVI

Solid Surface Viscous No-Slip Boundary Condition



Application

The **SSVI** specification is used to impose a no-slip boundary condition for solid surfaces used in a viscous flow solution. The graphic above illustrates a 3-D body-centered O-type mesh system for a turbine vane cascade. For turbomachinery calculations, the **SSVI** specification is normally used to define the blade and endwall surfaces (both rotating and non-rotating surfaces).

Boundary Data File Format

The boundary data file specifications for the hub and blade surfaces in the application described above and indicated in the example for the **SSVI** boundary condition are given below:

```
SSVI 1 1 J J P P I K 1 1 1 151 1 11 1 151 1 11
RPMWALL TWALL
0.0 0.0
```

```

SSVI   1   1   K   K   P   P   I   K   1   1   1 151   1   17   1 151   1   17
RPMWALL  TWALL
0.0      0.0

```

Note that a complete **SSVI** specification requires two additional lines following the **SSVI** boundary data file specification line. Failure to properly specify the data in these additional lines is a common **SSVI** specification error.

Description

The **SSVI** statement specifies that a solid surface viscous (no-slip) boundary condition is to be applied to the mesh surface specified by **LFACE1** on the block specified by **LBLOCK1**. The **SSVI** boundary condition may be applied to either a rotating or non-rotating surface and may indicate a rotational speed which is different than the rotational speed of the mesh (**RPM**) to which the boundary condition is applied (the most common example of this type of application is a mesh embedded in a rotating blade passage with an endwall which is non-rotating).

This boundary condition requires the specification of additional data. The first additional line following the **SSVI** specification is assumed to be a label. The next line contains the values imposed for the variables **RPMWALL** and **TWALL**. The value of the **RPMWALL** variable is the desired solid wall dimensional rotational speed in revolutions per minute. This value is sign dependent and follows the orientation for rotation as described in Figure 3.6. The variable **TWALL** determines which type of temperature condition is applied to the surface. If **TWALL** = 0.0, an adiabatic wall is assumed. For **TWALL** > 0.0, a constant temperature surface with a nondimensional wall temperature of **TWALL** defined as:

$$(T_{wall})_{non-dimensional} = \frac{T_{wall}}{T_{ref}}$$

is imposed. T_{ref} is the reference temperature imposed by the input file variable **TREF**. A value of **TWALL** < 0.0 is not permitted. Naturally, poor convergence or solution divergence can occur if **RPMWALL** or **TWALL** suggest boundary values which are significantly different from the remainder of the flowfield. In such cases where this occurs, it is recommended that the solution be started with more conservative boundary values, and then restarted using the final boundary values.

Restrictions/Limitations

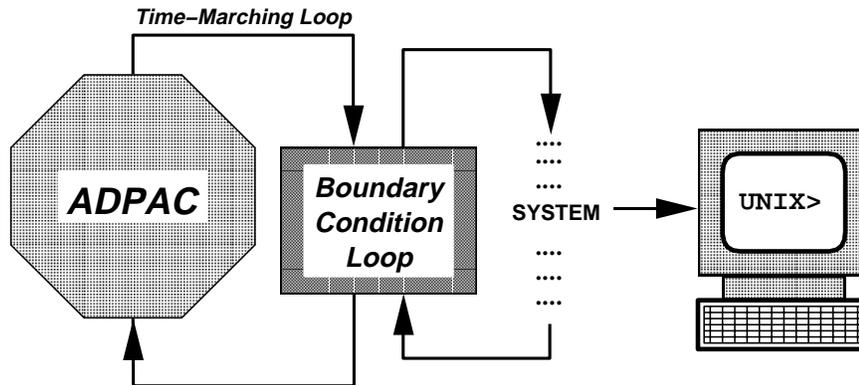
The **SSVI** boundary specification is applicable to both 2-D and 3-D mesh surfaces (2-D mesh surfaces could also use the **SS2DVI** boundary specification). The boundary rotational speed imposed by the **SSVI** boundary condition can only be nonzero when using the cylindrical coordinate solution algorithm in the *ADPAC* code. When using the Cartesian coordinate solution algorithm **FCART** or **FCARB** = 1.0, the boundary rotational speed must be zero (**RPMWALL** = 0.0 when **FCART** or **FCARB** = 1.0). Refer to input file parameters for a description of **TREF**, **RPM**, **FCARB**, and **FCART**.

Common Errors

- Incorrect sign for value of boundary rotational speed `RPMWALL`.
- Attempt to utilize a non-zero boundary rotational speed with the Cartesian coordinate solution algorithm (`FCART=1.0`).

SYSTEM

ADPAC UNIX System Call Specification



Application

The **SYSTEM** specification is not a boundary condition, but directs the *ADPAC* code to perform a UNIX system call at every application of the boundary condition loop. In the application illustrated above, every time the *ADPAC* code encounters the boundary condition specification **SYSTEM** the code is directed to perform a UNIX system call of the command *updatebc*, which is presumably a user-specified code used to update certain boundary variables (see sample format below). This new data could then be imported using the **BDATIN** boundary specification. The **SYSTEM** function can quickly lead to trouble due to the number of times it is called within the time-marching strategy, and the user should thoroughly review the documentation below before attempting to use this facility.

Boundary Data File Format

The boundary data file specifications for the mesh interface indicated in the illustrative graphic for the **SYSTEM** boundary condition are given below:

```
SYSTEM 1 1 J J P P I K 1 1 11 21 1 11 11 21 1 11
INTERVAL
1
COMMAND
/usr/local/bin/updatebc
```

Note that a complete **SYSTEM** specification requires the specification of additional data beyond the standard boundary specification line.

Description

The **SYSTEM** statement is provided to permit the specification of a UNIX system call from within the *ADPAC* code. Once the **SYSTEM** specification is directed into the *ADPAC* code, at a specified interval of iterations, during every execution of the boundary condition loop, when the **SYSTEM** specification is encountered, the code executes the command provided by the **SYSTEM** specification and pending successful completion, continues execution. The **SYSTEM** specification is based on the FORTRAN intrinsic *system* function which must be available in the compiling system. It should be noted that the command dictated by the **SYSTEM** specification could be performed *every* time the boundary condition loop is encountered. This suggests that the system call could be made a minimum of four times for each iteration of the time-marching scheme (for the four-stage scheme). This number can grow rapidly if more complicated iteration strategies are used such as multi-grid and subiterations. The user should be warned that such redundant system calls can wreak havoc on an otherwise friendly solution. A **SYSTEM** specification, in conjunction with the **BDATIN/BDATOU** boundary data specifiers can be effectively combined to provide a run time interface between the *ADPAC* code and an external flow solver.

A **SYSTEM** specification requires four additional lines in addition to the normal boundary data file descriptor, as shown above. The first additional line simply contains the label for the iteration interval **INTERVAL**, followed by the actual value of **INTERVAL**. The **SYSTEM** routine will only be invoked every **INTERVAL** time-marching iterations. The next line contains the label for the system call command (**COMMAND**) variable. The following line contains the actual UNIX command to be issued at every **SYSTEM** encounter in the boundary condition loop.

Restrictions/Limitations

Data provided in the **SYSTEM** specification should represent a valid UNIX system command. The FORTRAN intrinsic function *system* must be available on the compiling system.

Common Errors

- Invalid UNIX system command provided in **SYSTEM** boundary specification.
- Failure to provide the additional data **INTERVAL** and **COMMAND** for **SYSTEM** specification.
- FORTRAN intrinsic function *system* unavailable at compile time.
- User unaware that **SYSTEM** action occurs four or more times per iteration.

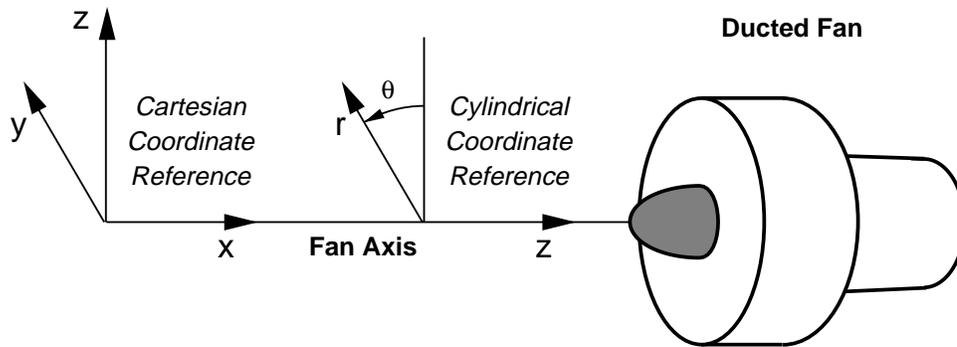
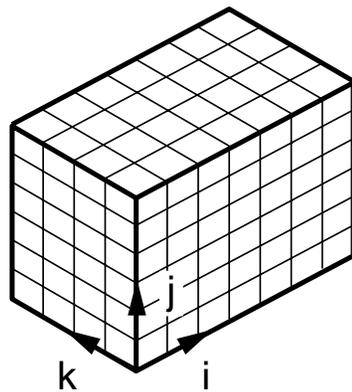


Figure 3.14: ADPAC mesh coordinate reference description.

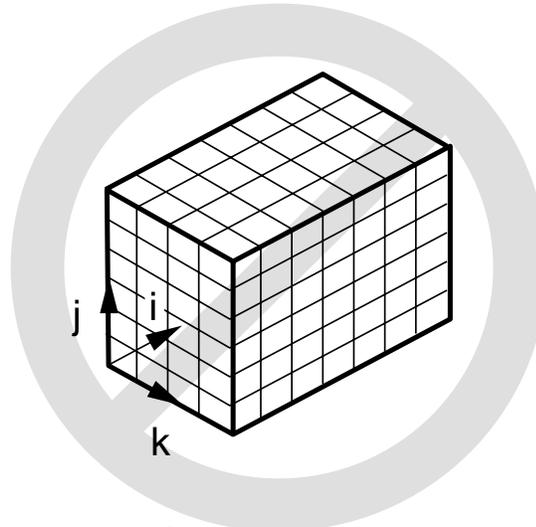
3.8 Mesh File Description

The *ADPAC case.mesh* file is a data file containing the x, y, z grid coordinates of the multiple mesh blocks which define the physical grid points used in the time-marching solution. The mesh coordinates are specified in a Cartesian reference frame, as shown in Figure 3.14, although the *ADPAC* program may ultimately convert these coordinates to a cylindrical coordinate system during execution. Regardless of whether the user intends to utilize the *ADPAC* code in a Cartesian or cylindrical solution mode, the mesh file coordinates are *always* defined by the Cartesian coordinates x, y , and z . The mesh coordinates are stored in what is known as *PLOT3D* multiple grid format, and are formatted using the Scientific Database Library (*SDBLIB*) [16, 17]. The *SDBLIB* system allows machine-independent binary file storage. The *case.mesh* file *must* be available for every *ADPAC* run. At the beginning of program execution, the *ADPAC* program attempts to open the mesh file and read in the mesh size to make sure that enough memory has been allocated for the given problem. If the mesh file is not found or if the mesh is too large, the appropriate error message is issued and the program will terminate.

Mesh coordinates are *assumed* to be nondimensional numbers. The *ADPAC* code employs a dimensional scaling factor (see input keyword **DIAM**) to convert the nondimensional mesh coordinates into dimensional coordinates with units of feet. Proper nondimensionalization and specification of the dimensionalizing factor **DIAM** is required in order to accurately achieve the desired flow Reynolds number and rotational speed (see input keyword **ADVR**). It is also required that the ordering of the mesh points form a “left-handed” mesh. This implies that at every point in the mesh, the vectors representing the positive i, j , and k coordinate directions form a left-handed coordinate system (see Figure 3.15). Consider the case of a sheared H-grid discretizing a single blade passage of a compressor. If one assumes that looking downstream through the blade passage is essentially the positive i direction, and that the radial direction from hub to tip is essentially the positive j direction, a left-handed mesh would require that the positive k direction be from right to left (counter-clockwise) in this orientation. In general, a mesh that will maintain a stable solution should contain the following characteristics: left-handed coordinates, matching surfaces at block interfaces (no overlap), minimal grid shear, and mesh expansion ratios less than 1.3. With some geometries, these criteria may be more difficult to reach, but are provided as a guide to the user.



Left-Handed Mesh



Right-Handed Mesh

Figure 3.15: All *ADPAC* mesh systems **must** have a left-handed coordinate system description.

The mesh file may be utilized directly with the *PLOT3D* program when the default real number size of the compiled *PLOT3D* code is defined as 32 bits (as it is on many workstations). The corresponding *PLOT3D* read command for an *ADPAC* mesh file is: `read/mg/bin/x=case.mesh`. The user should substitute his/her own case name in the *PLOT3D* input line. Unformatted mesh files may be converted to *ADPAC* format using the *MAKEADGRID* program described in Chapter 6.

In order to understand the *PLOT3D* multiple-grid mesh file format, and the utilization of the *SDBLIB* routines, a comparison of the FORTRAN coding for each method is given below for comparison. The x, y, z coordinates are read in as a single-dimensioned array in the *SDBLIB* format, and the *ADPAC* program includes a conversion routine (source file `convas.f`) which converts the single dimension array data to a three-dimensional data array. The FORTRAN coding to read a *PLOT3D* unformatted multiple-block mesh file might be given as:

```

OPEN(UNIT=IGRID, FILE=FNAME, FORM='UNFORMATTED', STATUS='OLD')
READ(IGRID) MG
READ(IGRID) (IL(L), JL(L), KL(L), L=1, MG)
DO 10 L = 1, MG
  READ(IGRID) (((X(I, J, K, L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L)),
  .           ((Y(I, J, K, L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L)),
  .           ((Z(I, J, K, L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L))
10 CONTINUE

```

Each of the terms used in the FORTRAN code given above are defined below:

IGRID FORTRAN logical unit number for read statement
 FNAME File name for mesh file
 MG Number of grid blocks

IL(L) Maximum *i* grid index for block L
 JL(L) Maximum *j* grid index for block L
 KL(L) Maximum *k* grid index for block L
 X(I,J,K,L) Cartesian coordinate value of x for point (I,J,K) in block L
 Y(I,J,K,L) Cartesian coordinate value of y for point (I,J,K) in block L
 Z(I,J,K,L) Cartesian coordinate value of z for point (I,J,K) in block L

An example of the corresponding FORTRAN coding to read an *ADPAC* binary mesh file using the *SDBLIB* routines is given below:

```

CALL QDOPEN( IGRID, FNAME, JE )
CALL QDGETI( IGRID, MG , JE )
DO 10 L = 1, MG
  CALL QDGETI( IGRID, IL(L), JE )
  CALL QDGETI( IGRID, JL(L), JE )
  CALL QDGETI( IGRID, KL(L), JE )
10 CONTINUE
IPOINT = 1
DO 20 L = 1, MG
  ILENGTH = IL(L) * JL(L) * KL(L)
  CALL QDGEEA( IGRID, X(IPOINT), ILENGTH, JE )
  CALL QDGEEA( IGRID, Y(IPOINT), ILENGTH, JE )
  CALL QDGEEA( IGRID, Z(IPOINT), ILENGTH, JE )
  IPOINT = IPOINT + ILENGTH
20 CONTINUE
CALL QDCLOS( IGRID, JE )
  
```

A listing of the additional terms used in the coding above is given below:

QDOPEN *SDBLIB* routine to open a file for input or output
 QDGETI *SDBLIB* routine to get an integer
 QDGEEA *SDBLIB* routine to get a real array of length ILENGTH
 QDCLOS *SDBLIB* routine to close a file
 IGRID FORTRAN logical unit number for grid input
 JE Integer error trigger; 0 for no error, $\neq 0$ if an error occurs
 ILENGTH Integer length of an array of data
 IPOINT Integer pointer for block L to locate the initial memory location for a block of data

3.9 Body Force File Description

The series of *ADPAC* body force files are data files containing the blade blockage, body force, and energy source terms for each individual mesh block used in a 2-D axisymmetric representation of an embedded blade row (see Section 2.3). The file naming procedure for the body force file is somewhat different than the mesh, plot, and restart files, where a single file contains all the data for a multiple-block solution. A separate file is used for

each mesh block employing body forces. Hence the file name is *case.bf.#* where # represents the block number to which the body forces should be applied.

The terms in the body force file are stored in binary format, based on the *SDBLIB* routines. The blockage, body forces, and energy sources are stored as nondimensional numbers using the nondimensionalization strategy described in Appendix A.

The body force data are read and written as a single-dimensioned array in the *SDBLIB* format, and the *ADPAC* program includes a conversion routine (source file *convaf.f*) which converts the one-dimensional array data to three dimension array data. In order to understand the body force file format and the utilization of the *SDBLIB* routines, a representative FORTRAN coding example to read in a body force file is given below:

```
CALL QDOPEN( IBODY, FNAME, JE )
CALL QDGETI( IBODY, NG , ILENGTH, JE )
CALL QDGETI( IBODY, IMX, ILENGTH, JE )
CALL QDGETI( IBODY, JMX, ILENGTH, JE )
CALL QDGETI( IBODY, KMX, ILENGTH, JE )
ILENGTH = IMX * JMX * KMX
CALL QDGETE( IBODY, DUMMY, JE )
CALL QDGEEA( IBODY, BFR (IPOINT(L)), ILENGTH, JE )
CALL QDGEEA( IBODY, BFRU (IPOINT(L)), ILENGTH, JE )
CALL QDGEEA( IBODY, BFRV (IPOINT(L)), ILENGTH, JE )
CALL QDGEEA( IBODY, BFRW (IPOINT(L)), ILENGTH, JE )
CALL QDGEEA( IBODY, BFRE (IPOINT(L)), ILENGTH, JE )
CALL QDGEEA( IBODY, BL (IPOINT(L)), ILENGTH, JE )
CALL QDGEEA( IBODY, BP (IPOINT(L)), ILENGTH, JE )
CALL QDCLOS( IBODY, JE )
```

A listing of the FORTRAN variables and their meanings is given below:

```
QDOPEN  SDBLIB routine to open a file for input or output
QDGETI  SDBLIB routine to get an integer
QDGETE  SDBLIB routine to get a real number
QDGEEA  SDBLIB routine to get a real array of length ILENGTH
QDCLOS  SDBLIB routine to close a file
IBODY   FORTRAN logical unit number for body force file input
JE      Integer error trigger; 0 for no error, ≠ 0 if an error occurs
NG      Number of blocks in body force file (must be 1)
IMX     Mesh size+1 in the i coordinate direction
JMX     Mesh size+1 in the j coordinate direction
KMX     Mesh size+1 in the k coordinate direction
ILENGTH Integer length of an array of data
```

IPOINT(L)	Integer pointer for block L to locate the initial memory location for a block of data
BFR	Body force for density (continuity equation)
BFRU	Body force for axial momentum
BFRV	Body force for radial momentum
BFRW	Body force for circumferential momentum
BFRE	Body force for internal energy
BL	Blockage term
BP	Pressure correction term (currently unused)

3.10 Standard Output File Description

The *ADPAC* standard output file *case.output* provides information regarding the status of a particular calculation during code execution. The status information includes startup, code response to input files (mesh, restart, standard input, and boundary data), convergence history, error messages, and output file generation (plot files, restart files, body force files). The information given in the standard output file is essentially self explanatory, so no further description is given here.

3.11 Plot File Description

The *ADPAC case.p3dabs* and *case.p3drel* plot files contain predicted absolute and relative reference frames flow data values, respectively, for each of the mesh points in a multiple-block mesh *ADPAC* solution. The grid-centered aerodynamic data is obtained by algebraically averaging the cell-centered data generated by the finite-volume solver during the time-marching process. As a result of the averaging procedure, this data can occasionally appear inconsistent at the edges of a mesh block, and should therefore *only* be used for graphical viewing, and not for post-processing (i.e., evaluating performance data, mass flow rates, pressure rise, etc.). The flow plot data are specified in a Cartesian coordinate system (velocities are v_x, v_y, v_z) to be consistent with the representation of the mesh file. The plot files are written in what is known as *PLOT3D* multiple grid binary format. The plot data are formatted using *SDBLIB*. The flow data are listed as nondimensional numbers using the nondimensionalization strategy described in Appendix A.

The plot files may be utilized directly with the *PLOT3D* program when the default real number size of the compiled *PLOT3D* code is defined as 32 bits (as it is on many workstations). The corresponding *PLOT3D* read command for an *ADPAC* mesh and flow file is: `read/mg/bin/x=case.mesh/q=case.p3dabs`. The user should substitute his/her own case name in the *PLOT3D* input line.

For solutions employing the one-equation Spalart-Allmaras or two-equation $k-\mathcal{R}$ turbulence model, an additional *PLOT3D* compatible file is written for plotting the turbulence parameter data. The *case.p3d1eq* or *case.p3d2eq* file is identical in format to that given above except that the variables are replaced with the turbulence parameters. For the one-equation model the variables included are $\rho, \rho\tilde{\nu}, d, \chi,$ and μ_t . For the

two-equation model the variables included are ρ , ρk , $\rho \mathcal{R}$, μ_{lam} , and μ_t . The user must be cautioned to avoid using this file in conjunction with *PLOT3D* functions which require specification of all the velocity components (i.e., pressure, temperature).

In order to understand the *PLOT3D* multiple-grid flow file format, and the utilization of the *SDBLIB* routines, a comparison of the FORTRAN coding for each method is given below for comparison. The equivalent FORTRAN coding to write an unformatted *PLOT3D* flow file could be given as:

```

WRITE(IFLOW) MG
WRITE(IFLOW) (IL(L), JL(L), KL(L),L=1, MG)
DO 10 L = 1, MG
  WRITE(IFLOW) EM(L), REY(L), ALF(L), TIME(L)
  WRITE(IFLOW) ((R(I,J,K,L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L)),
.             ((RU(I,J,K,L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L)),
.             ((RV(I,J,K,L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L)),
.             ((RW(I,J,K,L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L)),
.             ((RE(I,J,K,L), I=1, IL(L)), J=1, JL(L)), K=1, KL(L))
10 CONTINUE

```

Each of the terms used in the FORTRAN code given above are defined below:

MG	number of grid blocks
IL(L)	maximum <i>i</i> grid index for block L
JL(L)	maximum <i>j</i> grid index for block L
KL(L)	maximum <i>k</i> grid index for block L
X(I, J, K, L)	Cartesian coordinate value of x for point (I, J, K) in block L
Y(I, J, K, L)	Cartesian coordinate value of y for point (I, J, K) in block L
Z(I, J, K, L)	Cartesian coordinate value of z for point (I, J, K) in block L
EM(L)	<i>PLOT3D</i> Reference Mach number for block L
REY(L)	<i>PLOT3D</i> Reference Reynolds number for block L
ALF(L)	<i>PLOT3D</i> Reference angle for block L
TIME(L)	<i>PLOT3D</i> Reference time for block L
R(I, J, K, L)	ρ at point (I, J, K) in block L
RU(I, J, K, L)	ρu_x at point (I, J, K) in block L
RV(I, J, K, L)	ρu_y at point (I, J, K) in block L
RW(I, J, K, L)	ρu_z at point (I, J, K) in block L
RE(I, J, K, L)	ρe at point (I, J, K) in block L

An example of the corresponding FORTRAN coding to write an *ADPAC* binary flow file using the *SDBLIB* routines is given below:

```

CALL QDOPEN( IFLOW, FNAME, JE )
CALL QDPUTI( IFLOW, MG, JE )
DO 10 L = 1, MG
  CALL QDPUTI( IFLOW, IL(L), JE )
  CALL QDPUTI( IFLOW, JL(L), JE )

```

```

        CALL QDPUTI( IFLOW, KL(L), JE )
10    CONTINUE
        IPOINT = 1
        DO 20 L = 1, MG
            ILENGTH = IL(L) * JL(L) * KL(L)
            CALL QDPUTE( IFLOW, EM(L) , JE )
            CALL QDPUTE( IFLOW, REY(L) , JE )
            CALL QDPUTE( IFLOW, ALF(L) , JE )
            CALL QDPUTE( IFLOW, TIME(L), JE )
            CALL QDPUEA( IFLOW, R (IPOINT), ILENGTH, JE )
            CALL QDPUEA( IFLOW, RU(IPOINT), ILENGTH, JE )
            CALL QDPUEA( IFLOW, RV(IPOINT), ILENGTH, JE )
            CALL QDPUEA( IFLOW, RW(IPOINT), ILENGTH, JE )
            CALL QDPUEA( IFLOW, RE(IPOINT), ILENGTH, JE )
            IPOINT = IPOINT + ILENGTH
20    CONTINUE
        CALL QDCLOS( IFLOW, JE )

```

A listing of the additional terms used in the coding above is given below:

```

QDOPEN  SDBLIB routine to open a file for input or output
QDPUTI  SDBLIB routine to write an integer
QDPUTE  SDBLIB routine to write a real number
QDPUEA  SDBLIB routine to write a real array of length ILENGTH
QDCLOS  SDBLIB routine to close a file
IFLOW   FORTRAN logical unit number for flow input
        JE   An error trigger; 0 for no error,  $\neq 0$  if an error occurs
ILENGTH Integer length of an array of data
IPOINT  Integer pointer for block L to locate the initial memory
        location for a block of data

```

3.12 Restart File Description

The *ADPAC* restart file is a data file containing the cell-centered flow variables generated during an *ADPAC* solution. This file is intended to permit continued execution of the code from the point at which a previous calculation was terminated. This feature permits breaking large jobs into smaller computational pieces. This process of job restarting is considered a good practice to avoid loss of data due to computer malfunctions and job quota limitations. At the end of a given job, whether the calculation is a restart run or not, the *ADPAC* program will attempt to write out the current cell-centered data to the file *case.restart.new*. The restart file may then be used to continue the calculation at this same point by simply renaming the file *case.restart.new* to *case.restart.old*, setting the input trigger appropriately (see **FREST**), and rerunning the code. The restart data are written in either the cylindrical or Cartesian coordinate system depending on the variable format used during execution of the *ADPAC* code for each particular mesh block. Velocities are specified as either v_x, v_y, v_z (Cartesian) or v_x, v_r, v_θ (cylindrical), and all flow variables utilize the nondimensionalization strategy described in Appendix A.

Similar to the mesh file, the restart data are read as a single-dimensional array in the *SDBLIB* format, and the *ADPAC* program includes a conversion routine (source file *convass.f*) which converts the one-dimensional array data to three-dimensional array data.

For solutions employing the iterative implicit time-marching algorithm, several time levels of data must be stored in the restart file to properly restart the solution. The additional time levels of data are stored immediately following the current time level data and a simple trigger variable (*IDATA*, above) which informs the code of the existence of the additional time level data. If no additional data are present in the restart file, and an implicit solution is being restarted, the code initializes the additional time level data arrays with the best available values.

In order to demonstrate the format of the restart file, a sample of the FORTRAN coding utilizing the *SDBLIB* library required to read a restart file is given below:

```
CALL QDOPEN( IREST, FNAME, JE )
CALL QDGETI( IREST, MG    , JE )
DO 10 N = 1, MG
    CALL QDGETI( IREST, IMX(N), JE )
    CALL QDGETI( IREST, JMX(N), JE )
    CALL QDGETI( IREST, KMX(N), JE )
10  CONTINUE
DO 20 N = 1, MG
    LENGTH = IMX(N) * JMX(N) * KMX(N)
    CALL QDGEEA( IREST, R (IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RU(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RV(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RW(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RE(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, P (IJK(N)), LENGTH, JE )
20  CONTINUE
NLENGTH = MG
CALL QDGEIA( IREST, NCYC    , NLENGTH  , JE )
CALL QDGEEA( IREST, DTHETA  , NLENGTH  , JE )
CALL QDGEEA( IREST, OMEGAL  , NLENGTH  , JE )
```

Additional data for implicit calculations follows.

```
CALL QDGETI( IREST, IDATA, JE )
DO 30 N = 1, MG
    LENGTH = IMX(N) * JMX(N) * KMX(N)
    CALL QDGEEA( IREST, RM1 (IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RUM1(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RVM1(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RWM1(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, REM1(IJK(N)), LENGTH, JE )
    CALL QDGEEA( IREST, RM2 (IJK(N)), LENGTH, JE )
```

```

CALL QDGEEA( IREST, RUM2(IJK(N)), LENGTH, JE )
CALL QDGEEA( IREST, RVM2(IJK(N)), LENGTH, JE )
CALL QDGEEA( IREST, RWM2(IJK(N)), LENGTH, JE )
CALL QDGEEA( IREST, REM2(IJK(N)), LENGTH, JE )
30  CONTINUE
CALL QDCLOS( IREST, JE )

```

Each of the terms used in the FORTRAN code given above are defined below:

```

MG      Number of grid blocks
IMX(L)  Maximum i grid index for block L
JMX(L)  Maximum j grid index for block L
KMX(L)  Maximum k grid index for block L
R(IJK(L))   $\rho$  at point IJK(L) in block L
RU(IJK(L))  $\rho u_x$  at point IJK(L) in block L
RV(IJK(L))  $\rho u_y$  at point IJK(L) in block L
RW(IJK(L))  $\rho u_z$  at point IJK(L) in block L
RE(IJK(L))  $\rho e$  at point IJK(L) in block L
P(IJK(L)) pressure at point IJK(L) in block L
QDOPEN  SDBLIB routine to open a file for input or output
QDGETI  SDBLIB routine to get an integer
QDGEIA  SDBLIB routine to get an integer array of length ILENGTH
QDGEEA  SDBLIB routine to get a real array of length ILENGTH
QDCLOS  SDBLIB routine to close a file
IREST   FORTRAN logical unit number for restart input
JE      Integer error trigger; 0 for no error,  $\neq 0$  if an error occurs
ILENGTH Integer length of an array of data
IJK(L)  Integer pointer for block L to locate the initial memory location
        for a block of data
NCYC    Iteration number when restart file was written
DTHETA  Block rotation increment ( $\Delta\theta$ )
OMEGAL  Block rotational speed (nondimensional)
IDATA   Implicit restart data trigger (0-no implicit data, 1-restart data
        follows)
RM1(IJK(L))  $\rho$  at point IJK(L) in block L ( $n - 1$  time level)
RUM1(IJK(L))  $\rho u_x$  at point IJK(L) in block L ( $n - 1$  time level)
RVM1(IJK(L))  $\rho u_y$  at point IJK(L) in block L ( $n - 1$  time level)
RWM1(IJK(L))  $\rho u_z$  at point IJK(L) in block L ( $n - 1$  time level)
REM1(IJK(L))  $\rho e$  at point IJK(L) in block L ( $n - 1$  time level)
RM2(IJK(L))  $\rho$  at point IJK(L) in block L ( $n - 2$  time level)
RUM2(IJK(L))  $\rho u_x$  at point IJK(L) in block L ( $n - 2$  time level)
RVM2(IJK(L))  $\rho u_y$  at point IJK(L) in block L ( $n - 2$  time level)
RWM2(IJK(L))  $\rho u_z$  at point IJK(L) in block L ( $n - 2$  time level)
REM2(IJK(L))  $\rho e$  at point IJK(L) in block L ( $n - 2$  time level)

```

CYCLE NUMBER	MAXIMUM ERROR	RMS ERROR	MASS INFLOW	MASS OUTFLOW	PRESSURE RATIO	ADIABATIC EFFICIENCY	NUMBER SS	NUMBER PTS	NUMBER SEPPTS
301	-2.17895	-5.52081	3.46778	3.46781	1.58615	0.88737	308	0	
302	-2.42051	-5.57462	3.46777	3.46656	1.58762	0.89150	315	0	
303	-2.65891	-5.64842	3.46774	3.46646	1.58949	0.89685	317	0	
304	-2.78033	-5.71740	3.46765	3.46781	1.59123	0.90145	320	0	
.	
.	
.	

Figure 3.16: Sample *ADPAC* convergence file (*case.convergence*).

3.13 Convergence File Description

The *ADPAC* convergence history file *case.converge* is an ASCII data file which contains the residual convergence history of the time-marching solution. The residual history is useful for determining whether the numerical solution has converged sufficiently to permit interrogation of the numerical results, or whether additional restarted calculations are required to obtain an accurate solution. Typically, a solution is deemed converged when the residuals have been reduced by three orders of magnitude or more. The *case.converge* file is formatted in columns to permit convenient plotting using any of a number of plotting programs and is organized as shown in Figure 3.16.

The residual at any cell in the finite volume solution is calculated as the sum of the changes in the five conservation variables (ρ , ρu , ρv , ρw , and ρe). The maximum residual is then defined as the maximum of all the residuals over all the cells of all mesh blocks. The root-mean square residual is the square root of the sum of the squares of all the cells for all mesh blocks. The *case.converge* file residual data are reported as the base 10 logarithm of the actual residuals in order to quickly evaluate the convergence of the solution (e.g., if the reported \log_{10} maximum residual starts at -2.5 and ends up at -5.5, the solution has converged three orders of magnitude). Several additional data are also output to the convergence file based on flow parameters occurring across inflow and outflow boundaries. Since many flow cases involve a single inlet and a single exit, a useful measure of convergence is the difference between the inlet and exit mass flow rate, and how much the mass flow rate varies from iteration to iteration.

For 2-D Cartesian flow calculations a unit depth (1.0 in mesh coordinates) is assumed for the third coordinate direction to determine the mass flow rate. For 2-D cylindrical flow calculations, the geometry is assumed to be axisymmetric (in the $x-r$ plane) and a multiple of 2π is used in the mass flow integration (the mass flow is computed as if the full circumference of the axisymmetric geometry were employed). Data are provided in the convergence file for the sum of all mass crossing any inflow boundary (**INLETG**, **INLETX**, **INLETT**, **INLETM**, **INLETR**) and all mass crossing any exit boundary (**EXITG**, **EXITX**, **EXITT**, **EXITP**). The pressure ratio (ratio of mass-averaged total pressures from all inlet and exit boundaries) is also reported in the convergence file, as well as the adiabatic efficiency computed based on mass-averaged total temperature and total pressure of the inflow and outflow boundaries, respectively. Finally, the number

of computational cells with supersonic flow and the number of computational cells indicating negative axial flow ($v_x < 0$) are also reported in the convergence history file. The appearance of negative axial flow may or may not be a concern depending upon the geometry being modeled; however, this value should level off to a near-constant value for converged solutions. Oscillation of this value may indicate a shedding tendency within the flow domain which may be triggered either by physical or non-physical (numerical) means.

In the event that the one-equation or two-equation turbulence model is enabled (**F1EQ=1.0** or **F2EQ=1.0**), the convergence characteristics of the turbulence transport equation(s) are output into a separate file named *case.converge_SA* or *case.converge_KR*, respectively. This allows the user to also monitor the turbulence transport equation(s) convergence characteristics.

3.14 Image File Description

The *ADPAC* graphics display system (see Chapter 5) has the capability of saving a raster image of the local graphics screen to a file at specific iteration intervals using the Silicon Graphics image file format. This feature is included as a simple means of constructing flowfield animations (see **FGRAFIX**, **FGRAFIN**, **FIMGSAV**, and **FIMGINT**). Image files can be viewed after they have been saved by issuing the command: `ipaste case.img.#`, or converted to other image formats (i.e., TIFF, JPG, GIF). Other Silicon Graphics IRIX Operating System-specific commands such as `imgview`, `movie`, and others may also be suitable for viewing image files. Additional information on the IRIS image format and the image manipulation commands are available in the Silicon Graphics system documentation.

3.15 Running ADPAC With The One-Equation Turbulence Model

In order to run *ADPAC* with the one-equation Spalart-Allmaras turbulence model enabled, the following steps must be taken. First, the input file trigger must be enabled (**F2EQ=1.0**). This activates the additional partial differential equation solution scheme for the one-equation model. Note that this completely disables the standard algebraic (Baldwin-Lomax) turbulence model. It is generally recommended that the mesh should be sufficiently refined to adequately resolve the inner (laminar sublayer) of the turbulent boundary layer flow. There is, at present, no built in mechanism in the *ADPAC* code to verify that this mesh restriction has been met, and it is therefore up to the user to perform this check.

The specification of inlet boundary conditions and initial conditions for the turbulence model transport variable ($\tilde{\nu}$) is handled by specifying a value of the non-dimensional variable $\chi = \tilde{\nu}/\nu$. By specifying χ , the user does not need to account for variations in $\tilde{\nu}$ caused by changes in **PREF**, **TREF**, **DIAM**, or any other reference quantity used for non-dimensionalization. It was found in the cases tested, that a small initial value of χ does not provide a strong enough trigger for the production term and causes the solution to converge to the trivial solution ($\tilde{\nu} = 0.0$), resulting in a laminar flow field. The boundary data file modifications apply only to inflow boundary specifications (i.e.,

INLETT, INLETG). The reader is referred to the section on boundary conditions for details covering the specification of inlet flow χ values. Most cases are run using an initial value of χ equal to 20 (**FKINF**=20.0 set in *case.input*) with inlet values being specified to one ($\chi_{in} = 1.0$ set in the *case.boundata*).

The behavior of the partial differential equation can be monitored in both the standard output and convergence file iteration histories of maximum and RMS residual. Separate convergence history lines are tabulated by the *ADPAC* code for the $\rho\tilde{\nu}$ transport equation. This data is printed to the standard output immediately following the continuity, momentum, and energy equation residual information at each iteration. In order to plot the convergence history, a separate convergence file is created with only the one-equation residual data (*case.converge_SA*).

Finally, upon completion, a *PLOT3D* based data file representing the mesh point-averaged data ρ , $\rho\tilde{\nu}$, d , χ , and μ_t is written to the file *case.p3d1eq*. This file, in conjunction with the *ADPAC* mesh file (*case.mesh*) may be employed to graphically examine the predicted turbulence characteristics of the flowfield.

Additional details of the one-equation Spalart-Allmaras turbulence model and its implementation into *ADPAC* are included in Appendix A.

3.16 Running ADPAC With Two-Equation Turbulence Model

In order to run *ADPAC* with the two-equation k - \mathcal{R} turbulence model enabled, the following steps must be taken. First, the input file trigger must be enabled (**F2EQ**=1.0). This activates the additional partial differential equation solution scheme for the k - \mathcal{R} model. Note that this completely disables the standard algebraic (Baldwin-Lomax) turbulence model and the wall function formulation. The lack of wall functions implies that the mesh *must* be sufficiently refined to adequately resolve the inner (laminar sublayer) of the turbulent boundary layer flow. There is, at present, no built in mechanism in the *ADPAC* code to verify that this restriction has been met, and it is therefore up to the user to perform this check. In addition to the modified input file, the boundary data file must also be modified slightly. The boundary data file modifications apply only to inflow boundary specifications, specifically the boundary descriptor **INLETG**. At present, only the **INLETG** specification may be used to properly define an inflow boundary in the k - \mathcal{R} solution. Since the k - \mathcal{R} turbulence model is based on transport equations, it is necessary to properly specify the inflow values of k and \mathcal{R} much in the same manner that other inflow properties (total pressure, etc) must be specified. The inflow values for k and \mathcal{R} are specified on the 2 lines following the **INLETG** specification as shown in the example below:

```
INLETG      1   1  I  I  P  P  J  K   1   1   1  73   1   2   1  73   1   2
PTOT      TTOT   AKIN   ARIN
1.0      1.0    0.0001 0.001
```

Here, the extra variables labeled as **PTOT**, **TTOT**, **AKIN**, **ARIN** are the inlet non-dimensional total pressure, total temperature, turbulent kinetic energy (k), and

turbulent Reynolds number (\mathcal{R}), respectively. Note that **AKIN** and **ARIN** are also input as non-dimensional variables using the non-dimensionalization scheme previously described. Typical values of **AKIN** and **ARIN** are 0.0001 and 0.001, respectively. More accurate values for specific cases where detailed inflow turbulence characteristics are known can be determined based on the definitions of k and \mathcal{R} shown in Appendix A.

The modifications described above are all that is necessary to initiate the two-equation turbulence model solution sequence. Unfortunately, only a limited amount of experience with this turbulence model is available to guide the user in case a problem arises. The behavior of the partial differential equations can be monitored in both the standard output and convergence file (*case.converge_KR*) iteration histories of maximum and RMS residual. Separate convergence history lines are tabulated by the *ADPAC* code for the k and \mathcal{R} transport equations. This data is printed immediately following the continuity, momentum, and energy equation residual information at each iteration. It should be mentioned that the multi-grid solution strategy employed to solve the continuity, momentum, and energy equations is not completely enabled for the k - \mathcal{R} transport equations. This is an area of algorithmical research and may be completed in future releases of the *ADPAC* code. The “full” multi-grid solution initialization sequence is available to rapidly initiate the k - \mathcal{R} equation solution variables.

When employing the k - \mathcal{R} solution scheme the best practice found to date is to run the code with relatively small values of the input variable **CFL** (**CFL** = 3.0) and run large numbers of iterations. The full multi-grid start-up procedure has been found to be somewhat helpful (**FFULMG**=1.0). The k - \mathcal{R} solution scheme converges relatively slowly, and may take 2-3 times the number of iterations as the algebraic turbulence model to completely converge. In addition, the cost of a given iteration when the k - \mathcal{R} turbulence model is enabled may be up to 40% greater than a corresponding iteration using the algebraic turbulence model. It should be noted that there is no added numerical dissipation in the k - \mathcal{R} solution scheme. A naturally dissipative first order upwind differencing is used in the discretization of the convective fluxes in the k - \mathcal{R} equations. This implies that the input variables **VIS2** and **VIS4** have no effect on the two-equation turbulence model solution sequence. Finally, upon completion, a *PLOT3D* -based data file representing the mesh point-averaged data ρ , ρk , $\rho \mathcal{R}$, and μ_t is written to the file *case.p3d2eq*. This file, in conjunction with the *ADPAC* mesh file (*case.mesh*) may be employed to graphically examine the predicted turbulence characteristics of the flowfield.

In the event that the k - \mathcal{R} solution sequence diverges, or simply does not converge, the best way to stabilize the solution is to lower the value of **CFL**. Unfortunately, this also decreases the convergence rate of the solution and increases the overall CPU time required for a given run. Limited experience with this model, and interruptions in the *ADPAC* development schedule prevented a more thorough implementation of this promising model. Additional details of the two-equation k - \mathcal{R} turbulence model and its implementation into *ADPAC* are included in Appendix A.

3.17 Troubleshooting an ADPAC Failure

The *ADPAC* code contains a large number of error checking and handling facilities to determine and report to the user when a problem in the calculation occurs. Unfortunately,

some problems simply cannot be detected and it may occur that for a particular case the solution will diverge (uncontrolled increase in solution residual) or simply “blow up” as a result of numerical difficulties or an invalid numerical operation (i.e., divide by zero). These cases are notoriously frustrating for the user because the cause is often difficult to identify. The steps outlined below attempt to provide a structured approach to rectifying numerical problems for an *ADPAC* run based on the author’s experience.

Step 1. Carefully Check the ADPAC Input File for Errors

The *ADPAC* standard input file controls the overall characteristics of the computational process and plays a large role in determining the behavior of a job. The user should check the output file for the correct interpretation of the input file variables and possible unspecified variable defaults being used. Typical parameters to check are to make sure that the **CFL** variable is negative for steady-state calculations and positive for time-accurate calculations, and to make sure that the absolute value is not too large (5.0 is a typical magnitude). If the **CFL** value is greater than the **CFMAX** variable, or generally if the magnitude of **CFL** is larger than 2.5, then residual smoothing *must* be activated (**FRESID**=1.0). Naturally, the values for **VIS2** and **VIS4** should also be within their suggested limits. A common problem for rotating geometries is an incorrect rotational speed, or simply the wrong sign on the rotational speed (rotating the wrong way), so check the values of **RPM** and/or **ADVR** carefully. The user can also selectively turn off features such as the turbulence model (see **FTURBB**) and/or multi-grid (see **FMULTI**) to check on their influence on the stability of the solution. Finally, the user should make sure that the proper **CASENAME** and **DIAM** variables are specified in the input file. Other problems are discussed in the individual input file variable descriptions in Section 3.6.

Step 2. Carefully Check the ADPAC Boundary Data File for Errors

The *ADPAC* boundary data file controls the application of boundary conditions on the various mesh surfaces necessary to define the flow characteristics of an *ADPAC* run. Common errors in the boundary data file include mismatched **PATCH** specifications, incorrectly specifying inflow data (particularly when **INLETT** is used), and incorrectly specifying rotational speeds for solid surfaces using **SSVI**. If the solution will run for a few iterations, it may be helpful to get a *PLOT3D* output file at this point and examine the solution (using *PLOT3D*, *FAST*, *TECPLOT*, *Fieldview*, etc.). Check for obvious solution features such as flow going the right direction, flow that does not penetrate a solid boundary, contour lines matching at **PATCH** boundaries (although contour lines may not match exactly at any mesh edge), and obvious radical changes in flow variables (total pressures and/or total temperatures which are very large or negative). The user can often trace a faulty boundary condition by selectively commenting out several specifications from the boundary data file and rerunning to see if the same problem occurs. If the solution diverges even when no boundary conditions are specified (place **ENDDATA** at top of *case.boundata*), then a problem exists in the mesh or input file. Other boundary condition specific common errors are discussed in the individual boundary data file variable descriptions in Section 3.7.

Step 3. Carefully Check the ADPAC Mesh File for Errors

The user should verify the existence of the mesh file with the proper name and

non-zero size in the current working directory. The user can next check to make sure the file can be read with the *PLOT3D* [14] graphical plotting program. A mesh file which has been created using the *PLOT3D* binary file write option may *not* be acceptable due to the format of the Scientific Database Library (although it can be made so by appending 1024 bytes of any data to the end of the mesh file). The *MAKEADGRID* program is available to convert unformatted mesh files to *ADPAC* compatible mesh files.

Most common problems encountered when the *ADPAC* code does not perform adequately can be traced to poor mesh quality. Although the mesh may be free from obvious flaws such as crossed mesh lines and/or zero volumes, this does not guarantee that numerical difficulties will be avoided. The most common overlooked features of mesh quality are the mesh expansion ratio and the mesh shear angle. Mesh expansion ratio relates to the change in physical mesh spacing along a given coordinate direction from one point to another. For stability, the maximum mesh expansion ratio at any point should not exceed 1.3 on the fine mesh and not exceed 2.0 on the coarsest grid level when using multi-grid acceleration. The *ADPAC* code provides a listing of maximum mesh expansion ratios for each grid block and issues a warning if the mesh expansion ratio exceeds 1.3. The code can tolerate larger ratios in many cases, but definite problems can be expected if the maximum expansion ratio gets larger than 2.0. Mesh shear can also cause problems; the more orthogonal the mesh, the less likely mesh-induced numerical difficulties will occur. Another potential mesh problem involves mesh cells with very small radii (i.e., along a sting upstream of a propeller) which may require increasing the diameter of the sting to prevent problems. Application of the multi-grid iteration strategy and reducing the value of **EPSY** in the input file have been found to be effective remedies for such problems.

Step 4. Check for the Possibility of an Invalid Flow Condition

The author's experience has been that many users feel if a problem can be defined then it should possess a solution; in fluid dynamics this is certainly not true. If a solution is attempted for a fan rotor, for example at a pressure ratio which is beyond the stall point for that rotor, then no solution exists and the code will very likely diverge without explanation. In many cases, the numerical equivalent of an invalid flow condition is that the solution will either not converge, or will simply diverge. Another common example is attempting to extract a steady-state solution for a problem which is truly time-dependent. Blunt body flows often result in a time-dependent solution due to vortex shedding, and the steady-state analysis of this flow will likely never converge. This behavior also occurs frequently when a strong adverse pressure gradient or flow separation is present in the solution. Now it is true that in some cases, the level of convergence may also be limited by such factors as mesh quality, numerical accuracy, and/or turbulence model limit cycles, and it is difficult to determine whether the cause is numerical or physical. This is unfortunately a matter of experience and the user is encouraged to question whether their case can truly have a "steady" solution.

Step 5. Determine if the Problem is Computer Dependent

The *ADPAC* code was developed and tested on UNIX-based operating systems using FORTRAN 77 standard coding techniques. In spite of the standardization in the computer industry, not all machines produce the same answer for a given problem due to compiler optimizations and code handling features. It has been the author's experience

that when the results are unexpected or spurious, compilers are often a source of problems, particularly when the code has been compiled for the first time on a specific architecture, or when a new release of the operating system or FORTRAN compiler has been installed. Before reporting an unsolvable problem, it is a good practice to completely recompile the code on a known stable machine with a well-tested version of FORTRAN *without* using optimization (modifications to the *ADPAC* Makefile may be required). If the code displays the same error, then it is possible that a bug has been uncovered and this should be reported so future versions do not encounter the same problem. If the compiler supports static memory allocation, then this option should be enabled whenever possible.

Step 6. Determine the Effect of Key Input Variables

Some “fine-tuning” of input variables is occasionally required to obtain a converged solution, or to prevent an instability from forming. The following suggestions may be useful to aid in establishing the sensitivity of the solution to various inputs:

- A. Clear the input file and boundary data file of all specifications (except the case name, which must be activated). If the code diverges, there is almost certainly a problem with the grid. Examine the code output to determine where the maximum error occurs, and carefully check the grid in this region.
- B. Try to run the problem for a few cycles without any boundary conditions and **CFL** = -1.0. This is essentially a uniform flow test. If the code diverges, then the problem is either in the input file or the mesh.
- C. Turn off all multi-grid (**FMULTI** = 1.0, **FFULMG** = 0.0). If it now runs, check the mesh expansion ratios.
- D. Vary the parameter **CFMAX**. Lower values imply more smoothing. It is possible to have too much smoothing, so both larger and smaller values should be tested.
- E. Make sure **FRESID** is set to 1.0 if the magnitude of **CFL** is larger than 2.0.
- F. Examine and vary the values of **VIS2** and **VIS4**.
- G. Turn off the turbulence model **FTURBB** = 999999.0. If the problem still exists, try to run inviscid flow (**FINVVI** = 0.0).

Step 7. Report the Problem

In the event that no other cause of the problem can be detected, the problem should be reported to NASA. The recommended contact for problems (or successes) is:

Dr. Chris Miller
Mail Stop 77-6
NASA-Lewis Research Center
21000 Brookpark Road
Cleveland, OH 44135
(216) 433-6179
Christopher.J.Miller@lerc.nasa.gov

The author is also interested in keeping up with known problems and may be reached at:

Dr. Ed Hall
Speed Code T-14A
Allison Engine Company
Indianapolis, IN 46206-0420
(317) 230-3722
Edward.J.Hall@allison.com

Chapter 4

RUNNING ADPAC IN PARALLEL

4.1 Description of Parallel Solution Sequence

The *ADPAC* code possesses programming features which permit execution of multiple-block jobs across multiple-processor computing architectures. By definition, multiple-processor computing architectures includes both shared-memory multiprocessor computers such as the Silicon Graphics ORIGIN 2000, and network-connected UNIX workstation clusters acting as a virtual parallel computing resource, as shown in Figures 4.1 and 4.2, respectively. The parallelization philosophy in *ADPAC* is based on a coarse-grained domain decomposition. Individual blocks in a multiple-block mesh are distributed among the available processors to effectively spread the computational load. During parallel execution, the *ADPAC* code employs a master/slave programming style for parallel jobs. This implies that all input and output data processing occurs through a master, or *root*, process as shown in Figure 4.3.

The *ADPAC* source code invokes parallelization based on inter-processor message passing via native calls to the Application Portable Parallel Library (*APPL*) message passing library. This programming structure is illustrated in Figure 4.4. *APPL* was developed at the NASA Lewis Research Center and documentation is available that explains how to write code using *APPL* and how to run codes written with *APPL* [19]. Previous versions of this manual [20] provided specific instructions on how to run *ADPAC* using the direct *APPL* parallel interface.

While *APPL* by itself is a powerful programming medium, it does not provide widespread support for many computer systems, nor does it necessarily provide optimum communication performance (although in the authors' experience the *APPL* routines are better than many other alternatives). To exploit other options for parallel computing *ADPAC* relies on programming layers which translate the native *APPL* calls to the desired inter-processor communication language. Programming layers are available for the Parallel Virtual Machine (*PVM*) [21] and Message Passing Interface (*MPI*) [22] communication libraries. The modified programming structure for the *PVM* and *MPI* communication interfaces are illustrated in Figures 4.5 and 4.6, respectively. As such, *ADPAC* can be run in parallel using native *APPL*, or *PVM*, *MPICH*, *CHIMP*, or other proprietary *MPI* communication implementations. Based on the authors' overall experience with

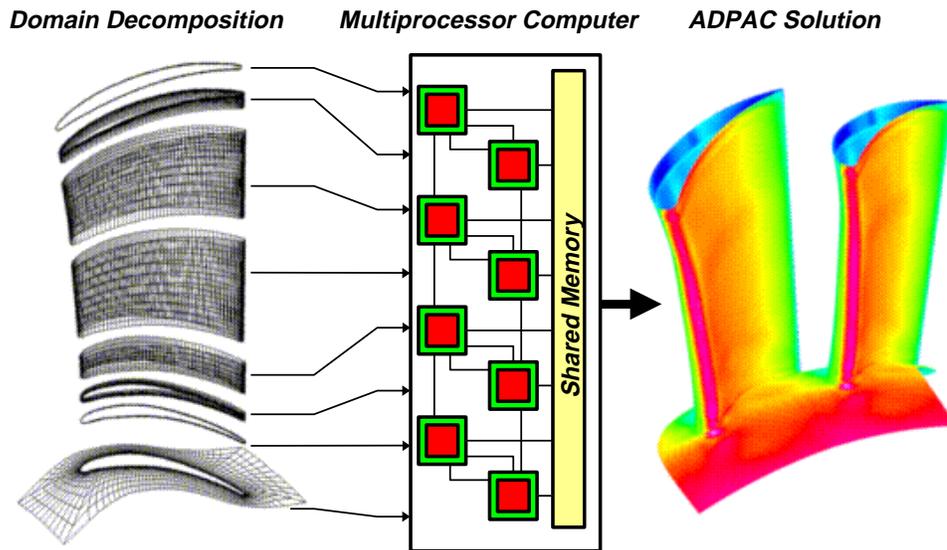


Figure 4.1: Illustration of domain decomposition parallel computing using the *ADPAC* code on a multiprocessor computing architecture.

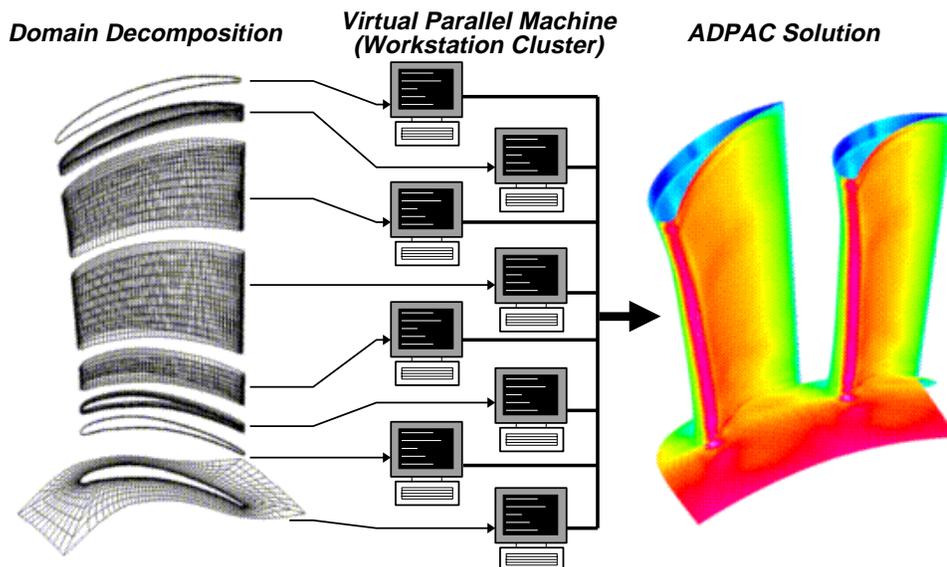


Figure 4.2: Illustration of domain decomposition parallel computing using the *ADPAC* code on a network-connected workstation cluster computing architecture.

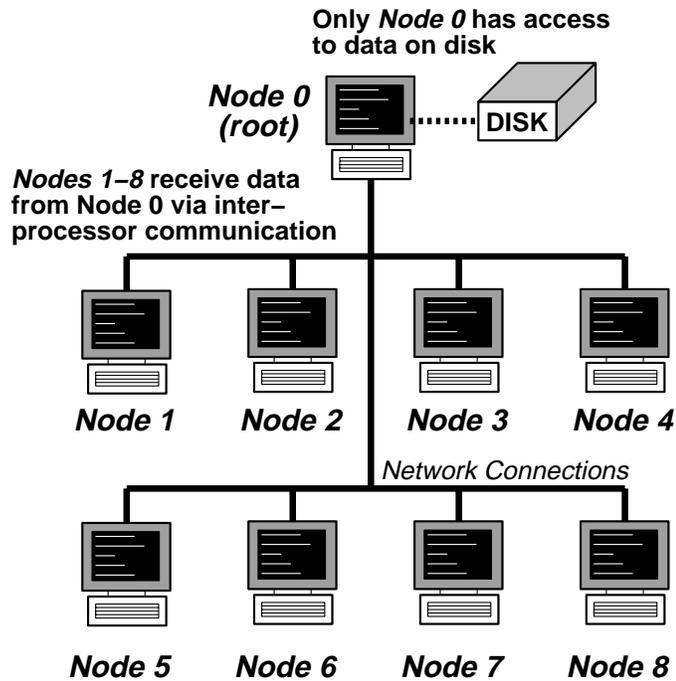


Figure 4.3: Illustration of *ADPAC* master-slave coding style and data input/output processing for parallel computing. Note that only the root process (Node 0) needs access to the data disk. The configuration shown represents a nine-node workstation cluster (nodes numbered 0-8).

ADPAC and the various communication libraries, the *MPICH* library is normally recommended due to the wide variety of platforms for which the library is available and the high communication performance achieved with this scheme. The *MPICH* code also provides an automatic configuration utility which simplifies compiling the library on different platforms. Complete details for compiling the *APPL* and *MPICH* libraries for every UNIX platform are beyond the scope of this report, and the reader is directed to the documentation provided with each library for help.

In order to employ the parallel execution features of *ADPAC* using the *MPICH* communication library, the following criteria must be satisfied:

- The user must have access to a multiprocessor computing platform such as a Silicon Graphics ORIGIN 2000 or equivalent. Virtual parallel computing platforms can also be constructed from clusters of network-connected workstations. Examples of this type of platform are given throughout this section.
- The *ADPAC* code must be compiled for parallel execution. This implies not only compilation of the *ADPAC* source code, but also successful compilation of the `applmpi` translation library, and the availability of a Message-Passing Interface (*MPI*) library. For general cases, the *MPICH* library is recommended. In many cases, computer vendors also provide *MPI* libraries which have been optimized for their specific architecture. In these cases, it may be desirable to use the proprietary library, and the vendors instructions should be consulted for proper compilation and running of the parallel code. A sample UNIX shell script illustrating the basic steps

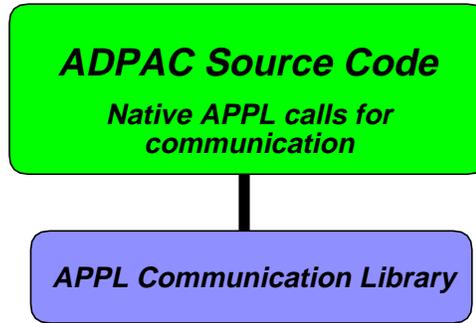


Figure 4.4: Illustration of *ADPAC* code programming structure for parallel communication using the native *APPL* interprocessor communication interface.

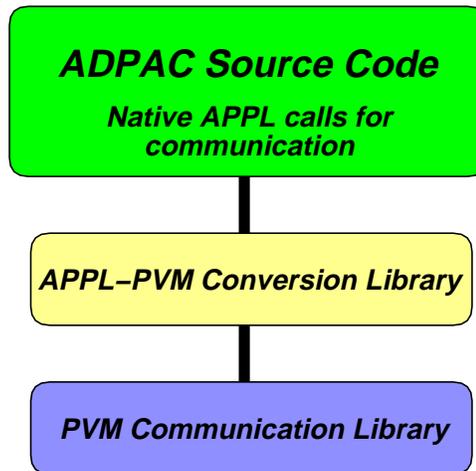


Figure 4.5: Illustration of *ADPAC* code programming structure for parallel communication using the native *APPL* procedure calls, `applpvm` translation library, and the *PVM* interprocessor communication library.

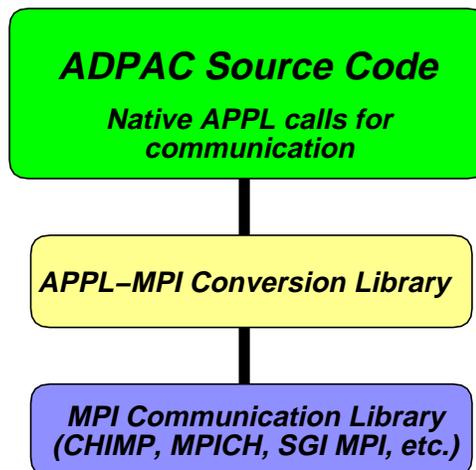


Figure 4.6: Illustration of *ADPAC* code programming structure for parallel communication using the native *APPL* procedure calls, `applmpi` translation library, and the *MPI* interprocessor communication library.

required for compiling and running the ADPAC code in serial and in parallel for both the *MPICH* and a proprietary *MPI* libraries is given in Appendix B. This script was developed for Silicon Graphics ORIGIN-class multiprocessor computers. Similar scripts for other popular UNIX-based workstations can be created easily by modifying this script for other computing platforms.

- The *ADPAC* multiple block mesh file should contain at least *NPROC* mesh blocks, where *NPROC* is the number of processors to be used in the parallel computing environment. It is possible to run more blocks than processors, in which case some processors will provide computational services for more than one block. In this case, the *ADPAC* includes a simple algorithm to distribute the blocks among the available processors. However, this technique does not necessarily provide optimal load balancing between processors. The user-definable *casename.blkproc* file is described later in this chapter to directly specify which blocks are run on a given processor. It is also possible to have more processors than blocks. In this case, one or more processors will effectively be idle and constitutes a waste of computational resources. If the user desires to run a job effectively on more processors than the current mesh block limit allows, the *SIXPAC* and *BACPAC* codes are available to allow the user to subdivide the initial problem into more blocks, thus allowing the use of additional processors. These tools are described in more detail later in this chapter.

MPI -based executables are initiated using the `mpirun` command as shown below:

```
mpirun -np numprocs adpac_executable < adpac.input > output
```

In the `mpirun` syntax above, *numprocs* is a number representing the number of processors to employ, *adpac_executable* is the name of the *ADPAC* executable file (i.e., `adpac_power_challenge_mpi`), *adpac.input* is the name of the *ADPAC* input file which must be named `adpac.input` for parallel jobs, and *output* is the name of the standard output file.

If a communications error is trapped, or if a process has died unexpectedly, the `mpirun` function shuts down all of the remaining processes gracefully. This feature is most important on workstation clusters, which have no built-in mechanism for monitoring parallel jobs. The `mpirun` script controls the execution of *ADPAC* on the various processors, and when using *MPICH*, may invoke additional input from the *MPICH*-based `mpich/util/machines/machines.ARCH` file (see the *MPICH* implementation notes for details) to determine the machine names upon which the job is to be executed. For multiprocessor machines, the machine names are all the same and `mpirun` simply assigns processes to individual processors in a single machine. For workstation clusters, the machines names will vary, and the `mpich/util/machines/machines.ARCH` file should contain a list of machine names to be used in the execution of the job. In theory, the *ADPAC* -*MPICH* system permits parallel execution on both homogeneous and heterogeneous workstation clusters. The bulk of experience to date has been for homogeneous workstation clusters. The reader is referred to the man pages and implementation notes for the *MPI* library employed for further details.

The host machines in a workstation cluster must be connected by Ethernet, but do not have to share disks or be part of the same subnet. This provides tremendous flexibility in constructing a workstation cluster. However, most performance bottlenecks encountered on workstation clusters involve the network. The benefits of adding

processors may be offset by poor network performance. The tradeoff varies with the problem and with the hardware configuration.

In general, the behavior of *ADPAC* in parallel is the same as in serial. This is especially true if there are no input errors. The output files may be different if there are input errors. There are two general types of input errors detected in *ADPAC*. Errors involving the grid or the input file will generally be detected by all processors, and the error messages will appear as they do in serial.

However, if an error is discovered in a boundary condition routine, the output messages will probably appear differently in the output file, and may not appear at all. Since *ADPAC* boundary conditions are applied in parallel, Node 0 does not execute all of them, but only those involving a block assigned to Node 0. If Node 0 does not encounter the error, then a different node writes the error message. Since the writing node is out of sync with Node 0, the error message may be written to a different place in the output file than if Node 0 had written it.

Buffering of output on the various processors can also cause a problem. Usually, after an error message is printed, execution is stopped on all processors. If execution fails before the buffer is flushed, then output may be lost from some processors. The result is that an error message could be caught in the buffer and never appear in the output file. If *ADPAC* terminates for no apparent reason, this may explain the problem. The solution is to rerun the job without redirecting the output. If output is not redirected, it is normally not buffered, and all of the output will appear at the user's terminal window.

It is also possible to get multiple copies of an error message if more than one processor encounters the error. Wherever possible, *ADPAC* has been coded to avoid these problems, but these unfortunate possibilities still exist. Therefore, running *ADPAC* interactively is the best way to track down input problems.

Aside from these considerations, running *ADPAC* in parallel is very much like running *ADPAC* in serial. The input files are identical, and the output files are very similar. The most common problems in running *ADPAC* in parallel are failing to use the `mpirun` function, improperly specifying the parallel configuration, and attempting to run a serial executable in parallel.

4.2 SIXPAC (Block Subdivision) Program

SIXPAC, which stands for **S**ubdivision and **I**nformation **eX**change for **P**arallel **ADPAC** **C**alculations, enables the user to redefine the block structure of an *ADPAC* job. Using *SIXPAC*, large grid blocks can be subdivided to improve load balance, or to make use of smaller memory processors in parallel calculations. Creating new blocks according to user specifications, *SIXPAC* generates new input, mesh, restart, and boundata files for the subdivided problem. The resulting files represent a problem equivalent to the original but with more, smaller blocks. Although the number of unique grid points is unchanged, the total number of points is larger because of duplication at the additional interfaces.

The motivation for *SIXPAC* comes from the way *ADPAC* was parallelized. Rather than parallelize the interior point solver, *ADPAC* was parallelized through the boundary

conditions. An individual block cannot be run across multiple processors; each processor must contain only whole blocks. This implies that a problem with a single large block could not be run in parallel. *SIXPAC* enables large blocks to be recast as groups of smaller blocks, so that they can be run in parallel. If a problem already contains multiple mesh blocks, *SIXPAC* is not required to run a problem in parallel, but it simplifies the process of setting up a problem for optimal parallel performance and load balancing.

4.2.1 *SIXPAC* Input

The input files required by *SIXPAC* are *case.input*, *case.mesh*, *case.boundata*, and *case.sixpac*. If a new restart file is to be created, then a *case.restart.old* file is also required. If a restart file is to be created for the subdivided problem, the input trigger **FREST** must be set equal to 1.0 in the *case.input* file. This tells *SIXPAC* to look for a *case.restart.old* file, and to subdivide it. Of this group, only the *case.sixpac* file is added to the standard *ADPAC* files previously defined.

The *case.sixpac* file contains information which specifies how the blocks are to be subdivided. The required information includes the number of original blocks, and how each block is to be subdivided in each indicial direction (i , j , and k). In each direction, the number of subdivided blocks, and possibly the locations of the subdivisions, must be specified. If the number of subdivided blocks in a particular coordinate direction is set to 1, then the block is not divided in that coordinate direction. *SIXPAC* has some sanity checks built in to warn users of problems in the *case.sixpac* file, and free format input is used so alignment is not important.

By default, blocks are split into the specified number of (as nearly) equally sized pieces. If unequal divisions are required in a particular direction, then the location of each division must be specified in that direction. Unequal divisions are often employed to preserve levels of multi-grid, or to put the edge of a geometric feature on a block boundary. Figure 4.7 illustrates how different block strategies affect multi-grid.

For equal divisions of the blocks in each direction, the *case.sixpac* is simple to construct. A sample *case.sixpac* file is listed in Figure 4.8. The first line is a comment, and the second line contains the number of blocks in the original problem. The third line is a comment, and there is an additional line for each original block, in ascending order. These lines contain the block number, and the number of subdivided blocks in each coordinate direction.

In the example *SIXPAC* input file, there are two original blocks. The first block is to be divided into 4 pieces along the i coordinate, 2 pieces along the j coordinate, and 1 piece along the k coordinate. The second block is to be divided into 4 pieces along the i coordinate, 2 pieces along the j coordinate, and 1 piece along the k coordinate. This means that there will be a total of 16 new blocks generated from the original 2 blocks.

If user-specified divisions are required in a direction, the *case.sixpac* file must be modified following the rules below, and as shown in Figure 4.9

- The number of subdivided blocks in the direction to be specified is set to 0. This tells *SIXPAC* that user specifications are to follow.

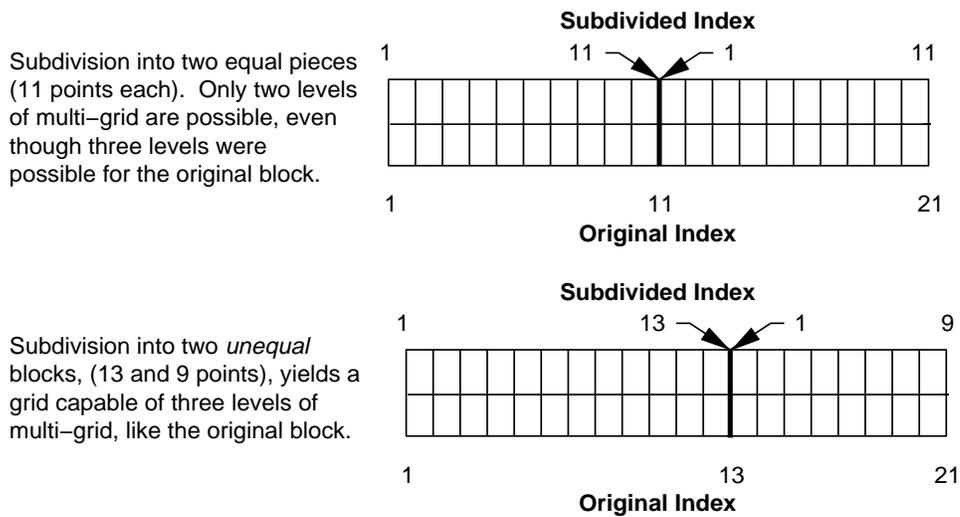


Figure 4.7: Careful block division can preserve levels of multi-grid.

```

Number of blocks
2
n  idiv  jdiv  kdiv
1   4    2    1
2   4    2    1

```

Figure 4.8: Sample input file for *SIXPAC* block division utility.

```

Number of blocks
2
n   idiv jdiv kdiv
1   4    0    1
    number of J divides
    2
    J break points
    3 10
2   4    2    1

```

Figure 4.9: Sample input file employing user-specified block divisions for *SIXPAC* block division utility.

- New lines are added to the *case.sixpac* immediately following the block to be modified. First, a comment line is added, which normally identifies which direction is being specified. Second, a line containing the number of subdivided blocks is specified. Third, a comment line is added, which normally indicates that the following line contains block division points. Fourth, lines are added containing the division positions for the new blocks.
- The block division positions are the upper limits of the new blocks in terms of the original block indices. The last division should be the block size in that direction.
- Block division positions must be specified in ascending order.
- If user-specified subdivisions are required in more than one direction of a single block, then the additions are made in “natural order,” that is *i* first, then *j* and *k*, as required.
- All blocks must appear in the *case.sixpac* file in ascending order.

In this modified example, there are again two original blocks. The first is to have 4 *i* divisions, 2 *j* divisions and 1 *k* division. The *j* divisions are to be at $j = 3$ and $j = 10$ in block 1. The second block is to have 4 *i* divisions 2 *j* division and 1 *k* division. This means that there will be a total of 16 new blocks generated from the original 2 blocks.

4.2.2 *SIXPAC* Output

The output files produced by *SIXPAC* are *Ncase.input*, *Ncase.mesh*, *Ncase.boundata*, and *Ncase.bacpac*. If a restart file is read in, a *Ncase.restart.old* file is written, and the new input file will be set up to run with the new restart file. The casename has been prepended by an “N” to avoid confusion with the original input files. The newly created files are *ADPAC* input files which can be run in either serial or parallel versions of *ADPAC*.

The *Ncase.bacpac* file is not required to run *ADPAC*, but is used by the code *BACPAC* to reassemble the blocks into their original, undivided structure. The *Ncase.bacpac* file contains information about the way *SIXPAC* subdivided the blocks. There is normally no reason for the user to alter the *Ncase.bacpac* file. The form of the *Ncase.bacpac* file is described below in the section describing *BACPAC*.

Running *SIXPAC* is very much like running *ADPAC*. The command syntax is:

```
sixpac < case.input > output
```

The *output* file is similar to an *ADPAC* output file, because the routines to read the grid, the input file and the boundary data file are the same as in *ADPAC*. One addition to the output file is a table of the new grid blocks and their sizes. After verifying the new block structure created by *SIXPAC*, the output file can be discarded.

4.3 BACPAC

BACPAC, which stands for **B**lock **A**ccumulation and **C**onsolidation for **P**arallel **ADPAC** **C**alculations, reassembles subdivided *ADPAC* files into their original, undivided form. It is used in conjunction with *SIXPAC*, and performs essentially the inverse operation of *SIXPAC*. *BACPAC* can reconstruct mesh, *PLOT3D*, or restart files, producing new files which are equivalent to what would have been produced had the problem been run with the original, undivided blocks. Using *SIXPAC* and *BACPAC*, a problem can be subdivided and reconstructed any number of ways to take advantage of available computer resources.

4.3.1 BACPAC Input

BACPAC queries the user for needed information, and reads from standard input (normally the keyboard). The user is first prompted for the casename. The user then selects which files are to be reconstructed by entering appropriate responses to questions about each file. Due to the potential size of these files, they are not created by default.

BACPAC expects to find a *case.bacpac* file which contains information detailing how the original problem was subdivided. The *case.bacpac* file is created automatically by *SIXPAC*, and requires no modifications by the user. However, if *SIXPAC* was not used to create the subdivided blocks, the user must construct a *case.bacpac* file in order to run *BACPAC*. A sample *case.bacpac* file, resulting from the sample *SIXPAC* input file shown in Figure 4.8, appears in Figure 4.10.

In the example, two blocks are subdivided into eight new blocks each (a total of 16 blocks). The dimensions of the original blocks are $73 \times 10 \times 9$, and there are 4, 2, and 1 subdivided blocks in each coordinate direction for each block. The table underneath each of the original block size declarations shows the original block number, and the new block number. The global *i*, *j*, and *k* indices are the position of the bottom right hand corner of the new block in the original block. For example, the point (1,1,1) in the new block 8 is the same as the point (17,9,9) in the original block 1. The local *im*, *jm*, and *km* indices are the block size of the new block. This data essentially maps the new blocks into the original block structure.

4.3.2 BACPAC Output

The output files produced by *BACPAC* are *Ncase.mesh.bac*, *Ncase.p3dabs.bac*, *Ncase.p3drel.bac*, and *Ncase.restart.bac*. If the one-equation or two-equation turbulence models are used options are also available to reconstitute the *Ncase.p3d1eq*, *Ncase.p3d2eq*,

```

      2  original number of blocks
imax      jmax      kmax
  73      10        9
nblki     nblkj     nblkk
  4        2        1
oldblk    newblk    global i  global j  global k  local im  local jm  local km
  1         1         1         1         1         19         6         9
  1         2        19         1         1         19         6         9
  1         3        37         1         1         19         6         9
  1         4        55         1         1         19         6         9
  1         5         1         6         1         19         5         9
  1         6        19         6         1         19         5         9
  1         7        37         6         1         19         5         9
  1         8        55         6         1         19         5         9
imax      jmax      kmax
  73      10        9
nblki     nblkj     nblkk
  4        2        1
oldblk    newblk    global i  global j  global k  local im  local jm  local km
  2         1         1         1         1         19         6         9
  2         2        19         1         1         19         6         9
  2         3        37         1         1         19         6         9
  2         4        55         1         1         19         6         9
  2         5         1         6         1         19         5         9
  2         6        19         6         1         19         5         9
  2         7        37         6         1         19         5         9
  2         8        55         6         1         19         5         9

```

Figure 4.10: Sample input file for *BACPAC* utility.

```

number of blocks
8
block #      proc #
1           0
2           1
3           1
4           2
5           1
6           1
7           2
8           2

```

Figure 4.11: Sample *case.blkproc* file used to distribute mesh blocks over parallel processors within *ADPAC* .

Ncase.restart_SA, and/or *Ncase.restart_KR* files. The *.bac* suffix is used to avoid confusion with existing files. Generally the *Ncase.mesh.bac* need not be created because it is identical to the original *case.mesh* file. If successfully run and converted, the *Ncase.*.bac* files should be moved or copied to replace their original problem equivalents (*case.**) before starting *SIXPAC* again.

4.4 Parallel ADPAC Block/Processor Assignment

Load balancing is a critical issue for parallel computing tasks. While it is beyond the scope of this program to perform detailed load balancing analyses for every parallel computing platform tested, it seems reasonable to provide some form of control in order to distribute computational tasks efficiently across a parallel computing network. In the parallel *ADPAC* code, this is best accomplished through manipulation of the block/processor distribution scheme. By default, the parallel operation of the *ADPAC* code provides an automatic block-to-processor assignment by dividing up the blocks as evenly as possible, and to the greatest degree possible assigning sequential block numbers on a given processor. For example, if 8 blocks were divided between 3 processors, blocks 1, 2, and 3 would be assigned to processor #0, blocks 4, 5, and 6 to processor #1, and blocks 7, and 8 to processor #2. Note that the processor numbering scheme *begins* with 0. This procedure is nearly optimal when each block is the same size and each processor has the same computational power. Unfortunately, our experience is that block sizes and computational resources often vary dramatically. In this regard, a system was developed which permits the user to specify the block to processor assignment through a special input file (*case.blkproc*). A sample *case.blkproc* file is shown in Figure 4.11 for an 8 block mesh distributed across 3 processors.

In the case described by the example file, block 1 is assigned to processor #0, blocks 2, 3, 5, and 6 to processor #1, and blocks 4, 7, and 8 to processor #2. This block assignment might be advisable for the case when block 1 is significantly larger in size than the other blocks, or if processor #0 has less memory or a slower CPU than the remaining processors. The original block assignment scheme is selected as the default when the *case.blkproc* file is not present. The *case.blkproc* file is also used with the

ADPAC memory-sizing utility code *ADSTAT* to determine the largest processor load and size the *ADPAC* parameters accordingly.

Chapter 5

ADPAC INTERACTIVE GRAPHICS DISPLAY

The *ADPAC* program is equipped with an option which permits real time interactive graphics display of flow data in the form of colored contours or velocity vectors on geometries represented by wiremesh grid surfaces. The interactive graphics are based largely on routines generated from the *PLOT3D* visualization program, and many of the features of this option should be familiar to anyone who has used *PLOT3D*. All interactive graphics must be displayed on a Silicon Graphics workstation, IRIX Operating System 4.0.1 or above. The graphics display can be operated on a single computing platform, or can be directed across a network for specific computer hardware configurations. Thus, it is possible to have a job running remotely on a Cray computer, with interactive graphics displayed locally on a network-connected Silicon Graphics workstation. When operating across a network which involves a non-Silicon Graphics computer, the communication program *AGTPLT-LCL* must be running on the local display device in order to capture the graphics commands issued by the remote compute server (details on *AGTPLT-LCL* are given below). A graphic illustrating the possible graphics display operating modes is given in Figure 5.1. It should be mentioned that the interactive graphics display was actually developed to aid in debugging the multiple block code. The description of this feature is included in this manual for completeness, but the user should be cautioned due to the immature nature of this portion of the code. It is also likely that the graphics option may not port correctly to future releases of the IRIX operating system, and again, the user is cautioned concerning the use of this feature.

5.1 Setting up the Program

The first step in producing the real time interactive graphics display is to correctly compile the code to include the graphics libraries. This is accomplished by utilizing the appropriate option in the *ADPAC* Makefile command. The valid graphics choices are *graphics*, *csdb_graphics*, and *dbx_graphics*. These options incorporate various levels of the included graphics libraries for execution on various machines.

Once the code has been correctly compiled to include the graphics libraries, several

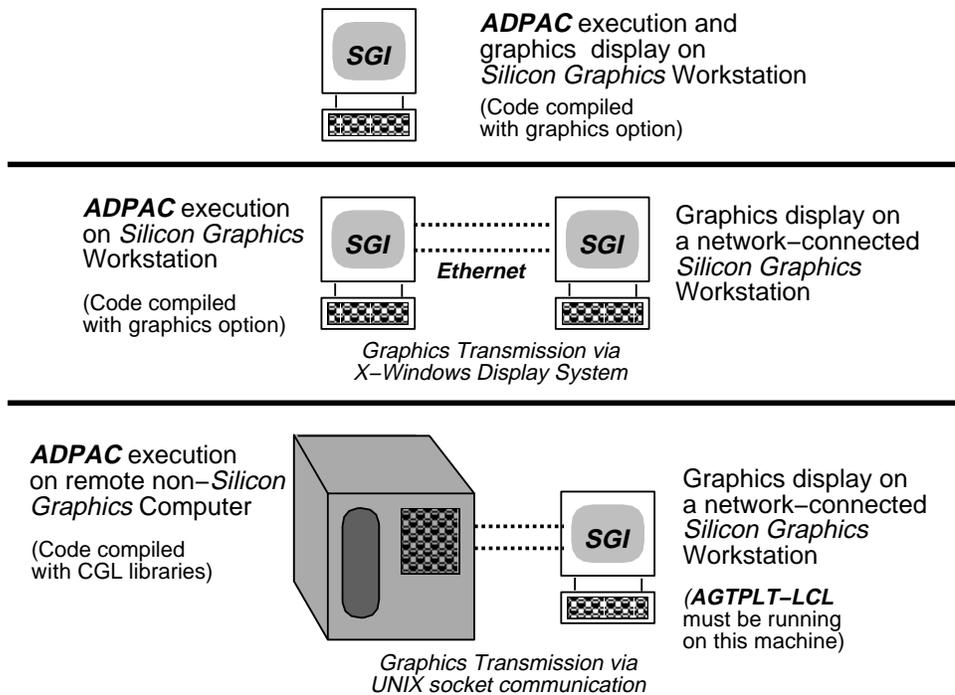


Figure 5.1: ADPAC interactive graphics display network configuration options.

input parameters must be correctly initiated to engage the graphics subroutines during the execution of the code. The input keyword **FGRAFIX** must have a value of 1.0 to initiate any graphics instructions. The keyword **FGRAFIN** determines the number of time-marching iterations between graphics window updates. The keyword **FIMGSAV** is a trigger (off = 0.0, on = 1.0) which determines whether periodic image capturing is enabled, and the keyword **FIMGINT** determines the number of time-marching iterations between image captures.

5.2 Graphics Window Operation

Once the graphics window has been initiated on the local display, and the initial data has been plotted, the program continues and the graphics display data are updated every **FGRAFIN** iterations. This process will continue until the program terminates, or until the user interrupts the process by pressing the left mouse button once with input focus directed to the graphics display window. A short time later, (the delay may be quite long for a network which is burdened), the graphics display will freeze, and the computational portions of the program will be suspended in order to permit the user to interactively translate, rotate, or scale the graphics image to their liking. When the display has been frozen, the viewpoint of the display may be altered by one of several mouse controls. The left mouse button controls rotation, the right mouse button controls translation, and the middle mouse button controls scaling (zoom in, zoom out). The mouse-directed viewpoint controls are identical to those used in *PLOT3D* [14]. Once the viewpoint has been altered, program control is returned to ADPAC by hitting the **ENTER** key on the keyboard with

input focus directed to the graphics window. At this point, the code will then return to the process of performing time-marching iterations, with periodic updating of the graphics screen.

It is also possible for the user to change the plotting function by entering any one of the following characters with input focus directed to the graphics window at any time during the process:

Key Result

- p Set flow function to pressure contours
- 2 Set flow function to velocity vectors

The surfaces plotted by the interactive graphics display is currently hardwired in the code. A wiremesh representation and the corresponding surface contours are generated for the $i=1$, $j=1$, and $k=1$ mesh surfaces. This restriction can be modified in future developments.

5.3 AGTPLT-LCL Program Description

The program *AGTPLT-LCL* is the receiving program for local graphics display of an *ADPAC* job running on a remote, network-connected computing platform. The *AGTPLT-LCL* program is a modified version of the NASA-AMES developed *PLOT3D-LCL* program. This program can only be run on a Silicon Graphics Workstation running at level 4.0.1 (or above) of the IRIX operating system. As such, compilation of the *AGTPLT-LCL* program has no options, and is performed simply by executing the command `make` in the *AGTPLT-LCL* source directory. Once initiated, the *AGTPLT-LCL* program waits for an outside process from *ADPAC* to communicate with the local workstation, and graphics commands received from the remote job are displayed locally.

An important consideration in setting up a remote calculation with local graphics display using *AGTPLT-LCL* is the manner in which the local display is defined in the calculation. The CGL libraries used to permit the network graphics instructions require an Internet network address in order to properly transmit the graphics commands to the correct destination. This definition should be provided in the standard input file following the normal keyword parameters. At the end of the standard input keyword data, the user should use an **ENDINPUT** statement to terminate the normal input stream. The **ENDINPUT** statement should then be followed by two blank lines, and then a line containing the destination network address of the local Silicon Graphics display device. This specification will ultimately be read by the CGL libraries in setting up the network connection.

The procedure to set up this network-connected graphics display option would be to start the job on the remote machine, and then immediately start the *AGTPLT-LCL* program on the local display. As long as the correct network address has been entered in the *case.input* file, then the remote program should begin communicating with the *AGTPLT-LCL* program, and the local graphics window will begin displaying the graphics instructions specified by the remote computing program. Again, this capability is not considered a mature technology in the *ADPAC* code and is not generally recommended.

Chapter 6

ADPAC UTILITY PROGRAMS

The standard distribution for the *ADPAC* program includes a number of tool programs designed to assist in examining and manipulating data generated for an *ADPAC* solution. Although running these programs is generally self-explanatory, a brief description is provided below to outline the function of each tool program. Prior to compiling any of these *ADPAC* utility programs, care should be taken in adjusting the program dimensions such that it is appropriately sized for the problem of interest. All the *ADPAC* utility programs which require reading/writing of mesh or flow solution files will need to be compiled including the Scientific Database Library (*SDBLIB*).

6.1 MAKEADGRID Program Description

The standard distribution for the *ADPAC* program includes a program called *MAKEADGRID* which aids the user in setting up a multiple-block mesh file from isolated *unformatted* mesh files. This program is useful for creating *ADPAC* compatible multiple-block meshes from mesh generation programs which do not support the use of the Scientific Database Library (*SDBLIB*). The *MAKEADGRID* program is an interactive program which queries the user for the number of blocks to be assembled for the final mesh, and then requests a file name for each of the individual mesh blocks. The user is then requested to name the final output file for the *ADPAC* compatible multiple-block mesh. The individual mesh blocks are assembled in the order in which the mesh file names are specified, so care must be taken to order these names appropriately.

6.2 COARSEN Tool Program Description

If a *ADPAC* mesh has been created with more than one level of multi-grid, the *COARSEN* utility can be used to create a duplicate geometry case with one multi-grid level fewer mesh points. The *COARSEN* program will read in an *ADPAC* mesh, input file, and boundary data file and remove every other mesh line in each of the computational directions (*i*, *j*, and *k*). The boundary data file will be updated to reflect the new indices of the coarsened mesh. An existing restart file will also be coarsened if triggered by the proper input variable. The new set of files will be named by replacing *case* with *caseC*. It

is sometimes useful to run an *ADPAC* analysis on the coarsened level to more quickly evaluate the solution and debug boundary conditions. The restart files obtained from a coarsened solution can also be used to restart the pre-*COARSEN* mesh problem by setting **FREST** = -1.0 (see input keyword **FREST**).

6.3 ADSPIN Tool Program Description

ADSPIN (*ADPAC* specific surface integrator) allows the user to specify any constant i , j , or k surface and the limits on that surface, and mass averages quantities over that surface. The output file includes mass flow rate, average total and static temperatures and pressures, and average directional and total velocity. The user can also specify a series of surfaces with the same limits, instead of entering them individually. *ADSPIN* has the capability of creating a journal file for each run was added. The journal file allows the running of several similar cases without the tedious manual selection of the same options.

6.4 ADSTAT Tool Program Description

The *ADSTAT* tool program was designed to provide statistical information about an *ADPAC* mesh. The program is run by typing the executable name followed by the *ADPAC* casename (**adstat case**). The *ADSTAT* program opens the mesh file and reports the number of mesh blocks contained within the file, as well as the individual mesh block sizes. The *ADSTAT* program also computes the maximum allowable number of multi-grid levels (based on mesh size alone) which can be used for an *ADPAC* run. In addition, the *ADSTAT* program computes and reports the minimum required *ADPAC* array size parameters for all allowable number of multi-grid levels. This capability is the most useful feature of the *ADSTAT* program. If no *case.blkproc* file is found in the directory, the minimum array dimensions reported are for the serial version of *ADPAC*. In parallel execution, *ADPAC* only needs to be dimensioned for the subset of blocks residing on the individual local processor. *ADSTAT* can use the mesh block allocation information in the *case.blkproc* file to determine the minimum dimensions required for *ADPAC* to run in parallel; depending on block allocation strategy, this can result in a substantial savings in memory requirements over running multiple *ADPAC* executables dimensioned for serial mode.

6.5 AOA2AXI Tool Program Description

The *AOA2AXI* tool program was designed to compute an axisymmetric average of a 3-D cylindrical coordinate system solution. The program is restricted to H-type meshes which possess uniform axisymmetric projections on each mesh plane in the circumferential direction (this simplifies the averaging process). When running *AOA2AXI*, the user is requested to enter the 3-D mesh and flow (*PLOT3D* format) file names. Then, the user is offered the option of redimensionalizing the data, and finally, the user is requested to enter the 2-D axisymmetric mesh and flow (*PLOT3D* format) file names. The *AOA2AXI* code computes the axisymmetric average of the 3-D mesh and flow file data and stores the

```

CASENAME = case      ADPAC case name
FMULTI   = 3.0       Three mesh levels for multi-grid
FCART    = 0.0       Cartesian/Cylindrical geometry trigger
PITCH    = 0.00      Blade pitch used to evaluate periodic boundaries in degrees
FWALLS   = 1.0       Wall B.C. trigger (0.0=none, 1.0=SSVI, -1.0=SSIN)
RPM      = 0.00      Rotational speed of viscous walls (SSVI) in RPM
FPRINT   = 0.0       Level of detail in standard output generated
TOLPERC  = 1.0       Percentage of minimum distance used as tolerance

```

Figure 6.1: Sample input file for *PATCHFINDER* utility.

result in the 2-D axisymmetric mesh and flow files. These data may then be used with *PLOT3D* and other graphics visualization tools to examine the axisymmetric average of the 3-D solution.

6.6 PATCHFINDER Tool Program Description

An *ADPAC* utility program was developed to aid in the construction of boundary condition files for complex, interconnected multiple block mesh systems. The utility, named *PATCHFINDER*, reads in an *ADPAC* mesh (a *PLOT3D SDBLIB* -binary multiple-block grid) and determines which blocks share matching faces. As the grid is read, the faces are striped off into a separate array. Individual points on these faces are compared until a “point” match is found. Neighboring points are then compared to find a “cell” match. This also determines the relative directions of the matching indices. From this cell match, a common face area is swept out and a **PATCH** boundary condition statement is written using the bounding indices of the common face area.

After all the **PATCH** specifications have been written, any remaining surfaces not accounted for will have a solid surface wall (**SSVI** or **SSIN**) boundary condition written out. This allows the user to simply replace a few solid wall statements with the proper inlet and exit boundary conditions and start running. The *PATCHFINDER* user input file contains approximately ten variables to customize a *PATCHFINDER* run to each grid although many grids can be processed without special input. Since the majority of boundary conditions prescribed in most geometries are block patches and solid walls, running *PATCHFINDER* will greatly simplify the generation of a boundary data file.

Several methods were implemented to accelerate the *PATCHFINDER* search and compare process. One of these methods includes using multi-grid. If a mesh is created to be run with *ADPAC* using multi-grid, this can also be used to accelerate *PATCHFINDER* since boundary conditions must be consistent across multi-grid levels. Each level of multi-grid decreases the number of face points by a factor of 4; therefore, with 3 levels of multi-grid a decrease in run time of roughly sixteen times can be expected.

PATCHFINDER uses the same input file format as *ADPAC* and some of the same input variables. The program is executed by redirecting the input file, similar to *ADPAC* (*patchfinder < patch.in*). A sample input file (*patch.in*) is listed in Figure 6.1.

6.7 Miscellaneous Tool Programs Description

A number of miscellaneous tool programs are included with the *ADPAC* distribution because of their usefulness in manipulating *ADPAC* mesh and flow files. A brief description of these utility programs is listed below. It should be noted that these programs are designed to deal with *SDBLIB* binary formatted files (the *ADPAC* standard binary format).

- **bin2unf** Converts a *SDBLIB* binary mesh file to a *PLOT3D* unformatted mesh file.
- **breakup** Creates a series of single-block binary mesh and flow files from a multiple-block mesh and flow file.
- **checkbal** Reports the processor load for a parallel *ADPAC* run given an *ADPAC* mesh file and *case.blkproc* file.
- **countpts** Quickly interrogates the header lines from a mesh, flow, or restart file and reports grid block sizes and total number of points.
- **cutmesh, cutflow, cutrest** Extracts a sub-domain defined through interactive index selection from an existing mesh, flow, or restart file.
- **flowinfo** Reads a *PLOT3D* flow file and returns minimum and maximum flow variables and their respective locations, also allows for individual point interrogation.
- **flowmix** Similar to **meshmix**, but for *PLOT3D* flow files.
- **meshmix** Combines separate multiple-blocked or single-blocked mesh files into one multiple-block mesh file.
- **restmix** Similar to **meshmix**, but for *ADPAC* restart files.
- **restinfo** Reads an *ADPAC* restart file and returns minimum and maximum flow variables and their respective locations, also allows for individual point interrogation.
- **right2left** Converts a right-handed mesh to a left-handed mesh by reversing the direction of the *k* index.
- **rotate** Rotates a multiple-block mesh about the X-axis by a user specified angle.
- **scale** Translates, scales, and/or mirrors a multiple-block *ADPAC* mesh.
- **swapjk** Exchanges the *j* and *k* index direction.
- **unf2bin** Converts a *PLOT3D* unformatted mesh file to a *SDBLIB* binary mesh file.

6.8 PLOTBC Tool Program Description

In order to facilitate a graphical examination of an *ADPAC* boundary data file, a utility program called *PLOTBC* was created. The *PLOTBC* program reads in a user-specifiable *ADPAC* mesh and boundary data file and creates several *PLOT3D* -compatible command

files. The command files, in conjunction with the mesh file, permit the user to graphically examine (using the *PLOT3D* program) a number of features of the mesh construction and boundary condition specifications. This utility provides a rapid means of assessing the completeness of a boundary data file and provides a visual method for determining the characteristics of an *ADPAC* computational model.

The *PLOTBC* program is invoked by simply running the executable and entering the *ADPAC* case name when prompted. After entering the case name, the *PLOTBC* program reads in the *ADPAC* mesh and boundary data files, extracts the boundary conditions and organizes them into categories. Each category is then used to construct a *PLOT3D* command file which allows the user to visualize all boundary conditions in a common category. The resulting *PLOT3D* command files and their functions are listed below:

axes.com	Grid index orientation
bcprm.com	Inter-block BCPRM boundaries
bcprp.com	Inter-block BCPRR boundaries
inext.com	Inflow/outflow boundaries
mbcavg.com	Inter-block MBCAVG boundaries
outline.com	Mesh block outline
patch.com	Inter-block PATCH boundaries
pint.com	Inter-block PINT boundaries
solid.com	Solid surfaces (viscous and inviscid, rotating and non-rotating)

Each of the *ADPAC* boundary conditions identified by *PLOTBC* is color-coded such that all the command files can be read sequentially, thus displaying all the boundaries at once. Once created, the *PLOTBC* command files may be used with *PLOT3D* by reading in the corresponding mesh file, and then invoking one or more of the scripts. Once the *PLOT3D* program is initialized, the mesh file should be read in using the standard *PLOT3D* commands, and then the command files may be invoked by the *PLOT3D* command *@comfile*, where *comfile* is one of the files listed above. The action of the command file is to essentially define what is to be plotted. The actual plotting is not performed until the user enters the plot command at the *PLOT3D* prompt. Additional details may be found in the *PLOT3D* User's Manual [14].

In addition to creating *PLOT3D* scripts, command scripts are also generated by *PLOTBC* for *FAST* and *Fieldview*, two additional visualization packages. These additional scripts are output *only* for the solid surfaces, rather than all the boundary conditions. Therefore, for boundary condition checking and debugging, *PLOT3D* should be used, and for post-processing visualization, the *FAST* and *Fieldview* scripts should be used to quickly render the solid surfaces.

References

- [1] Hall, E. J. and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task V - Counterrotation Ducted Propfan Analysis, Final Report," NASA CR-187126, NASA Contract NAS3-25270, 1992.
- [2] Hall, E. J., Topp, D. A., Heidegger, N. J., and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task VII - Endwall Treatment Inlet Flow Distortion Analysis Final Report," NASA CR-195468, NASA Contract NAS3-25270, July 1995.
- [3] Hall, E. J., Topp, D. A., Heidegger, N. J., and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task VIII - Cooling Flow/Heat Transfer Analysis, Final Report," NASA CR-195359, NASA Contract NAS3-25270, September 1994.
- [4] Rao, K. V. and Delaney, R. A., "Investigation of Unsteady Flow Through a Transonic Turbine Stage: Part I - Analysis," AIAA Paper 90-2408, 1990.
- [5] Jorgensen, P. C. E. and Chima, R. V., "An Unconditionally Stable Runge-Kutta Method for Unsteady Flows," NASA TM-101347, AIAA Paper 89-0205, 1989.
- [6] Rai, M. M., "Unsteady Three-Dimensional Navier-Stokes Simulations of Turbine Rotor-Stator Interaction," AIAA Paper 87-2058, 1987.
- [7] Adamczyk, J. J., "Model Equation for Simulating Flows in Multistage Turbomachinery," ASME Paper 85-GT-226, 1985.
- [8] Dawes, W.N., "Multi-Blade Row Navier-Stokes Simulations of Fan Bypass Configurations," ASME Paper 91-GT-148, 1991.
- [9] Goyal, R. K. and Dawes, W. N., "A Comparison of the Measured and Predicted Flowfield in a Modern Fan-Bypass Configuration," ASME Paper 92-GT-298, 1992.
- [10] Hall, E. J., "Aerodynamic Modeling of Multistage Compressor Flowfields - Part 1: Analysis of Rotor/Stator/Rotor Aerodynamic Interaction", ASME Paper 97-GT-344, 1997.
- [11] Hall, E. J., "Aerodynamic Modeling of Multistage Compressor Flowfields - Part 2: Modeling Deterministic Stresses", ASME Paper 97-GT-345, 1997.
- [12] Crook, A. J. and Delaney, R. A., "Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task IV -

-
- Advanced Fan Section Analysis, Final Report,” NASA CR-187128, NASA Contract NAS3-25270, 1992.
- [13] Steinbrenner, J., et. al. “The Gridgen 3D Multiple Block Grid Generation System,” Final Report WRDC-TR-90-3022, 1990.
- [14] Walatka, P. P., Buning, P. G., Pierce, L., and Elson, P. A., “PLOT3D User’s Manual,” NASA TM-101067, March 1990.
- [15] Walatka, P. P. and Buning, P. G., “FAST,” NASA Ames Research Center, 1990.
- [16] “IEEE Standard for Binary Floating-Point Arithmetic,” Standards Committee of the IEEE Computer Society, ANSI/IEEE Std 754-1985, 1985.
- [17] “XDR: External Data Representation Standard,” Sun Microsystems, Inc., June 1987.
- [18] Granville, P. S., “Baldwin-Lomax Factors for Turbulent Boundary Layers in Pressure Gradients,” *AIAA Journal*, Vol. 25, No. 12, pp. 1624-1627, December 1987.
- [19] Quealy, A., Cole, G. L., and Blech, R. A., “Portable Programming on Parallel/Networked Computers Using the Application Portable Parallel Library (APPL),” NASA TM-106238, July 1993.
- [20] Hall, E. J. and Delaney, R. A., “Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task VII - ADPAC User’s Manual,” NASA CR-195472, NASA Contract NAS3-25270, July 1995.
- [21] Sunderam, “PVM: A Framework for Parallel Distributed Computing,” *Concurrency: Practice & Experience*, Vol. 2, No. 4, 1990.
- [22] “MPI: A Message-Passing Interface Standard,” Message Passing Interface Forum, May 5, 1994, University of Tennessee, Knoxville, Report No. CS-94-230, (see also the International Journal of Supercomputing Applications, Volume 8, Number 3/4, 1994).
- [23] Hall, E. J., Delaney, R. A., and Bettner, J. L., “Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task I - Ducted Propfan Analysis,” NASA CR-185217, NASA Contract NAS3-25270, 1990.
- [24] Hall, E. J. and Delaney, R. A., “Investigation of Advanced Counterrotation Blade Configuration Concepts for High Speed Turboprop Systems: Task II - Unsteady Ducted Propfan Analysis - Final Report,” NASA CR-187106, NASA Contract NAS3-25270, 1992.
- [25] Barber, T., Choi, D., McNulty, G., Hall, E., and Delaney, R., “Preliminary Findings in Certification of ADPAC,” AIAA Paper 94-2240, June, 1994.
- [26] Baldwin, B. S. and Lomax, H., “Thin Layer Approximation and Algebraic Model for Separated Turbulent Flows,” AIAA Paper 78-257, AIAA 16th Aerospace Sciences Meeting, Huntsville, AL, January 1978.
- [27] Anderson, D. A., Tannehill, J. C., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer*, McGraw-Hill, New York, New York, 1984.

-
- [28] Boussinesq, J., "Essai Sur La Théorie Des Eaux Courantes," *Mem. Présentés Acad. Sci.*, Vol. 23, Paris, pp. 46-50, 1877.
- [29] Jameson, A., Schmidt, W., and Turkel, E., "Numerical Solutions of the Euler Equations by Finite Volume Methods Using Runge-Kutta Time-Stepping Schemes," AIAA Paper 81-1259, 1981.
- [30] Hung, C. M. and Kordulla, W., "A Time-Split Finite Volume Algorithm for Three-Dimensional Flow-Field Simulation," AIAA Paper 83-1957, 1983.
- [31] Martinelli, L., "Calculation of Viscous Flows with a Multigrid Method," Ph. D. Dissertation, MAE Department, Princeton University, 1987.
- [32] Hollanders, H., Lerat, A., and Peyret, R., "Three-Dimensional Calculation of Transonic Viscous Flows by an Implicit Method," *AIAA Journal*, Vol. 23, pp. 1670-1678, 1985.
- [33] Radespiel, R., Rossow, C., and Swanson, R. C., "Efficient Cell Vertex Multigrid Scheme for the Three-Dimensional Navier-Stokes Equations," *AIAA Journal*, Vol. 28, No. 8, pp. 1464-1472, 1990.
- [34] Arnone, A. A., Liou, M. S., and Povinelli, L. A., "Multigrid Time-Accurate Integration of Navier-Stokes Equations," AIAA Paper 93-3361-CP, 1993
- [35] Arnone, A., Pacciani, R., and Sestini, A., "Multigrid Computations of Unsteady Rotor-Stator Interaction Using the Navier-Stokes Equations," Submitted for Presentation to the 1995 ASME Gas Turbine Conference, 1995.
- [36] Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," AIAA Paper 91-1596, 1991.
- [37] Melson, N. D., Sabetrik, M. D., and Atkins, H. L., "Time-Accurate Navier-Stokes Calculations with Multigrid Acceleration," Presented at the Sixth Copper Mountain Conference on Multigrid Methods, Copper Mountain, Colorado, April 4-9, 1993.
- [38] Kreiss, H. O., "Initial Boundary Value Problems for Hyperbolic Systems," *Communications on Pure and Applied Mathematics*, Vol. 23, pp. 277-298, 1970.
- [39] Engquist, B. and Majda, A., "Absorbing Boundary Conditions for the Numerical Simulation of Waves," *Mathematics of Computation*, Vol. 31, pp. 629-651, 1977.
- [40] Erdos, J. I., Alzner, E., and McNally, W., "Numerical Simulation of Periodic Transonic Flow Through a Fan Stage," *AIAA Journal*, Vol. 15, pp. 1559-1568, 1977.
- [41] Giles, M. B., "Nonreflecting Boundary Conditions for Euler Equation Calculations," *AIAA Journal*, Vol. 28, No. 12, pp. 2050-2058, 1990.
- [42] Saxer, A. P., "A Numerical Analysis of 3-D Inviscid Stator/Rotor Interactions Using Non-Reflecting Boundary Conditions," MIT GTL Report 209, 1992.
- [43] Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 92-0439, AIAA 30th Aerospace Sciences Meeting & Exhibit, Reno, NV, January 1992.
- [44] Spalart, P. R. and Allmaras, S. R., "A one-equation turbulence model for aerodynamic flows," *La Recherche Aéronautique*, No. 1, pp. 5-21, 1994.

-
- [45] Spalart, P. R., "Improvement in Spalart-Allmaras Model," Boeing Commercial Airplane Company, March 1993.
- [46] Turner, M. G. and Jennions, I. K., "An Investigation of Turbulence Modeling in Transonic Fans Including a Novel Implementation of an Implicit $k-\epsilon$ Turbulence Model," *Journal of Turbomachinery*, Vol. 115, April 1993, p. 249-260.
- [47] Spalding, D. P., "Monograph on Turbulent Boundary Layer," Imperial College, Mechanical Engineering Department Report TWF/TN/33, 1976.
- [48] Wolfshtein, M. W., "The Velocity and Temperature Distribution in One Dimensional Flow with Turbulence Augmentation and Pressure Gradient," *International Journal of Heat and Mass Transfer*, Vol. 12, p. 301, 1969.
- [49] Patankar, S. V. and Spalding, D. B., *Heat and Mass Transfers in Boundary Layers, 2nd Edition*, Intertext Books, London, 1970.
- [50] Launder, B. E. and Spalding, D. B., "The Numerical Computation of Turbulent Flows," *Computer Methods in Applied Mechanics and Engineering*, Vol. 3, pp.269-289, 1974.
- [51] Heidegger, N. J., Hall, E. J., and Delaney, R. A., "Follow-on Low Noise Fan Aerodynamic Study, Task 15 - Final Report," NASA CR-206599, NASA Contract NAS3-27394, 1998.
- [52] Goldberg, U. C., "Towards a Pointwise Turbulence Model for Wall-Bounded and Free Shear Flows," *Boundary Layer and Free Shear Flows*, ASME FED-Vol. 184, pp. 113-118, 1994.
- [53] Baldwin, B. S. and Barth, T. J., "A One-Equation Turbulence Transport Model for High Reynolds Wall-Bounded Flows," AIAA Paper 91-0610, AIAA 29th Aerospace Sciences Meeting, Reno, NV, January 1991.
- [54] Wilcox, D. C., "Turbulence Modeling for CFD," DCW Industries, La Cañada, California, 1993.
- [55] Launder, B. E. and Sharma, B. I., "Application of the Energy Dissipation Model of Turbulence to the Calculation of Flow Near a Spinning Disk," *Letters in Heat and Mass Transfer*, Vol. 1, pp. 131-138, 1974.

Appendix A

ADPAC NAVIER-STOKES NUMERICAL ALGORITHM

The *ADPAC* code is a general purpose turbomachinery aerodynamic design analysis tool which has undergone extensive development, testing, and verification [23], [24], [1], [25]. Briefly, the *ADPAC* analysis utilizes a finite-volume, multi-grid, Runge-Kutta time-marching solution algorithm to solve a time-dependent form of the 3-D Reynolds-Averaged Navier-Stokes equations. The selection of turbulence models can be made from a wide range of models varying in complexity and computational expense. A relatively standard Baldwin-Lomax [26] turbulence model was incorporated to compute the turbulent shear stresses. A simple mixing-length turbulence model is also available. The more complex one-equation Spalart-Allmaras and a two-equation k - \mathcal{R} turbulence models were incorporated into *ADPAC* for enhanced turbulent flow predictions. Each of these turbulence models is detailed later within this appendix.

The code employs a multiple-blocked mesh discretization which provides extreme flexibility for analyzing complex geometries. The block gridding technique enables the coupling of complex, multiple-region domains with common grid interface boundaries through specialized boundary condition procedures. The *ADPAC* analysis has been successfully utilized to predict both the steady state and time-dependent aerodynamic interactions occurring in modern multistage compressors and turbines.

In this appendix, the governing equations and computational model methodology for the *ADPAC* code are described. The definitions of the pertinent variables used in this appendix may be found in Nomenclature.

A.1 Nondimensionalization

To simplify the implementation of the numerical solution, all variables are nondimensionalized by reference values as follows (note that variables with the caret (i.e., $\hat{\phi}$) are *dimensional* variables and consequently variables *without* a caret (i.e., ϕ) are

nondimensional variables):

$$\begin{aligned}
y &= \frac{\hat{y}}{\hat{L}_{ref}}, & z &= \frac{\hat{z}}{\hat{L}_{ref}}, & v_x &= \frac{\hat{v}_x}{\hat{V}_{ref}}, & v_y &= \frac{\hat{v}_y}{\hat{V}_{ref}}, & v_z &= \frac{\hat{v}_z}{\hat{V}_{ref}} \\
x &= \frac{\hat{x}}{\hat{L}_{ref}}, & r &= \frac{\hat{r}}{\hat{L}_{ref}}, & v_{ax} &= \frac{\hat{v}_{ax}}{\hat{V}_{ref}}, & v_r &= \frac{\hat{v}_r}{\hat{V}_{ref}}, & v_\theta &= \frac{\hat{v}_\theta}{\hat{V}_{ref}} \\
p &= \frac{\hat{p}}{\hat{p}_{ref}}, & \mu &= \frac{\hat{\mu}}{\hat{\mu}_{ref}}, & c_p &= \frac{\hat{c}_p}{\hat{R}_{ref}}, & c_v &= \frac{\hat{c}_v}{\hat{R}_{ref}}, & k &= \frac{\hat{k}}{\hat{k}_{ref}} \\
T &= \frac{\hat{T}}{\hat{T}_{ref}}, & \rho &= \frac{\hat{\rho}}{\hat{\rho}_{ref}}, & k &= \frac{\hat{k}}{\hat{V}_{ref}^2}, & \mathcal{R} &= \frac{\hat{\mathcal{R}}}{\hat{V}_{ref} \hat{L}_{ref}}, \\
t &= \frac{\hat{t}}{\hat{L}_{ref} / \hat{V}_{ref}}, & f &= \frac{\hat{f}}{\hat{V}_{ref}^2 / \hat{L}_{ref}},
\end{aligned} \tag{A.1}$$

The reference quantities are defined as follows:

- \hat{L}_{ref} is a constant length scale (user-defined **DIAM**)
- \hat{p}_{ref} is normally the inlet total pressure (user-defined **PREF**)
- \hat{T}_{ref} is normally the inlet total temperature (user-defined **TREF**)
- \hat{R}_{ref} is the freestream gas constant (user-defined **RGAS**)
- $\hat{\rho}_{ref}$ is the freestream or inlet total density ($\hat{\rho}_{ref} = \hat{p}_{ref} / \hat{R}_{ref} / \hat{T}_{ref}$)
- \hat{V}_{ref} is determined from the freestream total acoustic velocity as:
$$\hat{V}_{ref} = \frac{\hat{a}_{ref}}{\sqrt{\gamma}} = \sqrt{\hat{R}_{ref} \hat{T}_{ref}}$$
- $\hat{\mu}_{ref}$ is determined from the other factors as:
$$\hat{\mu}_{ref} = \hat{\rho}_{ref} \hat{V}_{ref} \hat{L}_{ref}$$
- \hat{k}_{ref} is the freestream thermal conductivity (extracted from user-defined parameters such as γ and Prandtl number)

A.2 Governing Equations

The *ADPAC* numerical solution procedure is based on an integral representation of the strong conservation law form of the 3-D Reynolds-averaged Navier-Stokes equations expressed in either a cylindrical or Cartesian coordinate system. User input determines which solution scheme is selected, and can be varied on a block by block basis. The Euler equations may be derived as a subset of the Navier-Stokes equations by neglecting viscous dissipation and thermal conductivity terms (i.e., μ and $k = 0$).

The derivations of the various forms of the equations employed in the *ADPAC* code are outlined below.

A.2.1 Vector Form of Navier-Stokes Equations

The Navier-Stokes equations may be efficiently described in a coordinate independent vector form as follows (see e.g. [27]):

Continuity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{V}) = 0 \quad (\text{A.2})$$

Momentum

$$\frac{\partial(\rho \vec{V})}{\partial t} + \nabla \cdot \rho \vec{V} \vec{V} = \rho \vec{f} + \nabla \cdot \mathbf{\Pi}_{ij} \quad (\text{A.3})$$

Energy

$$\frac{\partial(\rho e)}{\partial t} + \nabla \cdot (\rho e) \vec{V} = \frac{\partial Q}{\partial t} - \nabla \cdot \vec{q} + \rho \vec{f} \cdot \vec{V} + \nabla \cdot (\mathbf{\Pi}_{ij} \cdot \vec{V}) \quad (\text{A.4})$$

Here ρ is density, \vec{V} is the fluid velocity vector, e is the fluid total internal energy, t is time, ∇ is the spatial gradient operator, $\mathbf{\Pi}_{ij}$ is the fluid stress tensor, \vec{f} is an external force vector, Q represents added heat, and \vec{q} is the fluid conduction heat flux vector.

A.2.2 Reynolds-Averaged Form of Navier-Stokes Equations

Direct computation of turbulent flows using the Navier-Stokes equations in the form above is simply not practical at this point, and instead, we assume that the turbulence is *stationary* (see e.g. Wilcox [54]), and can be effectively represented numerically as a time-averaged effect. In this respect, it is useful to derive the Reynolds-averaged form of the Navier-Stokes equations by introducing time averaging operators. Any instantaneous flow variable $f(x, t)$ can be decomposed into a time-averaged and a fluctuating component as

$$f(x, t) = \bar{f}(x) + f'(x, t) \quad (\text{A.5})$$

The time average $\bar{f}(x)$ is defined as

$$\bar{f}(x) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_t^{t+T} f(x, t) dt \quad (\text{A.6})$$

Similarly, for compressible flows, it is useful to define the density weighted time average as

$$f(x, t) = \tilde{f}(x) + f''(x, t) \quad (\text{A.7})$$

where now the density weighted time averaged variable is defined as

$$\tilde{f}(x) = \frac{\overline{\rho f}}{\bar{\rho}} \quad (\text{A.8})$$

Application of the mass-weighted averaging procedure to the Navier-Stokes equations (see e.g. [27]) yields the Reynolds-averaged Navier-Stokes equations expressed in vector form as

Continuity

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial}{\partial x_j} (\bar{\rho} \tilde{u}_j) = 0 \quad (\text{A.9})$$

Momentum

$$\frac{\partial}{\partial t} (\bar{\rho} \tilde{u}_i) + \frac{\partial}{\partial x_j} (\bar{\rho} \tilde{u}_i \tilde{u}_j) = -\frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} (\bar{\tau}_{ij} - \overline{\rho u_i'' u_j''}) \quad (\text{A.10})$$

Energy

$$\frac{\partial}{\partial t} (\bar{\rho} \tilde{H}_{total}) + \frac{\partial}{\partial x_j} (\bar{\rho} \tilde{u}_j \tilde{H}_{total} + \overline{\rho u_j'' H_{total}''} - k \frac{\partial \bar{T}}{\partial x_j}) = \frac{\partial \bar{p}}{\partial t} + \frac{\partial}{\partial x_j} (\tilde{u}_i \bar{\tau}_{ij} + \overline{u_i'' \tau_{ij}}) \quad (\text{A.11})$$

where

$$\overline{\tau_{ij}} = \mu \left[\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} \frac{\partial \tilde{u}_k}{\partial x_k} \right] + \mu \left[\left(\frac{\partial \overline{u_i''}}{\partial x_j} + \frac{\partial \overline{u_j''}}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} \frac{\partial \overline{u_k''}}{\partial x_k} \right] \quad (\text{A.12})$$

where δ_{ij} is the Kronecker delta function ($\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$) and u_i represents the velocity vector components. The complication in this analysis is the presence of terms of the form $\overline{\rho u_i'' u_j''}$. These terms are often referred to as Reynolds stresses, and the specification of these terms is referred to as the turbulent *closure* problem. A large portion of turbulence modeling research is dedicated to suitably closing the system of equations by defining procedures to compute the Reynolds stress terms. In this study, turbulence closure is performed by employing the Boussinesq approximation. Boussinesq [28] suggested that the apparent turbulent stresses might be related to the mean strain rate through an *eddy* viscosity of the form

$$-\overline{\rho u_i'' u_j''} = \mu_t \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) - \frac{2}{3} \delta_{ij} \left(\mu_t \frac{\partial \tilde{u}_k}{\partial x_k} + \bar{\rho} \bar{k} \right) \quad (\text{A.13})$$

where \bar{k} is the *kinetic energy of turbulence* defined as $\bar{k} = \overline{u_i'' u_i''} / 2$. The resulting simplification is that all Reynolds stress terms are eliminated in favor of a modified viscosity $\mu_{eff} = \mu_{lam} + \mu_t$ where μ_t is the eddy viscosity described above. The turbulent flow thermal conductivity term is also treated as the combination of a laminar and turbulent quantity as:

$$k_{eff} = k_{lam} + k_t \quad (\text{A.14})$$

For turbulent flows, the turbulent thermal conductivity k_t is determined from a turbulent Prandtl number Pr_t such that:

$$Pr_t = \frac{c_p \mu_t}{k_t} \quad (\text{A.15})$$

The turbulent Prandtl number is normally chosen to have a value of 0.9. The turbulence models described later in this report define the means by which μ_t is prescribed.

Coordinate dependent forms of the Reynolds-averaged Navier-Stokes equations used in the numerical solution procedures are given in the sections which follow.

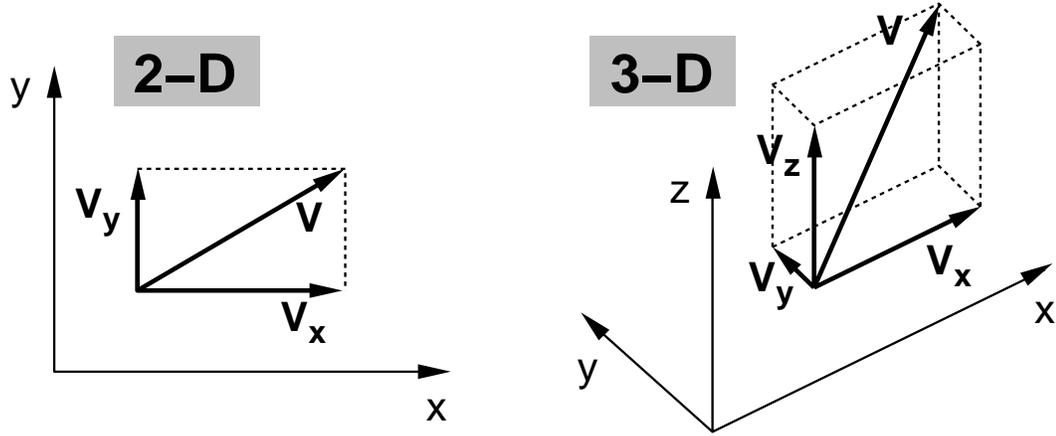


Figure A.1: ADPAC Cartesian coordinate system reference.

A.2.3 Governing Equations for Cartesian Solution

In this section, the governing equations for a Cartesian coordinate system solution are developed. In this discussion, since all solutions for turbulent flow employ the Boussinesq approximation, the overscores denoting time averaged (i.e., $\bar{\rho}$) and density weighted time averaged (i.e., \bar{v}_x) have been removed for simplicity.

The Reynolds-averaged Navier-Stokes equations for a Cartesian coordinate system may be written as:

$$\frac{\partial Q}{\partial t} + \frac{\partial F_{inv}}{\partial x} + \frac{\partial G_{inv}}{\partial y} + \frac{\partial H_{inv}}{\partial z} = S + \frac{\partial F_{vis}}{\partial x} + \frac{\partial G_{vis}}{\partial y} + \frac{\partial H_{vis}}{\partial z} \quad (\text{A.16})$$

For a Cartesian solution, the vector of dependent variables Q is defined as:

$$Q = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_y \\ \rho v_z \\ \rho e_t \end{bmatrix} \quad (\text{A.17})$$

where the velocity components v_x , v_y , and v_z are the absolute velocity components in the x , y , and z coordinate directions, respectively (Figure A.1).

The total internal energy is defined as:

$$e_t = \frac{p}{(\gamma - 1)\rho} + \frac{1}{2}(v_x^2 + v_y^2 + v_z^2) \quad (\text{A.18})$$

The individual flux functions are defined as:

$$F_{inv} = \begin{bmatrix} \rho v_x \\ \rho v_x^2 + p \\ \rho v_x v_y \\ \rho v_x v_z \\ \rho v_x H_{total} \end{bmatrix}, \quad G_{inv} = \begin{bmatrix} \rho v_y \\ \rho v_x v_y \\ \rho v_y^2 + p \\ \rho v_y v_z \\ \rho v_y H_{total} \end{bmatrix}, \quad H_{inv} = \begin{bmatrix} \rho v_z \\ \rho v_x v_z \\ \rho v_y v_z \\ (\rho v_z^2 + p) \\ \rho v_z H_{total} \end{bmatrix}, \quad (\text{A.19})$$

$$F_{vis} = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ q_x \end{bmatrix}, \quad G_{vis} = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ q_y \end{bmatrix}, \quad H_{vis} = \begin{bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ q_z \end{bmatrix} \quad (\text{A.20})$$

The total enthalpy, H_{total} , is related to the total energy by:

$$H_{total} = e_t + \frac{p}{\rho} \quad (\text{A.21})$$

The viscous stress and heat flux terms may be expressed as:

$$\tau_{xx} = 2\mu \left(\frac{\partial v_x}{\partial x} \right) + \lambda_v \nabla \cdot \vec{V}, \quad (\text{A.22})$$

$$\tau_{xy} = \mu \left[\left(\frac{\partial v_y}{\partial x} \right) + \left(\frac{\partial v_x}{\partial y} \right) \right], \quad (\text{A.23})$$

$$\tau_{xz} = \mu \left[\left(\frac{\partial v_y}{\partial z} \right) + \left(\frac{\partial v_z}{\partial x} \right) \right], \quad (\text{A.24})$$

$$\tau_{yy} = 2\mu \left(\frac{\partial v_y}{\partial y} \right) + \lambda_v \nabla \cdot \vec{V}, \quad (\text{A.25})$$

$$\tau_{yz} = \mu \left[\left(\frac{\partial v_y}{\partial x} \right) + \left(\frac{\partial v_z}{\partial y} \right) \right], \quad (\text{A.26})$$

$$\tau_{zz} = 2\mu \left(\frac{\partial v_z}{\partial z} \right) + \lambda_v \nabla \cdot \vec{V}, \quad (\text{A.27})$$

$$q_x = v_x \tau_{xx} + v_y \tau_{xy} + v_z \tau_{xz} + k \frac{\partial T}{\partial x}, \quad (\text{A.28})$$

$$q_y = v_x \tau_{yx} + v_y \tau_{yy} + v_z \tau_{yz} + k \frac{\partial T}{\partial y}, \quad (\text{A.29})$$

$$q_z = v_x \tau_{zx} + v_y \tau_{zy} + v_z \tau_{zz} + k \frac{\partial T}{\partial z} \quad (\text{A.30})$$

where μ is the first coefficient of viscosity, λ_v is the second coefficient of viscosity, and:

$$\nabla \cdot \vec{V} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \quad (\text{A.31})$$

The remaining viscous stress terms are defined through the identities

$$\tau_{yx} = \tau_{xy}, \quad (\text{A.32})$$

$$\tau_{zy} = \tau_{yz}, \quad (\text{A.33})$$

$$\tau_{zx} = \tau_{xz} \quad (\text{A.34})$$

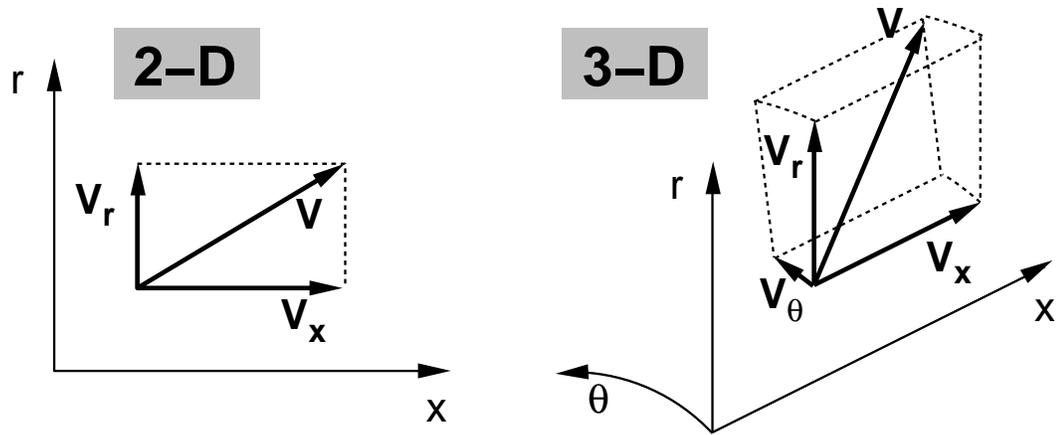


Figure A.2: ADPAC cylindrical coordinate system reference.

A.2.4 Governing Equations for Cylindrical Coordinate Solution

In this section, the governing equations for a rotating cylindrical coordinate system solution are developed. The rotating coordinate system permits the solution of rotating geometries such as turbomachinery blade rows. The rotation is always assumed to be about the x axis. In this discussion, since all solutions for turbulent flow employ the Boussinesq approximation, the overbars denoting time averaged (i.e., $\bar{\rho}$) and density weighted time averaged (i.e., \tilde{v}_r) have been removed for simplicity.

The Reynolds-averaged Navier-Stokes equations for a rotating cylindrical coordinate system may be written as:

$$\frac{\partial Q}{\partial t} + \frac{\partial F_{inv}}{\partial x} + \frac{\partial G_{inv}}{\partial r} + \frac{1}{r} \frac{\partial H_{inv}}{\partial \theta} = K + \frac{\partial F_{vis}}{\partial x} + \frac{\partial G_{vis}}{\partial r} + \frac{1}{r} \frac{\partial H_{vis}}{\partial \theta} \quad (\text{A.35})$$

For solutions employing the cylindrical coordinate system, the vector form of the equations contains only minor deviations from the Cartesian form, but the components of the solution and flux vectors must be redefined. For a cylindrical coordinate solution, the vector of dependent variables Q is defined as:

$$Q = \begin{bmatrix} \rho \\ \rho v_x \\ \rho v_r \\ \rho v_\theta \\ \rho e_t \end{bmatrix} \quad (\text{A.36})$$

where the velocity components v_x , v_r , and v_θ are the absolute velocity components in the axial, radial, and circumferential coordinate directions, respectively (Figure A.2).

The flux vectors are expressed as:

$$F_{inv} = \begin{bmatrix} \rho v_x \\ \rho v_x^2 + p \\ \rho v_x v_r \\ \rho v_x w_{rel} \\ \rho v_x H_{total} \end{bmatrix}, \quad G_{inv} = \begin{bmatrix} \rho v_r \\ \rho v_x v_r \\ \rho v_r^2 + p \\ \rho v_r w_{rel} \\ \rho v_r H_{total} \end{bmatrix}, \quad H_{inv} = \begin{bmatrix} \rho v_\theta \\ \rho v_x v_\theta \\ \rho v_r v_\theta \\ \rho v_\theta w_{rel} + p \\ \rho w_{rel} H_{total} \end{bmatrix}, \quad (\text{A.37})$$

$$F_{vis} = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xr} \\ \tau_{x\theta} \\ q_x \end{bmatrix}, \quad G_{vis} = \begin{bmatrix} 0 \\ \tau_{rx} \\ \tau_{rr} \\ \tau_{r\theta} \\ q_r \end{bmatrix}, \quad H_{vis} = \begin{bmatrix} 0 \\ \tau_{\theta x} \\ \tau_{\theta r} \\ \tau_{\theta\theta} \\ q_\theta \end{bmatrix} \quad (\text{A.38})$$

and the cylindrical coordinate system source term becomes:

$$K = \begin{bmatrix} 0 \\ 0 \\ \frac{\rho v_\theta^2 + p}{r} - \frac{\tau_{\theta\theta}}{r} \\ -\frac{\rho v_r v_\theta}{r} + \frac{\tau_{r\theta}}{r} \\ 0 \end{bmatrix} \quad (\text{A.39})$$

The total enthalpy, H , is related to the total energy by:

$$H_{total} = e_t + \frac{p}{\rho} \quad (\text{A.40})$$

The viscous stress and heat flux terms may be expressed as:

$$\tau_{xx} = 2\mu \left(\frac{\partial v_x}{\partial x} \right) + \lambda_v \nabla \cdot \vec{V}, \quad (\text{A.41})$$

$$\tau_{xr} = \mu \left[\left(\frac{\partial v_r}{\partial x} \right) + \left(\frac{\partial v_x}{\partial r} \right) \right], \quad (\text{A.42})$$

$$\tau_{x\theta} = 2\mu \left[\left(\frac{\partial v_\theta}{\partial x} \right) + \left(\frac{1}{r} \frac{\partial v_x}{\partial \theta} \right) \right], \quad (\text{A.43})$$

$$\tau_{rr} = 2\mu \left(\frac{\partial v_r}{\partial r} \right) + \lambda_v \nabla \cdot \vec{V}, \quad (\text{A.44})$$

$$\tau_{r\theta} = \mu \left[\left(\frac{1}{r} \frac{\partial v_r}{\partial \theta} \right) + \left(r \frac{\partial v_\theta}{\partial r} \right) \right], \quad (\text{A.45})$$

$$\tau_{\theta\theta} = 2\mu \left(\frac{1}{r} \frac{\partial v_\theta}{\partial \theta} \right) + \lambda_v \nabla \cdot \vec{V} + 2\mu \frac{v_r}{r}, \quad (\text{A.46})$$

$$q_x = v_x \tau_{xx} + v_r \tau_{xr} + v_\theta \tau_{x\theta} + k \frac{\partial T}{\partial x}, \quad (\text{A.47})$$

$$q_r = v_x \tau_{rx} + v_r \tau_{rr} + v_\theta \tau_{r\theta} + k \frac{\partial T}{\partial r}, \quad (\text{A.48})$$

$$q_\theta = v_x \tau_{\theta x} + v_r \tau_{\theta r} + v_\theta \tau_{\theta\theta} + k \frac{\partial T}{\partial \theta} \quad (\text{A.49})$$

where μ is the first coefficient of viscosity, λ_v is the second coefficient of viscosity, and:

$$\nabla \cdot \vec{V} = \frac{\partial v_x}{\partial x} + \frac{\partial v_r}{\partial r} + \frac{1}{r} \frac{\partial v_\theta}{\partial \theta} + \frac{v_r}{r} \quad (\text{A.50})$$

The remaining viscous stress terms are defined through the identities:

$$\tau_{rx} = \tau_{xr}, \quad (\text{A.51})$$

$$\tau_{\theta r} = \tau_{r\theta}, \quad (\text{A.52})$$

$$\tau_{\theta x} = \tau_{x\theta} \quad (\text{A.53})$$

A.3 Fluid Properties

The primary working fluid is assumed to be air acting as a perfect gas, thus the ideal gas equation of state has been used. Fluid properties such as specific heats, specific heat ratio, and Prandtl number are assumed to be constant. The fluid viscosity is temperature dependent and is derived from the Sutherland (see e.g. [27]) formula:

$$\mu = C_1 \frac{(T)^{\frac{3}{2}}}{T + C_2} \quad (\text{A.54})$$

where for air the coefficients are specified as:

$$C_1 = 2.2710^{-8} \frac{\text{lbm}}{\text{ft} - \text{sec}} \quad C_2 = 198.72^\circ \text{R}$$

The so-called second coefficient of viscosity λ_v is fixed according to:

$$\lambda_v = -\frac{2}{3}\mu \quad (\text{A.55})$$

The thermal conductivity is determined from the viscosity and the definition of the Prandtl number as:

$$k = \frac{c_p \mu}{Pr} \quad (\text{A.56})$$

A.4 Numerical Formulation

The numerical formulation for the *ADPAC* code is provided in the subsections below.

A.4.1 Finite Volume Discretization

Integration of the three-dimensional differential form of the Navier-Stokes equations over a finite control volume yields an equation of the form:

$$\int \int \int \frac{\partial}{\partial t} (Q) dV + L_{inv}(Q) = L_{vis}(Q) + \int \int \int D dV \quad (\text{A.57})$$

where:

$$L_{inv}(Q) = \int \int_{dA} [F_{inv}dA_1 + G_{inv}dA_2 + H_{inv}dA_3] \quad (\text{A.58})$$

and:

$$L_{vis}(Q) = \int \int_{dA} [F_{vis}dA_1 + G_{vis}dA_2 + H_{vis}dA_3] \quad (\text{A.59})$$

The Gauss divergence theorem has been employed to convert several volume integrals to surface flux integrals, which simplifies the numerical evaluation of many terms (see e.g. [27]). The inviscid (convective) and viscous (diffusive) flux contributions are expressed separately by the operators L_{inv} and L_{vis} , respectively. The vector of dependent variables Q and other terms are described separately for both a Cartesian and a cylindrical coordinate system above.

The discrete numerical solution is developed from the integral governing equations derived in the previous sections by employing a finite volume solution procedure. This procedure closely follows the basic scheme described by Jameson [29]. In order to appreciate and utilize the features of the *ADPAC* solution system, the concept of a multiple-block grid system must be fully understood. It is expected that the reader possesses at least some understanding of the concepts of computational fluid dynamics (CFD), so the use of a numerical grid to discretize a flow domain should not be foreign. Many CFD analyses rely on a single structured ordering of grid points upon which the numerical solution is performed. Multiple-block grid systems are different only in that several structured grid systems are used in harmony to generate the numerical solution. The domain of interest is subdivided into one or more structured arrays of hexahedral cells. Each array of cells is referred to as a “block”, and the overall scheme is referred to as a multiple blocked mesh solver as a result of the ability to manage more than one block. This concept is illustrated graphically in two dimensions for the flow through a nozzle in Figure 2.1 along with the accompanying text.

The solution for each mesh block in a multiple-block grid is computed identically, and therefore the numerical approach is described for a single mesh block. In any given mesh block, the numerical grid is used to define a set of hexahedral cells, the vertices of which are defined by the eight surrounding mesh points. This construction is illustrated in Figure A.3.

The cell face surface area normal vector components dA_x , dA_y , and dA_z are calculated using the cross product of the diagonals defined by the four vertices of the given face, and the cell volume is determined by a procedure outlined by Hung and Kordulla [30] for generalized nonorthogonal cells. The integral relations expressed by the governing equations are determined for each cell by approximating the area-integrated convective and diffusive fluxes with a representative value along each cell face, and by approximating the volume-integrated terms with a representative cell volume weighted value. The discrete numerical approximation to the governing equation then becomes:

$$\begin{aligned} (\mathcal{V}) \frac{Q_{i,j,k}^{n+1} - Q_{i,j,k}^n}{\Delta t} &= (F_{inv}(Q)_{i+\frac{1}{2},j,k} - F_{inv}(Q)_{i-\frac{1}{2},j,k} \\ &+ G_{inv}(Q)_{i,j+\frac{1}{2},k} - G_{inv}(Q)_{i,j-\frac{1}{2},k} \end{aligned} \quad (\text{A.60})$$

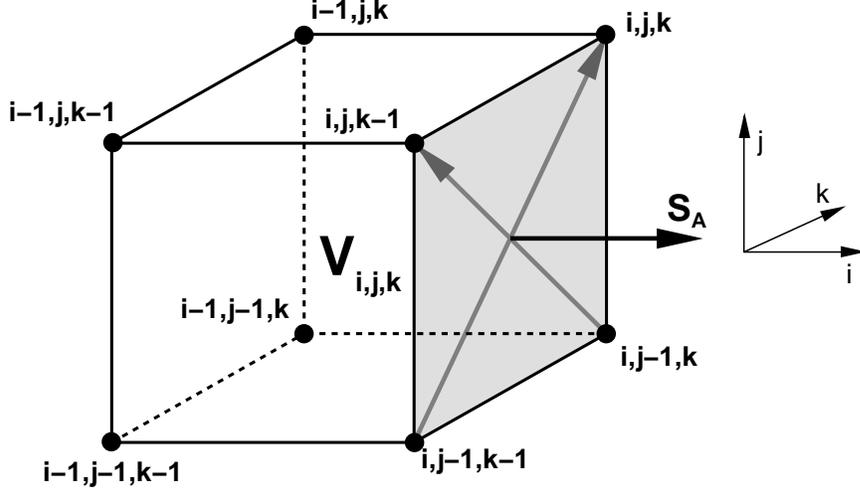


Figure A.3: Three-dimensional finite volume cell.

$$\begin{aligned}
& +H_{inv}(Q)_{i,j,k+\frac{1}{2}} - H_{inv}(Q)_{i,j,k-\frac{1}{2}} \\
& +F_{vis}(Q)_{i+\frac{1}{2},j,k} - F_{vis}(Q)_{i-\frac{1}{2},j,k} \\
& +G_{vis}(Q)_{i,j+\frac{1}{2},k} - G_{vis}(Q)_{i,j-\frac{1}{2},k} \\
& +H_{vis}(Q)_{i,j,k+\frac{1}{2}} - H_{vis}(Q)_{i,j,k-\frac{1}{2}} \\
& +(\mathcal{V})K + D_{i,j,k}(Q)
\end{aligned}$$

Following the algorithm defined by Jameson [29], it is convenient to store the flow variables as a representative value for the interior of each cell, and thus the scheme is referred to as cell-centered. Here, i, j, k represents the local cell indices in the structured cell-centered array, \mathcal{V} is the local cell volume, Δt is the calculation time interval, and $D_{i,j,k}$ is an artificial numerical dissipation function which is added to the governing equations to aid numerical stability, and to eliminate spurious numerical oscillations in the vicinity of flow discontinuities such as shock waves. Indicial expressions such as $i + \frac{1}{2}, j, k$ represents data evaluated at the cell face, or interface between two adjacent volumes. The discrete convective fluxes are constructed by using a representative value of the flow variables Q which is determined by an algebraic average of the values of Q in the cells lying on either side of the local cell face. A conceptual illustration of the finite-volume, cell centered data approach, and the subsequent convective flux evaluation process for a cell face are given on Figure A.4. Viscous stress terms and thermal conduction terms are constructed by applying a generalized coordinate transformation to the governing equations as follows:

$$\xi = \xi(x, y, z), \quad \eta = \eta(x, y, z), \quad \zeta = \zeta(x, y, z) \quad (\text{A.61})$$

The chain rule may then be used to expand the various derivatives in the viscous stresses as:

$$\frac{\partial}{\partial x} = \frac{\partial \xi}{\partial x} \frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial x} \frac{\partial}{\partial \eta} + \frac{\partial \zeta}{\partial x} \frac{\partial}{\partial \zeta}, \quad (\text{A.62})$$

$$\frac{\partial}{\partial y} = \frac{\partial \xi}{\partial y} \frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial y} \frac{\partial}{\partial \eta} + \frac{\partial \zeta}{\partial y} \frac{\partial}{\partial \zeta}, \quad (\text{A.63})$$

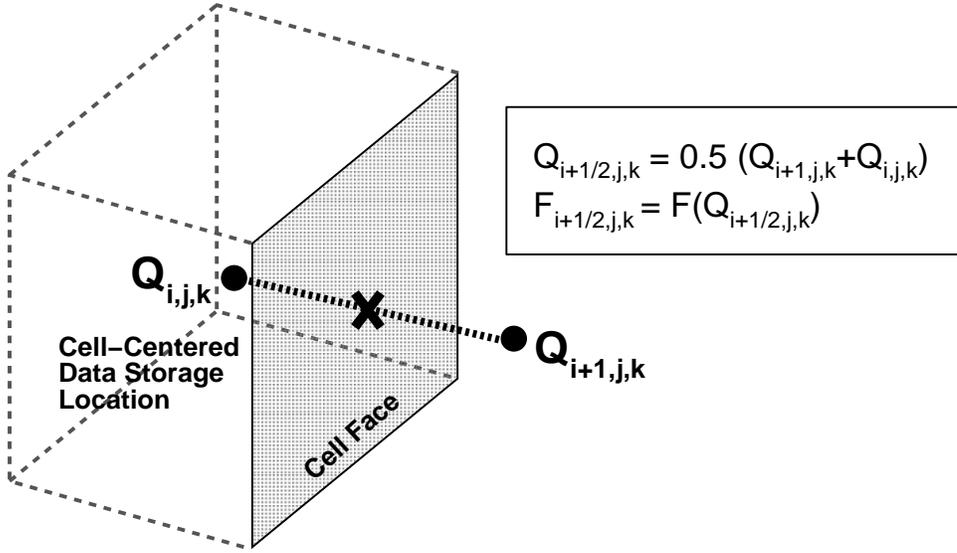


Figure A.4: ADPAC finite volume cell centered data configuration and convective flux evaluation process.

$$\frac{\partial}{\partial z} = \frac{\partial \xi}{\partial z} \frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial z} \frac{\partial}{\partial \eta} + \frac{\partial \zeta}{\partial z} \frac{\partial}{\partial \zeta}, \quad (\text{A.64})$$

The transformed derivatives may now be easily calculated by differencing the variables in computational space (i corresponds to the ξ direction, j corresponds to the η direction, and k corresponds to the ζ direction), and utilizing the appropriate identities for the metric differences (see e.g. [27]). This process is illustrated schematically in Figure A.5.

A.4.2 Runge-Kutta Time Integration

The time-stepping scheme used to advance the discrete numerical representation of the governing equations is a multistage Runge-Kutta integration. An m stage Runge-Kutta integration for the discretized equations is expressed as:

$$\begin{aligned} Q_1 &= Q^n - \alpha_1 \Delta t [L(Q^n) + D(Q^n)], \\ Q_2 &= Q^n - \alpha_2 \Delta t [L(Q_1) + D(Q^n)], \\ Q_3 &= Q^n - \alpha_3 \Delta t [L(Q_2) + D(Q^n)], \\ Q_4 &= Q^n - \alpha_4 \Delta t [L(Q_3) + D(Q^n)], \\ &\dots \\ &\dots \\ Q_m &= Q^n - \alpha_m \Delta t [L(Q_{m-1}) + D(Q^n)], \\ Q^{n+1} &= Q_m \end{aligned} \quad (\text{A.65})$$

where:

$$L(Q) = L_{inv}(Q) - L_{vis}(Q) \quad (\text{A.66})$$

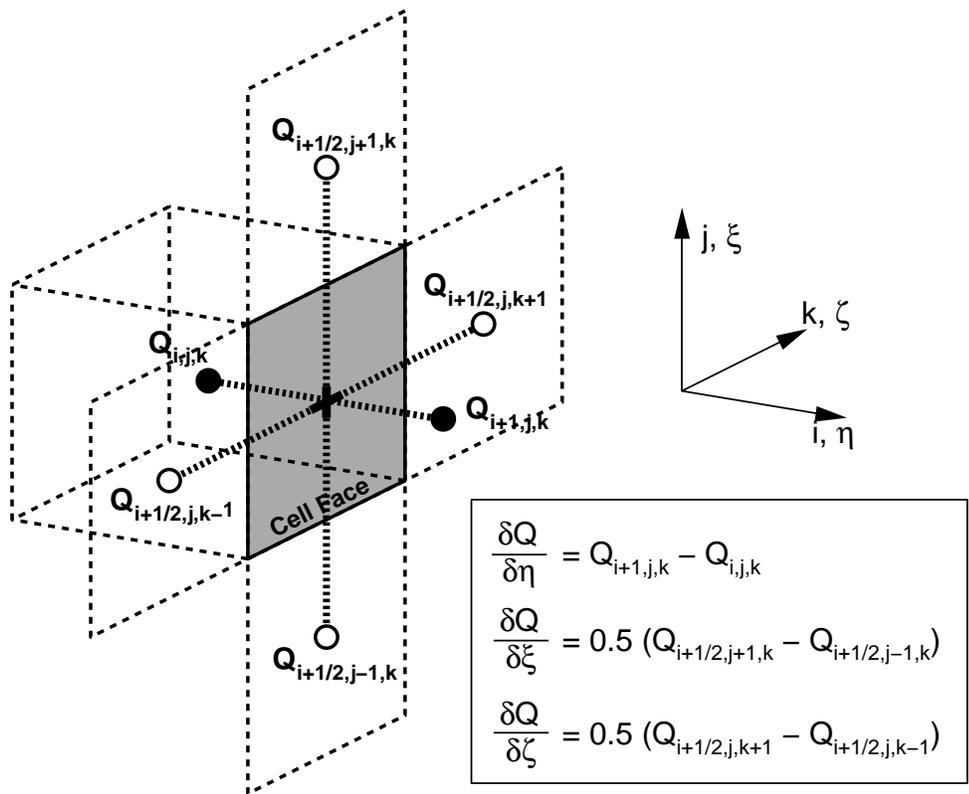


Figure A.5: ADPAC finite volume cell centered data configuration and diffusive flux evaluation process.

For simplicity, viscous flux contributions to the discretized equations are only calculated for the first stage, and the values are frozen for the remaining stages. This reduces the overall computational effort and does not appear to significantly alter the solution. It is also generally not necessary to recompute the added numerical dissipation terms during each stage. Three different multistage Runge-Kutta schemes (2 four-stage schemes, and 1 five-stage scheme) are available in the *ADPAC* code.

The coefficients for the four stage Runge-Kutta time-marching scheme employed in this study are listed below:

$$\alpha_1 = \frac{1}{8}, \quad \alpha_2 = \frac{1}{4}, \quad \alpha_3 = \frac{1}{2}, \quad \alpha_4 = 1 \quad (\text{A.67})$$

A linear stability analysis of the four stage Runge-Kutta time-stepping scheme utilized during this study indicate that the scheme is stable for all calculation time increments δt which satisfy the stability criteria $CFL \leq 2\sqrt{2}$. Based on convection constraints alone, the *CFL* number may be defined in a one-dimensional manner as:

$$CFL = \frac{\Delta t}{\frac{|v_x|+a}{\Delta x}} \quad (\text{A.68})$$

In practice, the calculation time interval must also include restrictions resulting from diffusion phenomena. The time step used in the numerical calculation results from both convective and diffusive considerations and is calculated as:

$$\Delta t = CFL \left(\frac{1.0}{\lambda_i + \lambda_j + \lambda_k + \nu_i + \nu_j + \nu_k} \right) \quad (\text{A.69})$$

where the convective and diffusive coordinate wave speeds (λ and ν , respectively) are defined as:

$$\lambda_i = \mathcal{V}/(\vec{V} \cdot \vec{S}_i + a) \quad (\text{A.70})$$

$$\nu_i = \frac{\rho(\mathcal{V})^2}{C_{\Delta t}(S^2)\mu} \quad (\text{A.71})$$

The factor $C_{\Delta t}$ is a “safety factor” of sorts, which must be imposed as a result of the limitations of the linear stability constraints for a set of equations which are truly nonlinear. This factor was determined through numerical experimentation and normally ranges from 2.5-7.5.

For steady flow calculations, an acceleration technique known as local time stepping is used to enhance convergence to the steady-state solution. Local time stepping utilizes the maximum allowable time increment at each point during the course of the solution. While this destroys the physical nature of the transient solution, the steady-state solution is unaffected and can be obtained in fewer iterations of the time-stepping scheme. For unsteady flow calculations, of course, a uniform value of the time step Δt must be used at every grid point to maintain the time-accuracy of the solution. Other convergence enhancements such as implicit residual smoothing and multi-grid (described in later sections) are also applied for steady flow calculations.

A.4.3 Dissipation Function

In order to prevent odd-even decoupling of the numerical solution, non-physical oscillations near shock waves, and to obtain rapid convergence for steady state solutions, artificial dissipative terms are added to the discrete numerical representation of the governing equations. The added dissipation model is based on the combined works of Jameson et al. [29], Martinelli [31], and Swanson et al. [33]. A blend of fourth and second differences is used to provide a third order background dissipation in smooth flow regions and first order dissipation near discontinuities. The discrete equation dissipative function is given by:

$$D_{i,j,k}(Q) = (D_i^2 - D_i^4 + D_j^2 - D_j^4 + D_k^2 - D_k^4)Q_{i,j,k} \quad (\text{A.72})$$

The second and fourth order dissipation operators are determined by:

$$D_\xi^2 Q_{i,j,k} = \nabla_\xi((\lambda_\xi)_{i+\frac{1}{2},j,k}) \Delta_\xi Q_{i,j,k} \quad (\text{A.73})$$

$$D_\xi^4 Q_{i,j,k} = \nabla_\xi((\lambda_\xi)_{i+\frac{1}{2},j,k}) \Delta_\xi \nabla_\xi \Delta_\xi Q_{i,j,k} \quad (\text{A.74})$$

where Δ_ξ and ∇_ξ are forward and backward difference operators in the ξ direction. In order to avoid excessively large levels of dissipation for cells with large aspect ratios, and to maintain the damping properties of the scheme, a variable scaling of the dissipative terms is employed which is an extension of the two dimensional scheme given by Martinelli [31]. The scaling factor is defined as a function of the spectral radius of the Jacobian matrices associated with the ξ , η , and ζ directions and provides a scaling mechanism for varying cell aspect ratios through the following scheme:

$$(\lambda_\xi)_{i+\frac{1}{2},j,k} = (\lambda_\xi)_{i+\frac{1}{2},j,k} \Phi_{i+\frac{1}{2},j,k} \quad (\text{A.75})$$

The function Φ controls the relative importance of dissipation in the three coordinate directions as:

$$\Phi_{i+\frac{1}{2},j,k} = 1 + \max \left(\left(\frac{(\lambda_\eta)_{i+\frac{1}{2},j,k}}{(\lambda_\xi)_{i+\frac{1}{2},j,k}} \right)^\alpha, \left(\frac{(\lambda_\zeta)_{i+\frac{1}{2},j,k}}{(\lambda_\xi)_{i+\frac{1}{2},j,k}} \right)^\alpha \right) \quad (\text{A.76})$$

The directional eigenvalue scaling functions are defined by:

$$(\lambda_\xi)_{i+\frac{1}{2},j,k} = U_{i+\frac{1}{2},j,k} (S_\xi)_{i+\frac{1}{2},j,k} + c(S_\xi)_{i+\frac{1}{2},j,k} \quad (\text{A.77})$$

$$(\lambda_\eta)_{i+\frac{1}{2},j,k} = U_{i+\frac{1}{2},j,k} (S_\eta)_{i+\frac{1}{2},j,k} + c(S_\eta)_{i+\frac{1}{2},j,k} \quad (\text{A.78})$$

$$(\lambda_\zeta)_{i+\frac{1}{2},j,k} = U_{i+\frac{1}{2},j,k} (S_\zeta)_{i+\frac{1}{2},j,k} + c(S_\zeta)_{i+\frac{1}{2},j,k} \quad (\text{A.79})$$

The use of the maximum function in the definition of Φ is important for grids where λ_η/λ_ξ and $\lambda_\zeta/\lambda_\xi$ are very large and of the same order of magnitude. In this case, if these ratios are summed rather than taking the maximum, the dissipation can become too large, resulting in degraded solution accuracy and poor convergence. Because three-dimensional solution grids tend to exhibit large variations in the cell aspect ratio, there is less freedom in the choice of the parameter α for this scheme, and a value of 0.5 was found to provide a robust scheme.

The coefficients in the dissipation operator use the solution pressure as a sensor for the presence of shock waves in the solution and are defined as:

$$\epsilon_{i+\frac{1}{2},j,k}^2 = \kappa^2 \max(\nu_{i-1,j,k}, \nu_{i,j,k}, \nu_{i+1,j,k}, \nu_{i+2,j,k}) \quad (\text{A.80})$$

$$\nu_{i,j,k} = \frac{|(p_{i-1,j,k} - 2p_{i,j,k} + p_{i+1,j,k})|}{(p_{i-1,j,k} + 2p_{i,j,k} + p_{i+1,j,k})} \quad (\text{A.81})$$

$$\epsilon_{i+\frac{1}{2},j,k}^4 = \max(0, \kappa^4 - \epsilon_{i+\frac{1}{2},j,k}^2) \quad (\text{A.82})$$

where κ^2, κ^4 are user-defined constants. Typical values for these constants are:

$$\kappa^2 = \frac{1}{2} \quad \kappa^4 = \frac{1}{64} \quad (\text{A.83})$$

The dissipation operators in the η and ζ directions are defined in a similar manner.

A.4.4 Implicit Residual Smoothing

The stability range of the basic time-stepping scheme can be extended using implicit smoothing of the residuals. This technique was described by Hollanders et al. [32] for the Lax-Wendroff scheme and later developed by Jameson [29] for the Runge-Kutta scheme. Since an unsteady flow calculation for a given geometry and grid is likely to be computationally more expensive than a similar steady flow calculation, it would be advantageous to utilize this acceleration technique for time-dependent flow calculations as well. In an analysis of two dimensional unsteady flows, Jorgensen and Chima [5] demonstrated that a variant of the implicit residual smoothing technique could be incorporated into a time-accurate explicit method to permit the use of larger calculation time increments without adversely affecting the results of the unsteady calculation. The implementation of this residual smoothing scheme reduced the CPU time for their calculation by a factor of five. This so-called time-accurate implicit residual smoothing operator was then also demonstrated by Rao and Delaney [4] for a similar two-dimensional unsteady calculation and by Hall, et al. [24],[1] for several three-dimensional time-dependent flows. Although this “time-accurate” implicit residual smoothing scheme is not developed theoretically to accurately provide the unsteady solution, it can be demonstrated that errors introduced through this residual smoothing process are very local in nature, and are generally not greater than the discretization error.

The standard implicit residual smoothing operator can be written as:

$$(1 - \epsilon \Delta \nabla) R_m^* = R_m \quad (\text{A.84})$$

To simplify the numerical implementation, this standard operator is traditionally approximately factored into the following coordinate specific form:

$$(1 - \epsilon_\xi \Delta_\xi \nabla_\xi)(1 - \epsilon_\eta \Delta_\eta \nabla_\eta)(1 - \epsilon_\zeta \Delta_\zeta \nabla_\zeta) \bar{R}_m = R_m \quad (\text{A.85})$$

where the residual R_m is defined as:

$$R_m = \alpha_m \frac{\Delta t}{V} (Q_m - D_m), \quad m = 1, mstages \quad (\text{A.86})$$

for each of the m stages in the Runge-Kutta multistage scheme. Here Q_m is the sum of the convective and diffusive terms, D_m the total dissipation at stage m , and \bar{R}_m the final (smoothed) residual at stage m .

The smoothing reduction is applied sequentially in each coordinate direction as:

$$\begin{aligned}
R_m^* &= (1 - \epsilon_\xi \Delta_\xi \nabla_\xi)^{-1} R_m \\
R_m^{**} &= (1 - \epsilon_\eta \Delta_\eta \nabla_\eta)^{-1} R_m^* \\
R_m^{***} &= (1 - \epsilon_\zeta \Delta_\zeta \nabla_\zeta)^{-1} R_m^{**} \\
\bar{R}_m &= R_m^{***}
\end{aligned} \tag{A.87}$$

where each of the first three steps above requires the inversion of a scalar tridiagonal matrix. In general, it is desirable to apply the smoothing at each stage of the Runge-Kutta time-marching procedure.

The use of constant coefficients (ϵ) in the implicit treatment has proven to be useful, even for meshes with high aspect ratio cells, provided additional support such as enthalpy damping (see [29]) is introduced. Unfortunately, the use of enthalpy damping, which assumes a constant total enthalpy throughout the flowfield, cannot be used for an unsteady flow, and many steady flows where the total enthalpy may vary. It has been shown that the need for enthalpy damping can be eliminated by using variable coefficients in the implicit treatment which account for the variation of the cell aspect ratio. Martinelli [31] derived a functional form for the variable coefficients for two-dimensional flows which are functions of characteristic wave speeds. In this study, the three-dimensional extension described by Radespiel et al. [33] is utilized, and is expressed as:

$$\epsilon_\xi = \max \left(0, \frac{1}{4} \left[\frac{CFL}{CFL_{max}} \frac{1 + \max(r_{\eta\xi}^\alpha r_{\zeta\xi}^\alpha)}{1 + \max(r_{\eta\xi}^\alpha r_{\zeta\xi}^\alpha)} \right]^2 - 1 \right) \tag{A.88}$$

$$\epsilon_\eta = \max \left(0, \frac{1}{4} \left[\frac{CFL}{CFL_{max}} \frac{1 + \max(r_{\xi\eta}^\alpha r_{\zeta\eta}^\alpha)}{1 + \max(r_{\xi\eta}^\alpha r_{\zeta\eta}^\alpha)} \right]^2 - 1 \right) \tag{A.89}$$

$$\epsilon_\zeta = \max \left(0, \frac{1}{4} \left[\frac{CFL}{CFL_{max}} \frac{1 + \max(r_{\eta\zeta}^\alpha r_{\xi\zeta}^\alpha)}{1 + \max(r_{\eta\zeta}^\alpha r_{\xi\zeta}^\alpha)} \right]^2 - 1 \right) \tag{A.90}$$

CFL represents the local value of the CFL number based on the calculation time increment Δt , and CFL_{max} represents the maximum stable value of the CFL number permitted by the unmodified scheme (normally, in practice, this is chosen as 2.5 for a four stage scheme and 3.5 for a five stage scheme, although linear stability analysis suggests that $2\sqrt{2}$, and 3.75 are the theoretical limits for the four and five stage schemes, respectively). From this formulation it is obvious then that the residual smoothing operator is only applied in those regions where the local CFL number exceeds the stability-limited value. In this approach, the residual operator coefficient becomes zero at points where the local CFL number is less than that required by stability, and the influence of the smoothing is only locally applied to those regions exceeding the stability limit. Practical experience involving unsteady flow calculations suggests that for a constant time increment, the majority of the flowfield utilizes CFL numbers less than the stability-limited value to maintain a reasonable level of accuracy. Local smoothing is therefore typically required only in regions of small grid spacing, where the

stability-limited time step is very small. Numerical tests both with and without the time-accurate implicit residual smoothing operator for the flows of interest in this study were found to produce essentially identical results, while the time-accurate residual smoothing resulted in a decrease in CPU time by a factor of 2-3. In practice, the actual limit on the calculation CFL number were determined to be roughly twice the values specified for CFL_{max} , above.

A.4.5 Multi-grid Convergence Acceleration

Multi-grid (not to be confused with a multiple-blocked grid!) is a numerical solution technique which attempts to accelerate the convergence of an iterative process (such as a steady flow prediction using a time-marching scheme) by computing corrections to the solution on coarser meshes and propagating these changes to the fine mesh through interpolation. This operation may be recursively applied to several coarsenings of the original mesh to effectively enhance the overall convergence. In the present multi-grid application, coarse meshes are derived from the preceding finer mesh by eliminating every other mesh line in each coordinate direction as shown in Figure 2.6. As a result, the number of multi-grid levels (coarse mesh divisions) is controlled by the mesh size, and, in the case of the *ADPAC* code, also by the indices of the embedded mesh boundaries (such as blade leading and trailing edges, etc.) (see Figure 2.6). These restrictions suggest that mesh blocks should be constructed such that the internal boundaries and overall size coincide with numbers which are compatible with the multi-grid solution procedure; the mesh size should be 1 greater than any number which can be divided by 2 several times and remain whole numbers (i.e., 9, 17, 33, 65).

The multi-grid procedure is applied in a V-cycle as shown in Figure A.6, whereby the fine mesh solution is initially “injected” into the next coarser mesh, the appropriate forcing functions are then calculated based on the differences between the calculated coarse mesh residual and the residual which results from a summation of the fine mesh residuals for the coarse mesh cell, and the solution is advanced on the coarse mesh. This sequence is repeated on each successively coarser mesh until the coarsest mesh is reached. At this point, the correction to the solution ($Q_{i,j,k}^{n+1} - Q_{i,j,k}^n$) is interpolated to the next finer mesh, a new solution is defined on that mesh, and the interpolation of corrections is applied sequentially until the finest mesh is reached. Following a concept suggested by Swanson et al. [33], it is sometimes desirable to smooth the final corrections on the finest mesh to reduce the effects of oscillations induced by the interpolation process. A constant coefficient implementation of the implicit residual smoothing scheme described in Section 3.5 is used for this purpose. The value of the smoothing constant is normally taken to be 0.2.

A second multi-grid concept which should be discussed is the so-called “full” multi-grid startup procedure. The “full” multi-grid method is used to initialize a solution by first computing the flow on a coarse mesh, performing several time-marching iterations on that mesh (which, by the way could be multi-grid iterations if successively coarser meshes are available), and then interpolating the solution at that point to the next finer mesh, and repeating the entire process until the finest mesh level is reached. The intent here is to generate a reasonably approximate solution on the coarser meshes before undergoing the expense of the fine mesh multi-grid cycles. Again, the “full” multi-grid

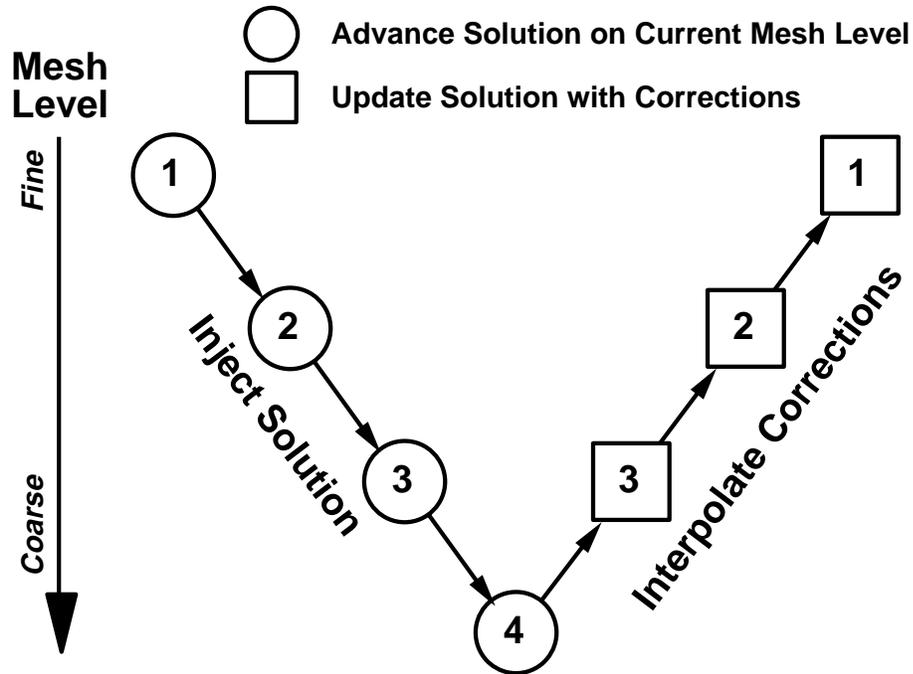


Figure A.6: Multi-grid V-cycle strategy.

technique only applies to starting up a solution.

A.4.6 Implicit Time-Marching Algorithm Procedure

The development of an implicit time-marching strategy for the *ADPAC* code was initiated during this study. This effort was directed at improving the computational efficiency of the *ADPAC* code for lengthy time-dependent calculations, particularly for viscous flows, where the restricted time step of the explicit time-marching algorithm due to highly clustered meshes is prohibitively expensive. The implicit algorithm type selected was chosen to take advantage of the multi-grid solution capabilities of the explicit *ADPAC* time marching algorithm, and the various steady state convergence acceleration techniques (implicit residual smoothing, local time stepping, etc.) which have been incorporated into the code.

The implicit algorithm is best explained through derivation. The original explicit steady state iterative numerical algorithm in the *ADPAC* code ultimately solves an equation of the form:

$$\frac{\partial Q}{\partial t} = -R(Q) \quad (\text{A.91})$$

where Q is the vector of dependent variables, t is time, and $R(Q)$ is the residual which includes convective, diffusive, and artificial dissipation fluxes. The solution is typically advanced in time t until the residual approaches zero, or simply $\partial Q/\partial t = 0$ which implies a time-independent (steady state) solution. The modified solution scheme introduces a fictitious time τ , and solves an equation of the form:

$$\frac{\partial Q}{\partial \tau} = \frac{\partial Q}{\partial t} + R(Q) = R^*(Q) \quad (\text{A.92})$$

Now the solution for driving the new residual $R^*(Q)$ to zero can be advanced by marching in τ using all of the previously developed steady state convergence acceleration techniques, and the final solution satisfies the equation:

$$\frac{\partial Q}{\partial t} + R(Q) = 0 \quad (\text{A.93})$$

which is the desired time-dependent solution. This approach follows the work of Jameson [29] and the more recent applications of Arnone et al. [34]. Derivatives with respect to the real time t are discretized using either a 2 point or a 3 point backward formula which results in an implicit scheme which is second order accurate in time. The discretized equation solved at each time level therefore becomes:

$$\frac{\partial Q}{\partial \tau} = \frac{3Q^{n+1} - 4Q^n + Q^{n-1}}{2\Delta t} + R(Q^{n+1}) = R^*(Q^{n+1}) \quad (\text{A.94})$$

where the subscript n is associated with real time. Between each real time step, then, the solution is advanced multiple iterations in the non-physical time to satisfy the time-accurate equations. The pseudo time numerical approach was recently demonstrated for time-dependent flows in turbomachinery geometries by Arnone et al. [35].

The time discretization described above is fully implicit; however, when solved by marching in τ , stability problems can occur when the time increment in the pseudo time variable $\Delta\tau$ exceeds the physical time step Δt . Linear stability analysis (see e.g. [34]) indicates that the pseudo time increment $\Delta\tau$ must be less than $2/3CFL^*\Delta t$ where CFL^* is the ratio of the local CFL number to the maximum CFL number of the explicit time-marching scheme.

Numerical experiments for the algorithm with the physical time derivative term treated in an explicit manner indicated that the algorithm exhibited a physical time step dependent divergent behavior for many problems. This formulation of the algorithm closely followed the development proposed by Jameson [36]. No indication of such behavior was identified in Jameson's [36] paper. Arnone [34] identified a time step modifier which sought to circumvent the unstable region by lowering the pseudo time increment in relation to the physical time step. This modification was originally included in the *ADPAC* formulation, but did not completely prohibit the instability. Further study of the problem, and, in particular, the assistance of researchers at the NASA-Langley Research Center, have identified the explicit treatment of the physical time derivative term as the source of the conditional stability. In a recent paper by Melson, Sanetrik, and Atkins [37], the stability characteristics of the implicit iterative algorithm with both explicit and implicit treatments of the physical time derivative term were analyzed. The implicit treatment of this term was found to be unconditionally stable, while the explicit treatment of the algorithm was conditionally stable based on the value of $\Delta\tau/\Delta t$, where $\Delta\tau$ is the pseudo time derivative time step, while Δt is the the physical time derivative time step. Small values of $\Delta\tau/\Delta t$ (corresponding to low CFL numbers) push the algorithm towards an unstable operation. Jameson [36] suggests that CFL numbers of 200 or greater be used for the explicit treatment, which is consistent with low values of $\Delta\tau/\Delta t$, or improved stability. The *ADPAC* code was modified to utilize an implicit treatment of the time derivative term. The implicit treatment of the physical time derivative greatly enhances the effectiveness of the implicit flow solver for a wider range of problems.

A further improvement of the implicit algorithm was discovered by Melson et al. This modification is based on the realization that the term Q^{n+1} actually appears on both sides of the implicit time marching equation described above. It should be possible, therefore, to collect these terms and modify the time marching equation in a more “fully implicit” manner (in terms of how the updated Q^{n+1} is calculated). Several variations on this technique are described below.

The development given here follows the derivation described by Melson et al. Suppose we seek to solve an equation of the form:

$$\frac{\partial Q}{\partial t} = L(Q) \quad (\text{A.95})$$

where $L(Q)$ is a collection of fluxes and source terms similar to that given by the Runge-Kutta time marching scheme. The physical time derivative term is approximated by a discrete operator of the form:

$$\frac{\partial Q}{\partial t} = \frac{1}{\Delta t} \left(\sum_{m=0}^M a_m Q^{n+1-m} \right) = \frac{1}{\Delta t} [a_0 Q^{n+1} + E(Q^n, Q^{n-1}, \dots, Q^{n+1-m})] \quad (\text{A.96})$$

Here $E(Q)$ represents the portion of the discrete approximation which involves values of the dependent variable Q evaluated from previous time steps.

If we interpret the explicit time marching scheme in the following form:

$$Q^{n+1} = Q^n + \Delta t R(Q) \quad (\text{A.97})$$

where $R(Q)$ is the summation of convective, diffusive, and dissipative fluxes, and internal source terms. For the implicit algorithm, the time step τ becomes a pseudo time step used in an iterative fashion to construct the time dependent solution according to the physical time step ΔT . In this case, the algorithm becomes:

$$Q^{n+1} = Q^n + \Delta \tau \left(\frac{\partial Q}{\partial t} + R(Q) \right) \quad (\text{A.98})$$

If we approximate the physical time derivative term with the discrete operator described above, we get:

$$Q^{n+1} = Q^n + \Delta \tau \left(\frac{1}{\Delta t} [a_0 Q^{n+1} + E(Q)] + R(Q) \right) \quad (\text{A.99})$$

and can consequently develop a new implicit-like equation for the term Q^{n+1} as:

$$Q^{n+1} = \frac{\left(Q^n + \Delta \tau \left(\frac{1}{\Delta t} [E(Q)] + R(Q) \right) \right)}{1 - \Delta \tau \left(\frac{1}{\Delta T} [a_0] \right)} \quad (\text{A.100})$$

A number of algorithms can now be developed based on the choice of discrete approximation to the physical time derivative term. Based on the work of Melson et al., this study was limited to approximations of both first and second order. Based on linear stability analysis, the algorithm can be made to be unconditionally stable for either first or second order accurate representations of the physical time derivative term. Several higher order approximations were found by Melson et al. to be conditionally stable, and require the added burden of storing more than 3 time levels of data to complete the algorithm.

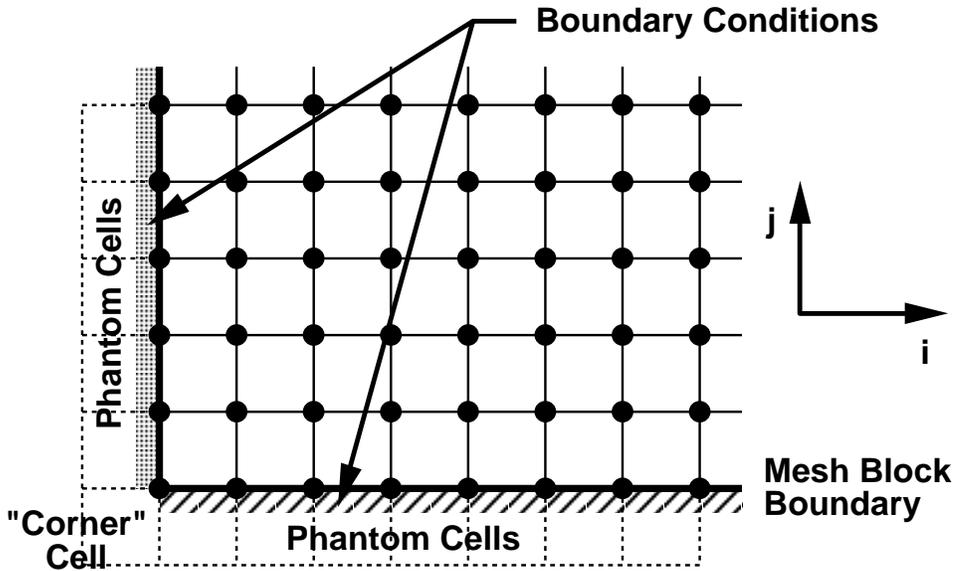


Figure A.7: 2-D mesh block phantom cell representation.

In every case tested, the modified implicit scheme described by Melson was more robust than the direct implicit scheme described by Arnone. Unfortunately, instabilities were found to occur for both algorithms under conditions which were believed to be in the unconditionally stable regime for each algorithm. It appears that a more thorough investigation of the stability characteristics of the iterative implicit algorithm should be performed to isolate the causes of the intermittent unstable behavior.

A.5 Boundary Conditions

In this section, the various types of boundary conditions utilized in the *ADPAC* analysis are described in general. Before describing the individual boundary conditions, it may be useful to describe how the boundary conditions are imposed in the discrete numerical solution. Finite volume solution algorithms such as the *ADPAC* program typically employ the concept of a phantom cell to impose boundary conditions on the external faces of a particular mesh block. This concept is illustrated graphically for a 2-D mesh representation in Figure A.7. Boundary condition specifications control the flow variables for the phantom cells adjacent to the mesh block boundary. "Corner" phantom cells cannot be controlled through boundary conditions, but must be updated to accurately compute grid point averaged values.

Some comments concerning the specifics of numerically treating block boundaries are in order at this point. Artificial damping is treated at inflow/outflow block boundaries by prescribing zero dissipation flux along the boundary to preserve the globally conservative nature of the solution. For inter-block communication, dissipative fluxes are also communicated between blocks to prevent inadequate numerical damping at inner block boundaries. Implicit residual smoothing is applied at all block boundaries by imposing a zero residual gradient (i.e. $(dR/dz) = 0.0$) condition at the boundary.

A phantom cell is a fictitious neighboring cell located outside the extent of a mesh which is utilized in the application of boundary conditions on the outer boundaries of a mesh block. Since flow variables cannot be directly specified at a mesh surface in a finite volume solution (the flow variables are calculated and stored at cell centers), the boundary data specified in the phantom cells are utilized to control the flux condition at the cell faces of the outer boundary of the mesh block, and, in turn, satisfy a particular boundary condition. All *ADPAC* boundary condition specifications provide data values for phantom cells to implement a particular mathematical boundary condition *on the mesh*. Another advantage of the phantom cell approach is that it permits unmodified application of the interior point scheme at near boundary cells.

A.5.1 Standard Inflow/Outflow Boundary Procedures

Inflow and exit boundary conditions are applied numerically using characteristic theory. A one-dimensional isentropic system of equations is utilized to derive the following characteristic equations at an inflow/outflow boundary:

$$\frac{\partial C^-}{\partial t} - (v_n - a) \frac{\partial C^-}{\partial n} = 0, \quad (\text{A.101})$$

$$\frac{\partial C^+}{\partial t} + (v_n + a) \frac{\partial C^+}{\partial n} = 0 \quad (\text{A.102})$$

where:

$$C^- = v_n - \frac{2a}{\gamma - 1}, \quad C^+ = v_n + \frac{2a}{\gamma - 1} \quad (\text{A.103})$$

Numerically, the equations are solved in a locally orthogonal coordinate system which is normal to the cell face of interest (indicated by the subscript and coordinate n). The procedure is essentially then a reference plane method of characteristics based on the Reimann invariants C^- and C^+ .

For subsonic normal inflow, the upstream running invariant C^- is extrapolated to the inlet, and along with the equation of state, specified total pressure, total temperature, and flow angles the flow variables at the boundary may be determined. For turbomachinery flow calculations, the flow angles are representative of the spanwise flow and the pitchwise (blade-to-blade) flow.

Outflow boundaries require a specification of the exit static pressure. In this case, the downstream running invariant C^+ is used to update the phantom cells at the exit boundary. Velocity components parallel to the cell face are extrapolated to the phantom cell from the neighboring interior cells.

It should be mentioned that all of the characteristic boundary schemes utilize a local rotated coordinate system which is normal (hence the subscript n) to the bounding cell face.

A.5.2 Solid Surface Boundary Procedures

All inviscid solid surface must satisfy the condition of no convective flux through the boundary (an impermeable surface). Mathematically, this is expressed as:

$$\vec{V} \cdot \vec{n} = 0 \quad (\text{A.104})$$

The phantom cell velocity components are thus constructed to ensure that the cell face average velocities used in the convective flux calculation satisfy the no throughflow boundary specification *at the bounding surface*. A simplified form of the normal momentum equation is used to update the phantom cell pressure as:

$$\frac{\partial p}{\partial n} = 0 \quad (\text{A.105})$$

It should be noted that this condition is theoretically oversimplified, but our experience using more complicated forms of the normal momentum equation indicate that numerically the results are quite accurate.

All viscous solid surfaces must satisfy the no slip boundary condition for viscous flows:

$$\vec{V}_{rel} = 0 \quad (\text{A.106})$$

where \vec{V}_{rel} is the relative flow velocity, $\vec{V} - r\omega$. No convective flux through the boundary (an impermeable surface) is permitted. The phantom cell velocity components are thus constructed to ensure that the cell face average velocities used in the convective flux calculation are identically zero. The phantom cell pressure is simply extrapolated based on the boundary layer flow concept $dp/dn = 0$. The phantom cell density or temperature is imposed by assuming either an adiabatic surface $dT/dn = 0$ or a specified surface temperature, which suggests that the phantom cell temperature must be properly constructed to satisfy the appropriate average temperature along the surface.

A.5.3 Inter-block Communication Boundary Procedures

For the multiple-block scheme, the solution is performed on a single grid block at a time. Special boundary conditions along block boundaries are therefore required to provide some transport of information between blocks. This transport may be accomplished through one of four types of procedures in the *ADPAC* code. Each procedure applies to a different type of mesh construction and flow environment, and details of each approach are given in Reference [1].

For neighboring mesh blocks which have coincident mesh points along the interface separating the two blocks, a simple direct specification of the phantom cell data based on the near boundary cell data from the neighboring block has been used successfully (**PATCH** boundary condition). Each phantom cell in the block of interest has a direct correspondence with a near boundary cell in the neighboring mesh block, and the block coupling is achieved numerically by simply assigning the value of the corresponding cell in the neighboring block to the phantom cell of the block of interest. This procedure essentially duplicates the interior point solution scheme for the near boundary cells, and uniformly enforces the conservation principles implied by the governing equations. Other

boundary conditions related to inter-block communication for endwall treatment flow calculations are described in Chapter 3.

A.5.4 Non-Reflecting Inflow/Outflow Boundary Condition Procedures

A perplexing aspect of the numerical simulation of turbomachinery flowfields is the requirement of specifying inflow/outflow boundary conditions for a limited computational domain for machinery which is operating in an essentially unlimited environment. For the purpose of analyzing these complex flows, the objective was to develop a solution procedure which satisfies three constraints. First, the boundary procedure must maintain a physically consistent far field flow condition which may be specified by the user. Second, the solution should be insensitive to the relative position of the computational inlet and exit boundaries. Third, the boundary conditions should be constructed in a manner which does not introduce spurious, non-physical reflections of traveling waves into the numerical solution.

Theoretical mathematical foundations for “non-reflecting” boundary conditions for initial value problems can be found in many references ([38], [39] for example). A number of so-called non-reflecting boundary condition procedures have been developed specifically for turbomachinery flow applications have been presented by Erdos et al. [40], among others.

The non-reflecting boundary condition procedure used in *ADPAC* follows the general procedure developed by Giles [41], which was later expanded and applied to 3-D time-dependent turbomachinery flow predictions by Saxer [42]. Reader interested in the details of the implementation of these boundary conditions into *ADPAC* should refer to Reference [2].

A.6 Turbulence Models

As a result of computer limitations regarding storage and execution speed, the effects of turbulence are introduced through an appropriate turbulence model and solutions are performed on a numerical grid designed to capture the macroscopic (rather than the microscopic) behavior of the flow. The effects of turbulence are introduced into the numerical scheme by utilizing the Boussinesq approximation [28]:

$$\tau_t = -\overline{\rho u'v'} = \mu_t \frac{\partial \bar{u}}{\partial y} \quad (\text{A.107})$$

where μ_t is the eddy viscosity, resulting in an effective calculation viscosity defined as:

$$\mu_{eff} = \mu_{lam} + \mu_t \quad (\text{A.108})$$

The simulation is therefore performed using an effective viscosity which combines the effects of the physical (laminar) viscosity and the effects of turbulence through the turbulence model and the turbulent viscosity μ_t . The turbulent flow thermal conductivity term is also treated as the combination of a laminar and turbulent quantity as:

$$k_{eff} = k_{lam} + k_t \quad (\text{A.109})$$

For turbulent flows, the turbulent thermal conductivity k_t is determined from a turbulent Prandtl number Pr_t such that:

$$Pr_t = \frac{c_p \mu_t}{k_t} \quad (\text{A.110})$$

The turbulent Prandtl number is normally chosen to have a value of 0.9.

The first type of model available in *ADPAC* is referred to as an algebraic turbulence model due to the algebraic nature by which the turbulent viscosity is calculated. Algebraic models are generally the simplest models available for computational aerodynamic analysis, and are “tuned” based on correlations with flat plate turbulent boundary layers. Unfortunately, the simplicity of the modeling approach limits the useful applicability of the model to flows which consist primarily of well behaved (non-separated) wall bounded shear layers. To overcome this limitation, an one-equation turbulence model was added to *ADPAC* based on the work of Spalart and Allmaras [43, 44, 45]. A two-equation k - \mathcal{R} turbulence model is also available. These models generally overcome the limitations of algebraic models and resolve the μ_t field more accurately, but require substantially greater coding and computer resources to implement. All the *ADPAC* turbulence models are described in greater detail in the sections which follow.

A.7 Algebraic Baldwin-Lomax Turbulence Model

A relatively standard version of the Baldwin-Lomax [26] turbulence model is implemented for the algebraic model used in the *ADPAC* analysis. This model is computationally efficient, and has been successfully applied to a wide range of geometries and flow conditions. The Baldwin-Lomax model specifies that the turbulent viscosity be based on an inner and outer layer of the boundary layer flow region as:

$$\mu_t = \begin{cases} (\mu_t)_{inner}, & y \leq y_{crossover} \\ (\mu_t)_{outer}, & y > y_{crossover} \end{cases} \quad (\text{A.111})$$

where y is the normal distance to the nearest wall, and $y_{crossover}$ is the smallest value of y at which values from the inner and outer models are equal. The inner and outer model turbulent viscosities are defined as:

$$(\mu_t)_{inner} = \rho l^2 |\omega| \quad (\text{A.112})$$

$$(\mu_t)_{outer} = K C_{cp} \rho F_{wake} F_{Kleb} y \quad (\text{A.113})$$

Here, the term l is the Van Driest damping factor:

$$l = \kappa y (1 - e^{(-y^+/A^+)}) \quad (\text{A.114})$$

ω is the vorticity magnitude, F_{wake} is defined as:

$$F_{wake} = y_{max} F_{max} \quad (\text{A.115})$$

where the quantities y_{max} , F_{max} are determined from the function:

$$F(y) = y |\omega| [1 - e^{(-y^+/A^+)})] \quad (\text{A.116})$$

The term y^+ is defined as:

$$y^+ = y \left(\sqrt{\frac{\rho|\omega|}{\mu_{laminar}}} \right)_{wall} \quad (\text{A.117})$$

The quantity F_{max} is the maximum value of $F(y)$ that occurs across the boundary layer profile, and y_{max} is the value of y at which $F_{max}(y)$ occurs. The determination of F_{max} and y_{max} is perhaps the most difficult aspect of this model for three-dimensional flows. The profile of $F(y)$ versus y can have several local maximums, and it is often difficult to establish which values should be used. In this case, F_{max} is taken as the maximum value of $F(y)$ between a y^+ value of 100 and 1200. The function F_{Kleb} is the Klebanoff intermittency factor given by:

$$F_{kleb}(y) = [1 + 5.5 \left(\frac{C_{kleb}y}{y_{max}} \right)^6]^{-1} \quad (\text{A.118})$$

and the remainder of the terms are constants defined as:

$$\begin{aligned} A^+ &= 26 & C_{cp} &= 1.6 & C_{kleb} &= 0.3 \\ \kappa &= 0.4 & K &= 0.0168 \end{aligned}$$

In practice, the turbulent viscosity is limited such that it never exceeds 1000 times the laminar viscosity.

In order to properly utilize this turbulence model, a fairly large number of grid cells must be present in the boundary layer flow region, and perhaps of greater importance, the spacing of the first grid cell off of a wall should be small enough to accurately account for the inner “law of the wall” turbulent boundary layer profile region ($y^+ \leq 5$). Unfortunately, this constraint is often not satisfied due to grid-induced problems or excessive computational costs.

Practical applications of the Baldwin-Lomax model for three-dimensional viscous flow must be made with the limitations of the model in mind. The Baldwin-Lomax model was designed for the prediction of wall bounded turbulent shear layers, and is not well suited for flows with massive separations or large vortical structures. There are, unfortunately, a number of applications for turbomachinery where this model is likely to be invalid.

A.7.1 Modified Coefficients Baldwin-Lomax Model

In an effort to improve the Baldwin-Lomax model results, changes were made to the values of two of the model’s coefficients to account for the adverse pressure gradients occurring in turbomachinery. Based on a sensitivity analysis by Granville [18], the option to modify the coefficients in the standard Baldwin-Lomax turbulence model was added to *ADPAC*. The coefficients to be varied are C_{cp} and C_{Kleb} ; the standard values for these two coefficients are $C_{cp} = 1.6$ and $C_{Kleb} = 0.3$ [26]. The model modifications were treated in a similar manner as work presented by Turner and Jennions [46]. In their paper, these modification to the algebraic model produced results for a transonic fan almost as good as using a two-equation $k-\epsilon$ model.

The variation of these model coefficients with respect to pressure gradient is shown in Figure 3.3 [18]; the values for C_{cp} are read off the left-hand y-axis and range from 1.0 to 1.80, and the values for $C_{K_{elb}}$ are read off the right-hand y-axis and range from 0.44 to 0.64. The plot shows regions for both favorable and adverse pressure gradients, such that the values of the coefficients can be chosen properly for either compressor or turbine applications.

This modification to *ADPAC* was accomplished through the addition of two new input cards: **CCP** and **CKLEB**. The default values are set in the code to the standard Baldwin-Lomax values listed above, and are only changed if the **CCP** or **CKLEB** input lines are read in from the *ADPAC* input file. Values entered by the user for these coefficients are checked to ensure they are reasonable.

A.7.2 Wall Functions

The Baldwin-Lomax turbulence model currently implemented in the *ADPAC* code is valid for many flows of engineering interest provided that adequate mesh resolution is available to capture the subscale, near-wall viscous flow behavior which is crucial to correctly predict the overall boundary layer flow characteristics. For most applications, this implies that the first mesh point away from the wall must be located at a value of y^+ less than or equal to 1.0, where y^+ is defined as:

$$y^+ = \frac{y\rho_{wall}}{\mu_{wall}} \sqrt{\frac{\tau_{wall}}{\rho_{wall}}}$$

where τ_{wall} represents the viscous shear stress at the wall, and μ_{wall} and ρ_{wall} are the fluid viscosity and density at the wall. Unfortunately, for many cases it is not possible or feasible to comply with this restriction due to tradeoffs between minimizing both the overall number of grid points and mesh stretching ratios. Additional computational considerations must be given for time-dependent flows, where the maximum allowable time step is often dictated by the near-wall mesh spacing. The wall function method is widely used as an approach towards resolving the influence of the near-wall flow behavior without actually discretizing the inner portion of the boundary layer flow. The wall function method is, quite simply, an empirical specification of the wall shear stress based on local near-wall flow characteristics. This approach has several advantages including a computational savings (both CPU time and memory), and by providing a means by which additional empirical information about a particular flow may be introduced to the numerical solution (e.g. surface roughness modeled by a modified wall shear stress relation) at little or no additional cost. Wall function techniques for turbulence models have been proposed and used by many authors including Spalding [47], Wolfshtein [48], Patankar and Spalding [49], and Launder and Spalding [50].

The implementation of the wall function procedure in the *ADPAC* code is based on a rather novel approach involving the manipulation of the near-wall eddy viscosity. The “standard” method for implementing wall functions is to relax the no-slip wall boundary condition (allow a slip velocity at the wall) based on the requirement of no normal flow and the specified wall shear stress. This approach often requires specific modifications to the turbulence model, near-wall boundary conditions, viscous stress calculation, and energy equation solution routines. The finite volume formulation utilized in the

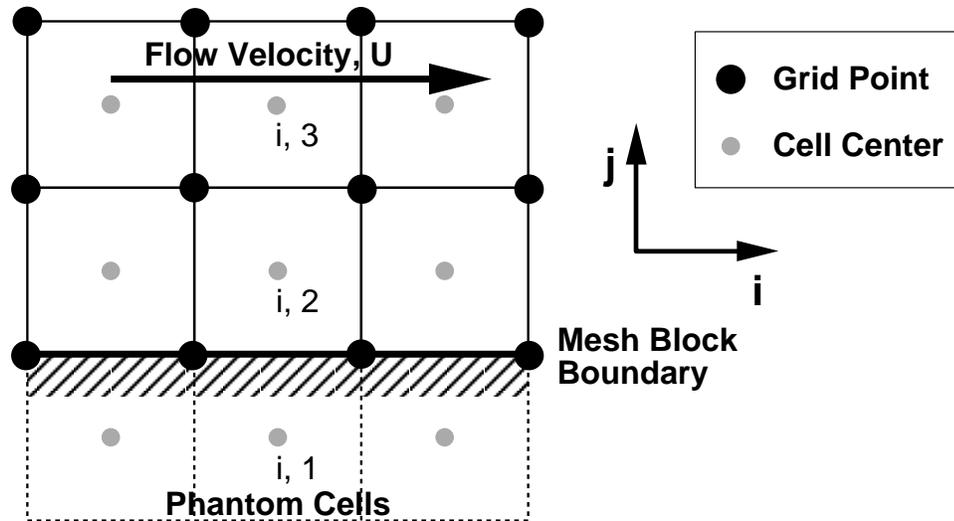


Figure A.8: Near-wall computational structure for wall function turbulence model.

ADPAC code allows for a number of options when implementing the wall functions formula. An illustration of the near-wall computational configuration for the *ADPAC* code is given in Figure A.8.

The objective during the *ADPAC* wall function implementation was to minimize the number of routines which required modification in order to implement the wall function model. This goal suggested that it was desirable to maintain the no-slip wall boundary condition in its original form, and hence the specified wall shear stress was implemented by manipulating the value of the phantom cell turbulent viscosity.

The *ADPAC* approach can be illustrated most easily by considering the viscous flow over a flat plate as shown in Figure A.8. A shear stress term at the wall such as:

$$\tau_{xy} = \mu \frac{\partial u}{\partial y}$$

is calculated numerically as:

$$\mu_{wall} * \frac{\Delta u}{\Delta y}$$

where, in this case, μ represents the combined turbulent and laminar viscosities:

$$\mu_{wall} = \frac{1}{2}(\mu_{i,2} + \mu_{i,1}) \frac{(u_{i,2} - u_{i,1})}{\Delta y}$$

The subscript i, j indicates a mesh oriented cell-centered flow value, where $i, 1$ is the value in the phantom cell, and $i, 2$ is the value at the first interior cell near the wall. As mentioned, many computational schemes satisfy the shear stress requirement by modifying the wall velocity boundary condition through manipulation of the term $u_{i,1}$ in the example above. For three-dimensional flows, this becomes even more complicated as multiple velocity components must be adjusted to satisfy the overall wall shear stress. The *ADPAC* implementation instead modifies the wall turbulent viscosity boundary condition through manipulation of the single term $\mu_{i,1}$. This implies that the turbulent viscosity at

the wall is non-zero, which violates the normal specification. However, since the turbulent viscosity is used exclusively in the calculation of the wall shear stress and heat conduction terms, the resulting calculations are consistent with the desired shear stress specification both in magnitude and direction, since the near-wall velocities drive the wall shear stress directly. This formulation, in effect, also implies a wall function based heat flux. For calculations in which heat transfer is unimportant, this effect is negligible. The influence of this approach on flows with heat transfer may be tested in future studies.

The shear stress specification used in the present application of wall functions is based on the following formula for the wall shear stress coefficient:

$$c_f = -0.001767 + 0.03177/Re_n + 0.25614/Re_n^2$$

The term Re_n is the Reynolds number based on near-wall velocity, density, and viscosity, where the length scale is the normal distance from the wall to the first interior domain calculation cell. The wall shear stress may then be calculated from the formula:

$$\tau_{wall} = 0.5 * c_f * \rho * V_{rel}^2$$

where V_{rel} is the near-wall relative flow total velocity.

A.8 One-Equation Spalart-Allmaras Turbulence Model

The turbulence model was modified slightly from its original presentation by Spalart and Allmaras [43] for incorporation into *ADPAC*. A brief review of the equations and constants comprising the turbulence model are presented below for review including the modifications for *ADPAC*.

The equation to calculate the eddy viscosity (μ_t) is given by:

$$\mu_t = \rho \tilde{\nu} f_{v1}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi \equiv \frac{\tilde{\nu}}{\nu} \quad (\text{A.119})$$

where ν is the kinematic viscosity, and $\tilde{\nu}$ is the working variable in the transport equation outlined below. The original Spalart-Allmaras formulation of the $\tilde{\nu}$ transport equation follow as:

$$\begin{aligned} \frac{D\tilde{\nu}}{Dt} = & \underbrace{c_{b1} [1 - f_{t2}] \tilde{S}\tilde{\nu}}_{\text{production}} + \underbrace{\frac{1}{\sigma} [\nabla \cdot ((\nu + \tilde{\nu}) \nabla \tilde{\nu}) + c_{b2} (\nabla \tilde{\nu})^2]}_{\text{diffusion}} \\ & - \underbrace{\left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2} \right] \left[\frac{\tilde{\nu}}{d} \right]^2}_{\text{destruction}} + \underbrace{f_{t1} \Delta U^2}_{\text{trip}} \end{aligned} \quad (\text{A.120})$$

The auxiliary equations needed to complete the model include:

$$\tilde{S} \equiv S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \quad f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}}, \quad S = |\nabla \times \vec{V}| \quad (\text{vorticity}) \quad (\text{A.121})$$

where d is the distance to the nearest viscous wall as calculated at the beginning of the *ADPAC* run.

$$f_w = g \left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6} \right]^{1/6}, \quad g = r + c_{w2}(r^6 - r), \quad r \equiv \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2} \quad (\text{A.122})$$

Since f_w reaches a constant for large values of r , r is upper bounded by the value of 10.

$$f_{t2} = c_{t3} \exp(-c_{t4}\chi^2) \quad (\text{A.123})$$

The capability of the Spalart-Allmaras turbulence model to incorporate a trip function is currently not part of the *ADPAC* implementation, but is presented below for completeness. The trip function f_{t1} is represented by:

$$f_{t1} = c_{t1} g_t \exp\left(-c_{t2} \frac{\omega_t^2}{\Delta U^2} [d^2 + g_t^2 d_t^2]\right) \quad (\text{A.124})$$

$$g_t \equiv \min\left(0.1, \frac{\Delta U}{\omega_t \Delta x}\right) \quad (\text{A.125})$$

where: d_t is the distance to the trip, ω_t is the wall vorticity at the trip, ΔU is the difference between the velocity at the current point and the trip location, and Δx is the grid spacing along the wall at the trip point.

The original constants for this model are listed below:

$$\begin{array}{llll} c_{b1} = 0.1355 & \sigma = 2/3 & c_{b2} = 0.622 & \kappa = 0.41 \\ c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma} & c_{w2} = 0.3 & c_{w3} = 2 & c_{v1} = 7.1 \\ c_{t1} = 1 & c_{t2} = 2 & c_{t3} = 1.1 & c_{t4} = 2 \end{array}$$

In a reprint of the original model formulation [44], Spalart recommended the following modifications to two of the constants:

$$c_{t3} = 1.2 \quad c_{t4} = 0.5 \quad (\text{A.126})$$

Additional modifications were also noted to help prevent \tilde{S} from going negative [45]. In this update of the model, the $f_{v2}(\chi)$ term is redefined below where c_{v2} is equal to 5:

$$f_{v2} = \left(1 + \frac{\chi}{c_{v2}}\right)^{-3} \quad (\text{A.127})$$

In order to match the finite volume approach coded in *ADPAC*, it is better to place the original transport equation presented by Spalart and Allmaras in a ‘‘conservative’’ form using $\rho\tilde{\nu}$ instead of $\tilde{\nu}$ only. Multiply both sides of the equation by ρ and make use of continuity:

$$\rho \frac{D\tilde{\nu}}{Dt} = \frac{D(\rho\tilde{\nu})}{Dt} - \tilde{\nu} \frac{D\rho}{Dt} = \frac{D(\rho\tilde{\nu})}{Dt} = \rho(\text{RHS})$$

For convenience, the numerical solution of the Spalart-Allmaras turbulence transport equation is based on the slightly altered conservation statement given below:

$$\begin{aligned} \frac{D(\rho\tilde{\nu})}{Dt} &= \rho c_{b1} [1 - f_{t2}] \tilde{S}\tilde{\nu} + \frac{\rho}{\sigma} \left[\nabla \cdot ((\nu + \tilde{\nu}) \nabla \tilde{\nu}) + c_{b2} (\nabla \tilde{\nu})^2 \right] \\ &\quad - \rho \left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2} \right] \left(\frac{\tilde{\nu}}{d} \right)^2 + \rho f_{t1} \Delta U^2 \end{aligned} \quad (\text{A.128})$$

Before proceeding, it is useful to rearrange the diffusion terms by noting the identity:

$$\nabla \cdot (\tilde{\nu} \nabla \tilde{\nu}) = (\nabla \tilde{\nu})^2 + \tilde{\nu} (\nabla^2 \tilde{\nu}) \quad (\text{A.129})$$

such that the governing transport equation can be restated as:

$$\begin{aligned} \frac{D(\rho\tilde{\nu})}{Dt} &= \rho c_{b1} [1 - f_{t2}] \tilde{S}\tilde{\nu} + \frac{\rho}{\sigma} \left[\nabla \cdot ((\nu + (1 + c_{b2})\tilde{\nu}) \nabla \tilde{\nu}) - c_{b2} \tilde{\nu} \nabla^2 \tilde{\nu} \right] \\ &\quad - \rho \left[c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2} \right] \left(\frac{\tilde{\nu}}{d} \right)^2 + \rho f_{t1} \Delta U^2 \end{aligned} \quad (\text{A.130})$$

Additional details of the implementation of the Spalart-Allmaras model into *ADPAC* (including the derivation of the Spalart-Allmaras transport equation for generalized coordinates, the implicit discretization of the Spalart-Allmaras transport equation, and the linearization of the Spalart-Allmaras transport equation source term) can be found in Reference [51].

Since this turbulence model, as do many, requires the distance to the nearest viscous wall to be known, a searching routine was developed to calculate this minimum distance for all computational cells in the mesh. Due to the flexibility of the multi-block capability and parallelization of *ADPAC*, this task was not as straight forward as might first appear. Despite the length of time needed to calculate the distance field, this calculation only needs to be performed once and the resulting values will be included in the turbulence model restart file, eliminating the need to execute the distance finding routine on the restart of a simulation. In fact it may be advisable to initially run *ADPAC* with zero iterations to calculate the near-wall distance field. These results can be checked in the *PLOT3D* output file (*case.p3d1eq*) after which the solution can be restarted.

The specification of inlet boundary conditions and initial conditions for the turbulence model transport variable ($\tilde{\nu}$) is handled by specifying a value of the non-dimensional variable χ ($\tilde{\nu}/\nu$). By specifying χ , the user does not need to account for variations in $\tilde{\nu}$ caused by changes in **PREF**, **TREF**, **DIAM**, or any other reference quantity used for non-dimensionalization. It was found in the cases tested, that a small initial value of χ does not provide a strong enough trigger for the production term and causes the solution to converge to the trivial solution ($\tilde{\nu} = 0.0$), resulting in a laminar flow field. Most cases are run using an initial value of χ equal to 20 with inlet values being specified at $\chi_{in} = 1$.

A.9 Two-Equation Turbulence Model

Limitations associated with the implementation of the algebraic (Baldwin-Lomax) turbulence model in the *ADPAC* code prompted the development of an advanced

(two-equation) turbulence modeling capability. As part of this study, an advanced turbulence model was incorporated into the *ADPAC* code to permit accurate prediction of a wider range of flow conditions, and hopefully improve the ability to predict highly loaded fan and compressor blade row flowfields. Initially, this effort was directed at primarily two-equation turbulence models ($k - \epsilon$, $q - \omega$, etc.) but was not necessarily intended to be limited to these models. Of particular interest was the use of “pointwise” turbulence models which do not require predetermination of the location of the nearest solid surface, as do most turbulence models. This feature provides a significant simplification of the turbulence modeling problems associated with a multiple blocked mesh code such as *ADPAC*.

The form of the two-equation turbulence model used in the *ADPAC* advanced turbulence model implementation was based on the two-equation $k - \mathcal{R}$ model described by Goldberg [52]. This is essentially an extension of the Baldwin-Barth [53] equation system as implemented by Goldberg [52]. The transport equations defining this model are derived from the “standard” form of the $k - \epsilon$ turbulence model equations as follows (see e.g. Wilcox [54])

$$\begin{aligned} \frac{\partial(\bar{\rho}\tilde{k})}{\partial t} + \nabla \cdot (\bar{\rho}\vec{V}\tilde{k}) &= \\ \nabla \cdot [(\mu + \frac{\mu_t}{\sigma_k})\nabla\tilde{k}] + P - \frac{(\bar{\rho}\tilde{k})^2}{\epsilon} & \\ \frac{\partial(\bar{\rho}\epsilon)}{\partial t} + \nabla \cdot (\bar{\rho}\vec{V}\epsilon) &= \\ \nabla \cdot (\mu + \frac{\mu_t}{\sigma_\epsilon})\nabla\epsilon + (2 - C_{\epsilon_1})\frac{\epsilon}{k} - (2 - C_{\epsilon_2})\bar{\rho}\tilde{k} & \end{aligned}$$

where P is the turbulent kinetic energy production term defined (for a Cartesian coordinate system) as:

$$\begin{aligned} P = \mu_t [(\nabla\tilde{u} + \frac{\partial\vec{V}}{\partial x}) \cdot \frac{\partial\vec{V}}{\partial x} + (\nabla\tilde{v} + \frac{\partial\vec{V}}{\partial y}) \cdot \frac{\partial\vec{V}}{\partial y} + (\nabla\tilde{w} + \frac{\partial\vec{V}}{\partial z}) \cdot \frac{\partial\vec{V}}{\partial z} - \frac{2}{3}(\nabla \cdot \vec{V})^2] \\ - \frac{2}{3}\bar{\rho}\tilde{k}(\nabla \cdot \vec{V}) \end{aligned}$$

By defining a new variable \mathcal{R} as

$$\mathcal{R} = \frac{k^2}{\epsilon} \tag{A.131}$$

and noting the identity

$$\frac{D\mathcal{R}}{\mathcal{R}} = \frac{2Dk}{k} - \frac{D\epsilon}{\epsilon}. \tag{A.132}$$

Baldwin and Barth subsequently developed a transport equation for \mathcal{R} from the k and ϵ equations. The final form of the $k - \mathcal{R}$ equation system can be expressed as:

$$\begin{aligned} \frac{\partial(\bar{\rho}\tilde{k})}{\partial t} + \nabla \cdot (\bar{\rho}\vec{V}\tilde{k}) &= \\ \nabla \cdot [(\mu + \frac{\mu_t}{\sigma_k})\nabla\tilde{k}] + P - \frac{(\bar{\rho}\tilde{k})^2}{\bar{\rho}\mathcal{R}} & \end{aligned}$$

$$\frac{\partial(\bar{\rho}\mathcal{R})}{\partial t} + \nabla \cdot (\bar{\rho}\vec{V}\mathcal{R}) =$$

$$\left(\mu + \frac{\mu_t}{\sigma_\epsilon}\right)\nabla^2\mathcal{R} - \frac{\bar{\rho}}{\sigma_\epsilon}\nabla\nu_t \cdot \nabla\mathcal{R} + (2 - C_{\epsilon_1})\frac{\mathcal{R}P}{\tilde{k}} - (2 - C_{\epsilon_2})\bar{\rho}\tilde{k}$$

where:

$$\mathcal{R} = \tilde{k}^2/\epsilon$$

In developing the diffusion terms in the \mathcal{R} equation certain terms were omitted based on order of magnitude considerations (although the actual steps used in the derivation are not obvious from the authors' description in Reference [53]).

Goldberg subsequently developed a "pointwise" turbulence model based on the $k - \mathcal{R}$ equation system based on early work by Launder and Sharma [55]. Here the adjective "pointwise" implies that the calculation of the turbulent viscosity is only dependent on flow data which is local to any point in the flow. Traditionally, most turbulence models require a calculation to determine the physical distance to the nearest wall or shear layer centerline. Since the *ADPAC* code possesses arbitrary numbers of mesh blocks, wall boundaries, etc., determining the distance from any given mesh point to the nearest wall is a formidable computational task, and therefore, the pointwise turbulence model provides an enormous simplification.

The calculation (as reported by Goldberg [52]) of the turbulent viscosity proceeds as follows:

$$\mu_t = C_\mu f_\mu \bar{\rho}\mathcal{R}$$

$$C_{\epsilon_1} = 1.44$$

$$C_{\epsilon_2} = 1.92$$

$$\sigma_k = 1.0$$

$$\sigma_\epsilon = 1.3$$

$$C_\mu = 0.09$$

$$f_\mu = f_\mu(\mathcal{R}_t) = \frac{1 - e^{-A_\mu\mathcal{R}_t^2}}{1 - e^{-A_\epsilon\mathcal{R}_t^2}}$$

$$A_\mu = 2.5 \times 10^{-6}$$

$$A_\epsilon = 0.2$$

$$\mathcal{R}_t = \frac{\bar{\rho}\mathcal{R}}{\mu} = \frac{k^2}{\nu\epsilon}$$

where \mathcal{R}_t is, in effect, the turbulence Reynolds Number.

The form of the $k - \mathcal{R}$ model is similar to other two-equation models, and the code was constructed so that it can be rapidly altered to solve other two-equation models ($k - \epsilon$, $q - \omega$, etc.) as needed. The $k - \mathcal{R}$ model is particularly attractive due to the simplified solid surface boundary conditions.

$$k = 0 \quad \mathcal{R} = 0 \tag{A.133}$$

and flowfield initialization

$$k \approx 1x10^{-5} \quad \mathcal{R} \approx 1x10^{-6} \quad (\text{A.134})$$

Numerical implementation of the two-equation turbulence modeling strategy involves several differences from the solution procedure described for the Reynolds-averaged Navier-Stokes equations. The k - \mathcal{R} equations were advanced in time using the same finite volume discretization and Runge-Kutta time marching procedures described in previous sections. No added dissipation was used for the k - \mathcal{R} equations. Instead, the convective cell face flux evaluations were performed using a first order upwind approximation for the flow variables on the cell face. The k and \mathcal{R} variables themselves were limited by enforcing the conditions $k, \mathcal{R} \geq 1x10^{-6}$. The production term P was also required to have a nonnegative value.

Appendix B

PARALLEL ADPAC EXECUTION SCRIPT

A sample UNIX shell script illustrating the basic steps required for compiling and running the ADPAC code in serial and in parallel for both the MPICH and a proprietary MPI libraries is given below. This script was developed for Silicon Graphics ORIGIN class multiprocessor computers. Similar scripts for other popular UNIX-based workstations can be created easily by modifying this script for other computing platforms.

```
#!/bin/csh
#set echo
#
#
cat <</eof
This is a global script for compiling various flavors of
ADPAC on Multiprocessor Silicon Graphics Workstations:

This script includes compilation for:

1. Serial code
2. Parallel execution using MPICH and APPLMPI-0.3
3. Parallel execution using SGI's MPI and APPLMPI-0.3

The script assumes that you have the following files available:

ADPAC_V1.0.tar.Z    --- compressed ADPAC   source distribution
sdblib.tar.Z       --- compressed SDBLIB  source distribution
csdb.tar.Z         --- compressed CSDB   source distribution
mpich.tar.gz       --- gzip'd      MPICH    source distribution
applmpi-0.3.tar.Z  --- compressed APPLMPI  source distribution

The script provides automatic de-compression and extraction for
each utility prior to compilation.

A simple "bump" test case is provided to verify the proper
operation of the code following execution. The user should
check to ensure the correct execution was performed and
the flow solution appears resonable.

Like all scripts, there are some inherent assumptions here:

The compiler on the machine is set up to be f90 for ADPAC, some machines
only have f77 - which could cause a problem. This might
```

require a respecification in some of the Makefiles. When in doubt,
make sure that the F77 environment variable is set to either f77 or f90.

The demo cases may not work if your computer requires some form of
"batch" job submission (PBS, NQS, LSF, etc.)

```
/eof
#
echo " "
echo " "
echo "====="
echo "Beginning ADPAC compile script for SGI installation"
echo "====="
echo " "
echo " "
sleep 1
#
#--> Try to detect the proper FORTRAN compiler to use
#
echo "Looking for the fortran compiler."
set tstf90 = `which f90 | grep "f90" | grep -v "not in" | wc -l`
set tstf77 = `which f77 | grep "f77" | grep -v "not in" | wc -l`

if ( $tstf90 == 1 ) then
    echo "Found f90."
    setenv F77 f90
else
    echo "No f90 found"
    if ( $tstf77 == 1 ) then
        echo "Found f77."
        setenv F77 f77
    endif
endif

if ( $tstf90 == 0 && $tstf77 == 0 ) then
    echo "Neither f90 or f77 found. Cannot determine FORTRAN compiler"
    echo "Unable to continue - Aborting."
    exit(1)
endif
sleep 1
#
#--> Make the SDBLIB Stuff
#
echo " "
echo "-----"
echo " Extracting the SDBLIB FORTRAN Library"
echo "-----"
echo " "
sleep 1

/bin/rm -r -f sdblib
zcat sdblib.tar.Z | tar xvf -
(cd sdblib; make clean)
(cd sdblib; /bin/rm -f *.a)
(cd sdblib; make)

#
#--> Make the CSDB Stuff
#

echo " "
echo "-----"
echo " Extracting the CSDB C Library"
echo "-----"
echo " "
sleep 1

/bin/rm -r -f csdb
zcat csdb.tar.Z | tar xvf -
```

```

(cd csdb; make clean)
(cd csdb; /bin/rm -f *.a)
(cd csdb; make)

#
#---> Make the MPICH stuff
#

echo " "
echo " -----"
echo " Extracting the MPICH Communication Libraries"
echo " -----"
echo " "
sleep 1

/bin/rm -r -f mpich
/usr/sbin/gzip -d mpich.tar.gz
tar xvf mpich.tar
/usr/sbin/gzip mpich.tar
(cd mpich; make clean)
(cd mpich; /bin/rm -f lib/IRIX64/ch_shmem/libmpi.a)
(cd mpich; ./configure -arch=IRIX64 -cc="cc -64 -mips4"
    -fc="f77 -64 -mips4" -opt="-O2" -device=ch_shmem)
(cd mpich; make)

#
#---> Make the Serial ADPAC Stuff
#

echo " "
echo " -----"
echo " Extracting the ADPAC FORTRAN Source Code"
echo " -----"
echo " "
sleep 1

/bin/rm -r -f ADPAC_V1.0
zcat ADPAC_V1.0.tar.Z | tar xvf -
(cd ADPAC_V1.0; make clean)
(cd ADPAC_V1.0; /bin/rm -f adpac_power_challenge
    adpac_power_challenge_mpi adpac_power_challenge_mpich)
(cd ADPAC_V1.0; pmake power_challenge)

#
#---> Make the applmpi-0.3 library for MPICH
#

echo " "
echo " -----"
echo " Extracting the APPLMPI Libraries"
echo " -----"
echo " "
sleep 1

/bin/rm -r -f applmpi-0.3
zcat applmpi-0.3.tar.Z | tar xvf -
(cd applmpi-0.3; make clean)
(cd applmpi-0.3; make clobber)
(cd applmpi-0.3; /bin/rm -f lib/*.a)
setenv MPICH_ROOT $PWD"/mpich"
setenv SGIMP ON
(cd applmpi-0.3; make)
(cd applmpi-0.3/src; mv libapplmpi.a libapplmpich.a)
(cd applmpi-0.3/lib; ln -s ../src/libapplmpich.a .)
echo " "
echo " "
echo "COMPILE MESSAGE: Error code 1 is expected from the applmpi-0.3 make"
echo " "

```

```

echo " "

#
#---> Now make the ADPAC Parallel MPICH executable
#

echo " "
echo " -----"
echo " Compiling the ADPAC+MPICH Parallel Code"
echo " -----"
echo " "
sleep 1

(cd ADPAC_V1.0; pmake power_challenge_mpich)

#
#---> Test to see if the SGI MPI 64 bit libraries have been installed
#
echo " "
echo " -----"
echo " Testing for the SGI MPI 64 bit Libraries"
echo " -----"
echo " "
sleep 1

echo " "
echo "Looking for the SGI MPI implementation"
set tstsgimpi = `versions mpi | grep mpi.sw64 | wc -l`
if ( $tstsgimpi == 2 ) then
    echo "Found SGI MPI 64 Bit Libraries"
    echo "Continuing with SGI MPI Parallel compilation"
#
#---> Now make the "Fake" MPICH directory using SGI MPI code
#

/bin/rm -r -f sgimpi
mkdir sgimpi
mkdir sgimpi/include
mkdir sgimpi/lib
ln -s /usr/include/mpi.h sgimpi/include/mpi.h
ln -s /usr/lib64/libmpi.a sgimpi/lib/libmpi.a

#
#---> Now remake the applmpi-0.3 library for the SGIMPI MPI
#

echo " "
echo " -----"
echo " Re-make APPLMPI with the SGI MPI Libraries"
echo " -----"
echo " "
sleep 1

(cd applmpi-0.3; make clean)
(cd applmpi-0.3; make clobber)
(cd applmpi-0.3; /bin/rm lib/libapplmpi.a)
setenv MPICH_ROOT $PWD"/sgimpi"
(cd applmpi-0.3; make)

#
#---> Now finally make the sgimpi ADPAC Parallel executable
#

echo " "
echo " -----"
echo " Compiling the ADPAC+SGI/MPI Parallel Code"

```

```

echo " -----"
echo " "
sleep 1

(cd ADPAC_V1.0; pmake power_challenge_mpi)
#
#--> In case the SGI MPI 64 bit libraries were not found..
#
else
    echo "Unable to find SGI MPI 64 bit libraries"
endif

#
#--> Test the serial and parallel codes using the NBUMP test case)
#
echo " "
echo " -----"
echo " Extracting the NBUMP test case"
echo " -----"
echo " "
sleep 1

/bin/rm -r -f NBUMP
zcat NBUMP.tar.Z | tar xvf -
#
#--> Serial ADPAC test run
#
echo " "
echo " -----"
echo " Executing the Serial ADPAC test case"
echo " -----"
echo " "
sleep 1

(cd NBUMP; /bin/rm -f Nbump.out.serial)
(cd NBUMP; ../ADPAC_V1.0/adpac_power_challenge < Nbump.input > Nbump.out.serial)
    echo " "
    echo "-----"
    echo "Serial ADPAC Test case ran to completion"
    echo "-----"
    echo " "
#
#--> Parallel ADPAC test run using MPICH
#
echo " "
echo " -----"
echo " Executing the Parallel ADPAC test case Using MPICH"
echo " -----"
echo " "
sleep 1

(cd NBUMP; /bin/rm -f Nbump.out.mpich)
(cd NBUMP; /bin/cp Nbump.input adpac.input)
(cd NBUMP; ../mpich/lib/IRIX64/ch_shmem/mpirun -np 4
    ../ADPAC_V1.0/adpac_power_challenge_mpich < adpac.input > Nbump.out.mpich)
    echo " "
    echo "-----"
    echo "Parallel ADPAC Test using MPICH ran to completion"
    echo "-----"
    echo " "
#
#--> Parallel ADPAC test run using SGI MPI
#
echo " "
echo " -----"
echo " Executing the Parallel ADPAC test case Using SGI MPI"
echo " -----"
echo " "

```

```

sleep 1

(cd NBUMP; /bin/cp Nbump.input adpac.input)
(cd NBUMP; /bin/rm -f Nbump.out.mpi)
(cd NBUMP; mpirun -np 4 ../ADPAC_V1.0/adpac_power_challenge_mpi <
  adpac.input > Nbump.out.mpi)
  echo " "
  echo "-----"
  echo "Parallel ADPAC Test using SGI MPI ran to completion"
  echo "-----"
  echo " "
#
#--> All Done!
#

echo " "
echo "*****"
echo "*****"
echo "**"
echo "** ADPAC SGI Compile/Execute Script completed **"
echo "**"
echo "*****"
echo "**"
echo "** The user should verify that all portions **"
echo "** of the analysis completed sucessfully and **"
echo "** the flow solution appears resonable. **"
echo "**"
echo "*****"
echo "*****"
echo " "

```

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE February 1999	3. REPORT TYPE AND DATES COVERED Final Contractor Report	
4. TITLE AND SUBTITLE ADPAC v1.0 – User's Manual			5. FUNDING NUMBERS WU-538-03-11-00 NAS3-27394 Task 15	
6. AUTHOR(S) Edward J. Hall, Nathan J. Heidegger, and Robert A. Delaney				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Allison Engine Company P.O. Box 420 Indianapolis, Indiana			8. PERFORMING ORGANIZATION REPORT NUMBER E-11089	
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-1999-206600	
11. SUPPLEMENTARY NOTES Project Manager, Christopher J. Miller, Structures and Acoustics Division, NASA Lewis Research Center, organization code 5940, (216) 433-6179.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category: 02 This publication is available from the NASA Center for AeroSpace Information, (301) 621-0390.			12b. DISTRIBUTION CODE Distribution: Nonstandard	
13. ABSTRACT (Maximum 200 words) The overall objective of this study was to evaluate the effects of turbulence models in a 3-D numerical analysis on the wake prediction capability. The current version of the computer code resulting from this study is referred to as ADPAC v7 (Advanced Ducted Propfan Analysis Codes – Version 7). This report is intended to serve as a computer program user's manual for the ADPAC code used and modified under Task 15 of NASA Contract NAS3-27394. The ADPAC program is based on a flexible multiple-block grid discretization scheme permitting coupled 2-D/3-D mesh block solutions with application to a wide variety of geometries. Aerodynamic calculations are based on a four-stage Runge-Kutta time-marching finite volume solution technique with added numerical dissipation. Steady flow predictions are accelerated by a multigrid procedure. Turbulence models now available in the ADPAC code are: a simple mixing-length model, the algebraic Baldwin-Lomax model with user defined coefficients, the one-equation Spalart-Allmaras model, and a two-equation k-R model. The consolidated ADPAC code is capable of executing in either a serial or parallel computing mode from a single source code.				
14. SUBJECT TERMS Computational fluid dynamics; Stall; Turbomachinery			15. NUMBER OF PAGES 268	
			16. PRICE CODE A12	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT	