Swift - Multiblock Analysis Code for Turbomachinery
User's Manual and Documentation
Version 300, August, 2003

Dr. Rodrick V. Chima

NASA Glenn Research Center, MS 5-10

21000 Brookpark Road

Cleveland, Ohio 44135 USA

phone: 216-433-5919

fax: 216-433-5802

email: Rodrick.V.Chima@nasa.gov

internet: http://www.grc.nasa.gov/WWW/5810/rvc

Introduction

Swift is a multiblock computer code for analysis of three-dimensional viscous flows in turbomachinery. The code solves the thin-layer Navier-Stokes equations using an explicit finite-difference technique. It can be used to analyze linear cascades or annular blade rows with or without rotation. Three turbulence models are available. Limited multiblock capability can be used to model tip clearance flows and multistage machines.

Swift has been tested on numerous fan and turbine blades and has been used heavily at NASA Glenn Research Center for fan analysis and design, analysis of turbine endwall heat transfer, and many other applications. Swift is a multiblock version of RVC3D, which was originally described in (2) along with calculations of a blunt fin and an annular turbine. The flow equations, numerical method, and calculations of a transonic fan were given in (3). The algebraic turbulence models and calculations of turbine endwall heat transfer were described in (4), and the k-ω turbulence model was described in two-dimensions in (5). The multiblock code known as Swift was introduced in (6) along with calculations made of tip clearance flow in a transonic compressor. The preconditioning scheme and calculations of a large low-speed centrifugal impeller were described in (7). Multistage capability was described in (8) along with calculations of a two-stage turbine. Finally, the AUSM$^+$ and H-CUSP upwind schemes and SST version of the k-ω model were described in (13.)

This report serves as the user's manual and documentation for Swift. The code and some aspects of the numerical method are described. Steps for code installation and execution are given for both unix workstations and Windows PC's. The grid, input, and output variables are described in detail.

Features of Swift

- **New in Version 300**
  AUSM$^+$ & H-CUSP upwind schemes
  k-ω SST turbulence model
  Surface roughness input as actual height instead of wall units
  H-grids for blades or ducts
  Simultaneous modeling of hub & tip clearances
  Simultaneous output of inlet & exit mass flow
  Loss and efficiency calculated across each blade row

  Code distributed in tarred format that should compile and run directly on unix machines, or a zipped format that should compile and run directly on Windows PC's.
  Code converted to Fortran 90
  Dynamic memory allocation reduces memory requirements and avoids recompiling for most problems

  Note: For inlet H-grids, the dummy grid line at j=1 has been deleted in Swift v.300. Compatible grids can be generated with TCGRID v.300. Files with inlet H-grids generated with previous versions of TCGRID or Swift are incompatible with version 300. Please let me know if this creates a serious problem.

- **Applications**
  Linear cascades
  Axial compressors and turbines
  Isolated blade rows or multistage machines
  Centrifugal impellers and mixed-flow machines without splitters
  Radial diffusers
  Pumps
  Rectangular ducts

- **Multi-block Capability**
  C-grids around blades
  H-grid upstream
  O-grids in hub- or tip-clearance regions (or periodic clearance model)
  Mixing-planes between blade rows

- **Formulation**
  Navier-Stokes equations written in Cartesian coordinates with rotation about the x-axis
  Thin-layer equations in streamwise direction, all cross-channel viscous terms retained
  Second-order finite-difference discretization

- **Turbulence Models**
  Baldwin-Lomax (algebraic)
  Cebeci-Smith (algebraic)
  Wilcox's k-ω (two-equation)
  Surface roughness effects in all models

- **Numerical Method**
  Explicit multi-stage Runge-Kutta scheme
  Variable time-step and implicit residual smoothing for convergence acceleration
  Preconditioning for low-speed (incompressible) flows

- **Input**
  General grid files in PLOT3D format, usually generated using TCGRID
  Namelist input of flow parameters

- **Printed Output**
  Residual history
  Spanwise output of circumferentially-averaged flow quantities at the grid inlet and exit
  Streamwise output of blade surface properties
  Printed output can be edited manually and plotted with Microsoft Excel or other line plot software

- **Computer Requirements**

Runs as a batch process on most unix, linux, or Windows computers
Parallel processing on SGI computers using OpenMP directives
Runs well under linux, has been run under Windows
Solution times range from one to several hours on modern workstations
Written in Fortran 90, requires a Fortran 90 compiler

- **Graphical Output**

  No graphical output is provided with Swift but access to some CFD visualization package is absolutely necessary to view and evaluate the solutions. Grid and solution files are in standard PLOT3D format and can be read directly and plotted with public-domain CFD visualization tools PLOT3D and FAST, or the commercial tools EnSight, FIELDVIEW, or TecPlot. Check the following web sites for more information.

  PLOT3D & FAST: http://www.nas.nasa.gov/Research/Software/swdescription.html
  TecPlot: http://www.amtec.com/
  FIELDVIEW: http://www.ilight.com/
  EnSight: http://www.mscsoftware.com.au/products/software/cei/ensight/

k	α^1	α^2	α^3	α^4	α^5	λ^*
2	1.2	1.				.95
3	.6	.6	1.			1.5
4	.25	.3333	.5	1.		2.8
5	.25	.1667	.375	.5	1.	3.6

Table 1 — Runge-Kutta parameters $\alpha^1 - \alpha^5$ and maximum Courant number λ^* for *k*-stage schemes.

Numerical Method

Multistage Runge-Kutta Scheme

Multistage schemes were developed by Jameson, Schmidt, and Turkel (10) as a simplification of classical Runge-Kutta integration schemes for ODE's. The simplification reduces the required storage, but also reduces the time-accuracy of the schemes, usually to second order. The following discussion of these schemes should give some guidance in choosing parameters for running the code.

A k-stage scheme may be written as:

$$q^k = q^0 - \alpha^k \Delta t (R_I^k + R_V^0)$$

where q is an array of five conservation variables (see "Solution Q-File", pp. 27), k is the stage count, q^0 is the previous time step, α^k are the multistage coefficients discussed below, Δt is the time step, R_I^k is the inviscid part of the residual, and R_V^0 is the viscous part of the residual including the artificial dissipation. Note that R_I^k is evaluated every stage, but R_V^0 is evaluated only at the initial stage for computational efficiency.

The maximum stable Courant number λ^* for an n-stage scheme can be shown to be $\lambda^* = n - 1$. The actual stability limit depends on the choice of α^k. For consistency α^n must equal 1, and for second-order time accuracy α^{n-1} must equal *1/2*. The values of α^k used in the code and the theoretical maximum Courant number λ^* are set by a data statement in subroutine rk and are given in table 1.

The number of stages is set with the variable *nstg*. Using $nstg = 4$ is recommended, although Jameson et. al. tend to favor 5 stages. Updating the physical and artificial viscosity terms more often increases the robustness of the multistage schemes, but also increases the CPU time per stage. Setting *ndis=2* causes the 4-stage scheme to update the dissipative terms during stages 1 and 2, and causes the 5-stage scheme to update them during stages 1, 3,and 5. A good compromise is *nstg=4* and *ndis=2*.

A spatially-varying Δt is used to accelerate the convergence of the code. Setting $ivdt = 1$ causes the Courant number to be set to a constant (input variable *cfl*) everywhere on the grid. The spatially-varying Δt option is highly recommended. For a multiblock grid the time step is recalculated every iteration. For a single block grid the time step is stored and recalculated every *icrnt* iterations. Set *icrnt* to a moderate number, e.g. 10, so that the time step is recalculated occasionally. The time step is recalculated when the code is restarted and may cause jumps in the residual if *icrnt* is too big.

Implicit residual smoothing (described later) may be used to increase the maximum Courant number by a factor of two to three, thereby increasing the convergence rate as well.

Artificial Viscosity

The code uses second-order central differences throughout and requires an artificial viscosity term to prevent odd-even decoupling. A fourth-difference artificial viscosity term is used for this purpose. This term is third-order accurate in space and thus does not affect the formal second-order accuracy of the scheme. The input variable *avisc4* scales the fourth-difference artificial viscosity, and should be set between 0.25 and 2. Start with *avisc4=1.0*. If the solution is wiggly, increase *avisc4* by 0.5. If it is smooth, try reducing avisc4 by 0.5. Larger values of *avisc4* may improve convergence somewhat, but the magnitude of *avisc4* has little effect on predicted losses or efficiency.

The code also uses a second-difference artificial viscosity term for shock capturing. The term is multiplied by a second difference of the pressure that is designed to detect shocks. Note that the second-difference artificial viscosity is first-order in space, so that the solution reduces to first-order accurate near shocks. Two other switches developed by Jameson (10) are used to reduce overshoots around shocks. The input variable *avisc2* scales the second difference artificial viscosity, and should be

set between 0. and 2. Use 0. for purely subsonic flows, and start with *avisc2=1.0* for flows with shocks. If shocks are wiggly, increase *avisc2* by 0.5. If they are smeared out, try decreasing *avisc2* by 0.5. Shocks will be smeared over four or five cells regardless of the value of *avisc2*. The magnitude of *avisc2* also has little effect on predicted loss or efficiency.

Eigenvalue scaling described in (3) is used to scale the artificial viscosity terms in each grid direction. This greatly improves the robustness of the code. The artificial viscosity is also reduced linearly by grid index near walls to reduce its effect on the physical viscous terms. Input variables *jedge, kedgh,* and *kedgt* are the indices where the linear reduction begins.

A first-order artificial viscosity term may be added to smooth the solution drastically during solution start-up. The variable *avisc1* scales this term. First-order artificial viscosity will greatly improve the convergence rate while greatly diminishing the accuracy of the solution. It will thicken boundary layers, smear shocks, and greatly increase predicted loss. Do not use first-order artificial viscosity except to start a new solution. A warning is printed in the output when $avisc1 > 0$.

Implicit Residual Smoothing

Implicit residual smoothing was introduced by Lerat in France and popularized by Jameson in the U.S. as a means of increasing the stability limit and convergence rate of explicit schemes. The idea is simple: run the multistage scheme at a high, unstable Courant number, but maintain stability by smoothing the residual occasionally using an implicit filter. The scheme can be written as follows:

$$(1 - \varepsilon_\xi \delta_{\xi\xi})(1 - \varepsilon_\eta \delta_{\eta\eta})(1 - \varepsilon_\varsigma \delta_{\varsigma\varsigma})\overline{R} = R$$

where ε_ε, ε_η, and ε_ς are constant smoothing coefficients in the three body-fitted coordinate directions ε, , and ς indicated in figure 1. Here δ is a second-difference operator, R is the smoothed residual, and R is the unsmoothed residual.

It can be shown that if the scheme converges, implicit residual smoothing does not change the solution. Linear stability theory shows that the scheme can be made unconditionally stable if ε_i is big enough, but also shows that the effects of the artificial viscosity are diminished as the Courant number is increased. In practice the best strategy seems to be to double or triple the Courant number of the unsmoothed scheme. If the residual is smoothed after every stage, the theoretical 1-D values of ε_i needed for stability are given by:

$$\varepsilon_i \geq \frac{1}{4}\left[\left(\frac{\lambda}{\lambda^*}\right)^2 - 1\right]$$

where λ^* is the Courant limit of the unsmoothed scheme (given in the previous table,) and λ is the larger operating Courant number. For example, to run a four-stage scheme at a Courant number $\lambda = 5.6$, the smoothing coefficient should be:

$$\varepsilon_i \geq \frac{1}{4}\left[\left(\frac{5.6}{2.8}\right)^2 - 1\right] = 0.75$$

A single variable $eps = \varepsilon$ is input to Swift. The 1-D limit given above usually gives a reasonable estimate for ε, but the code will converge best when ε is minimized. Values of ε_ε, ε_η, and ε_ς are evaluated at each grid point within the code by scaling *eps* using the same Eigenvalue scaling coefficients used for the artificial dissipation. This has proven to be quite robust.

In rare cases it may be necessary to increase the residual smoothing coefficient in a particular direction. This can be accomplished using input variables *epi, epj,* and *epk,* which are constants (usually 1.) that multiply ε_i at each point.

Implicit residual smoothing involves a scalar tridiagonal inversion for each variable along each grid line in each direction. It adds about 20 percent to the cpu time when applied after each stage. Smoothing can be done after every other stage to reduce cpu time (about 7 percent,) but *eps* must be increased (approximately doubled.)

Recommended starting values are: $nstg = 4$, $cfl = 5.6$, $irs = 1$, and $eps = 0.75$. If the code blows up quickly try increasing *eps* to 1.5. Very large values of *eps* (e.g. > 3) may stabilize a stubborn calculation but prevent the residuals from decreasing. If the residuals drop a little then climb to a large, constant value, *eps* is probably too big and the solution is probably incorrect.

Preconditioning

Density-based schemes like Swift solve the continuity equation by driving the density residual to zero. For low speed (nearly incompressible) flows the density residual is physically near zero, and the schemes fail to converge. Preconditioning,

described by Turkel in (12) improves the convergence rate in two ways. First, it replaces the q-variables $q = [\rho, \rho u, \rho v, \rho w, e]$ with variables that are better-behaved at low speeds $W = [p, \rho u, \rho v, \rho w, h]$, where p is the pressure and h is the total enthalpy. Second it multiplies the equations by a matrix designed to equalize the wave speeds of each equation. Preconditioning works extremely well for the Euler equations and less well for the Navier-Stokes equations. It allows solutions at very low speeds that simply would not converge otherwise.

It can be difficult to get a solution started with preconditioning. It is usually best to run 200-500 iterations with preconditioning turned off, then run to convergence with preconditioning turned on.

The preconditioning operator is designed so that it has no effect on the steady-state solution; however, for the central difference scheme preconditioning also modifies the artificial dissipation operator. This tends to reduce the artificial dissipation when preconditioning is used.

The preconditioning matrix has the local relative Mach number in the denominator and must be limited to avoid dividing by a small number. The denominator is limited to be $> pck \times (M'_{ref})^2$, where pck and M'_{ref} are input parameters. M'_{ref} is roughly the largest relative Mach number "" ≤ 1 expected in the flow, and pck is a scaling parameter, typically $0.1 - 0.3$. The solution will diverge quickly if these parameters are too small, and may converge slowly if they are too big.

At low Mach numbers the default inlet and exit boundary conditions may not work properly. In general, if $M' < 0.1$, or if contour plots of pressure or velocity look bad near the inlet or exit, set *ibcinu=2* to extrapolate meridional velocity at the inlet, and set *ibcext0=0* to extrapolate velocity components at the exit.

H-CUSP Scheme

The H-CUSP upwind scheme was developed by Tatsumi, Martinelli, and Jameson in 1994-5. The scheme is implemented as a standard central-difference (C-D) flux plus a dissipative flux. For efficiency the dissipative flux is only updated after the first stage or two the Runge-Kutta scheme. Nevertheless, the H-CUSP scheme is 30-40 percent slower than the standard C-D scheme. The H-CUSP scheme has excellent shock capturing properties but is a little more dissipative than the C-D or AUSM[+] schemes. Details of the implementation of the H-CUSP scheme in Swift are given in (13.)

The dissipative flux is calculated using a a blend of two schemes: the SLIP scheme and the CUSP scheme. The SLIP (Symmetric Limited Positive) scheme is used at low speeds. It adds a second difference of the conservation variables to the C-D flux. The conservation variables are evaluated at cell centers using the SLIP limiter, which gives a third-order dissipation in smooth regions of the flow and first-order dissipation near shocks. The CUSP (Convective Upward Split Pressure) scheme is used at high speeds. It adds upwind differences of the inviscid fluxes to give a true upwind scheme. The fluxes are split into advective and pressure parts similar to the AUSM[+] scheme. The H-CUSP scheme is based on the total enthalpy. Switching between the SLIP and CUSP schemes is done using piecewise linear functions of the interface Mach number.

In the H-CUSP scheme the low-speed dissipation is scaled by $hcuspk \times M'_{ref}$. where M'_{ref} is defined as before for preconditioning and *hcuspk* sets the minimum value of dissipation. Typical values for *hcuspk* are $0.05 - 0.10$. Smaller values may cause wiggles in the solution. Larger values may improve convergence but will increase predicted losses.

AUSM[+] Scheme

The Advection Upstream Splitting Method (AUSM) upwind scheme was introduced by Liou and Steffen in 1991. Since the AUSM scheme evaluates the inviscid fluxes using true upwind differences at each stage of the Runge-Kutta scheme, it is 40-50 percent slower than the standard C-D scheme. The AUSM[+] scheme has excellent shock capturing properties and seems to be more robust at low speeds than the C-D or H-CUSP schemes. Details of the implementation of the AUSM[+] scheme in Swift are given in (13.)

The AUSM scheme splits the inviscid fluxes into advective and acoustic parts, and differences them separately. Upwinding depends on polynomial functions of the cell interface Mach number. Second-order accuracy is maintained using the van Albada limiter. Liou revised the polynomial functions in 1996 and renamed the scheme AUSM[+].

In 1999 Liou and Edwards described a numerical speed of sound, which makes the inherent numerical dissipation of the scheme behave properly even at very low speeds. Two additional diffusive terms are also included in the scheme to insure pressure-velocity coupling at low speeds: a pressure-diffusion term is added to the interface Mach number, and a velocity diffusion term is added to the interface pressure.

In the AUSM[+] scheme the numerical speed of sound is used to calculate the pressure fluxes and the pressure diffusion term. The numerical speed of sound is a function of a reference Mach number limited by $ausmk \times M'^2_{ref}$, where M'_{ref} is defined as before for preconditioning. *Ausmk* controls the dissipation of the scheme to some extent, but in a less obvious way than *hcuspk*. Typical values for *ausmk* are $0.3 - 0.8$. Larger values seem to be needed for convergence, but don't seem to hurt accuracy.

Grids

Swift can handle single-block grids and a limited variety of multi-block grids. Grids are usually generated using TCGRID (9). Dummy grid lines are used to handle periodic boundary conditions and transfer of information between blocks, and must be included in the grid file. All grid types currently supported by Swift will have a dummy grid line at j=jm, except for grids in rectangular ducts which have no dummy grid line. Grids are stored in standard PLOT3D format(see "Grid XYZ-File", pp. 27.)

The connectivity between the grids is specified using an index file (see "Index File", pp. 22.) In TCGRID, setting *iswift=1* in namelist 5 will automatically add dummy grid lines and produce a preliminary index file.

C-grids (Blades)

The basic Swift grid consists of a C-type grid around a blade, as shown in figures 1 and 2. The i-direction goes from i=1 at the lower exit to i=im at the upper exit. The j-direction goes from j=1 at the blade to j=jm-1 at the periodic boundary (j=jm at the dummy grid line.) The k-direction goes from 1 at the hub to k=km at the tip. Calculations run with a single grid avoid some data I/O and thus run about 10 percent faster than a multiblock grid with the same number of points.

H-grids (Upstream, Blades, Rectangular Ducts)

Three types of H-grids are supported in Swift v.300. The type of H-grid is flagged by index file variable i1 (see Index File, pp. 22.
  1. An H-grid can be added to extend a C-grid upstream, as shown in figure 2. A dummy grid line is needed at j=jm to apply the periodic boundary conditions. Flagged by setting *i1=0.*
  2. A single H-grid can be used inside a blade passage, as shown in figure 3. A dummy grid line is needed at j=jm to apply the periodic boundary conditions. Multiblock grids are not supported with this grid type. Flagged by setting *i1>1.*
  3. An H-grid can also be used inside a rectangular duct (not shown.) No dummy grid lines are needed. Multiblock grids are not supported with this grid type. Flagged by setting *i1=1.*

In each case the i-index goes from inlet to exit, the j-direction goes from blade to blade, and the k-direction goes from hub to tip.

O-grids (Hub and Tip Clearance Gaps)

O-type grids can be used to resolve the hub or tip clearance regions of blade (visible in figure 5.) Clearance regions can also be modeled using a simple periodic boundary condition that does not require gridding the region. For O-grids the i-direction starts at the trailing edge cut and wraps around the O. The j-direction starts at the center line cut and goes to the perimeter of the O. j=jm is a dummy grid line that overlaps the connecting C-grid by one point. The k-direction goes from the hub to the blade for hub clearances, or from the blade to t he casing for tip clearances.

Multistage Grids

Using TCGRID, multi-stage grids must be generated row-by-row. A grid for a one-stage turbine is shown in figure 4.

The meridional grid must be continuous across multistage grids. This means that all grids must have identical hub and casing coordinates *zhub, rhub*, etc., identical number of spanwise points *km*, identical spanwise spacing parameters *dshub, dstip*, etc., and identical clearance parameters. Neighboring grids must have matching downstream/upstream boundary coordinates *zbc, rbc*.

Blades must be oriented correctly in the θ-direction and placed in their correct axial location. TCGRID variables *zscale*, *tscale*, *rscale, ztrans,* and *tflip* will help with blade placement.

Dummy grid lines must overlap the neighbors exactly one cell, as shown in figure 4. The spacing ahead of the inlet is set using *dsmax*. The spacing at the exit can be set using *dslap*. Thus *dslap* for the upstream grid must equal *dsmax* for the downstream grid. It may take a few iterations to get nicely overlapping grids. It may help to start with the downstream grid and work upstream.

Grid files from neighboring blade rows are combined into a multi-block PLOT3D file using the included utility routine MULTIX, which prompts for file names and should be self-explanatory. TCGRID generates index files for each blade row. These must be merged manually using an editor.

Additional details about generating multistage grids are given in the TCGRID user's manual.

Getting Started – Unix

Unpacking

    For unix systems Swift is supplied as a gzipped tar file called swift.tgz. To open:

```
gunzip swift.tgz
tar -xvf swift.tar
```

This will create a directory called swift.300 with subdirectories for the source, documentation, and test cases.

Compiling

    Swift must be compiled with a Fortran 90 compatible compiler. Edit modules.f90 and change the maximum array size if desired (see *Parameter Statement below.*) Edit the Makefile and change the compiler name (f90) and options (OPTS) as necessary for your compiler. Full optimization and no debug are recommended. On SGI machines, the -apo option compiles Swift for multiple processors.

```
cd swift.300/src; make
```

The executable file swift remains in the src directory. To delete the object files and executable,

```
make clean
```

Important Note: Modules.f90 and mut1d.f90 contain several Fortran 90 modules that are used within other routines. They must be compiled before any other routines are compiled. If you ever need to recompile, be sure to compile modules.f90 and mut1d.f90 first.

Parameter Statement

    Swift uses dynamic memory allocation for most arrays to avoid redimensioning. However, for programming convenience the maximum size of many work arrays are set using a Fortran module defined in modules.f90.

```
module maxsize
! Maximum dimensions of q-vector & small arrays
! Must be compiled first
! Change & recompile for larger grids

  save
  integer,parameter::ni=255,nj=54,nk=63
  integer,parameter::maxjk=max(nj,nk)

end module maxsize
```

This grid size (255 x 54 x 63) is large enough for most problems but can be increased to any size as needed.

Running Swift for the Goldman Turbine Vane Test Case

    First cd to the appropriate directory and generate the grid using TCGRID.

```
cd swift.300/gold
~/tcgrid.300/src/tcgrid < gold.int
```

The grid should run in a few seconds. The output can be redirected to a file if desired. The grid and index file are written to the appropriate Fortran units for Swift, but it helps to give them descriptive names and link them back to the correct unit.

```
mv fort.1 gold.xyz; ln gold.xyz fort.1
mv fort.10 gold.ind; ln gold.ind fort.10
```

Gold.ins contains the namelist input. It should be set up to run 50 iterations of the central-difference scheme. Run Swift as a standard unix process.

```
../src/swift < gold.ins > gold.out &
```

The 50 iterations should run fairly quickly. You will need to run about 1500 iterations to get a converged solution. The output solution file is written to fort.3 in binary format. Rename it and link it to fort.2.

```
mv fort.3 gold.0050.q; ln gold.0050.q fort.2
```

Edit the input file, set *iresti=1* and *itmax=1450*. Then run Swift again as before.

Alternative I

    Gold.jcl is a unix shell script that names the files, cats the namelist input to a file named input, and runs Swift. Two shell variables are set near the top. *pin* is a prefix for the input file names. The solution input file is named *pin*.q (not used if

iresti=0.) *pout* is a prefix for the output file names. The solution output file is named *pout*.q and the printed output is named *pout*.out. The shell variable *kw* is a flag for the k-ω file names. If kw=1 the k-ω files are named *pin*.kw and *pout*.kw.

Alternative II

Setting *iopen=1* in the input file causes all files to be opened explicitly with a default file name. For example, the grid must be named grid.xyz. Other file names are given in Table 2 on page 11. By using the default file names you can avoid the file linking steps, but you may want to rename the files with more descriptive names later.

It is also possible to input your own file names using the namelist input. See the note under File Names, pp. 11.

Getting Started – Windows

Unpacking

For Windows, Swift is supplied as a zipped file called swift.zip. Use Winzip or PKZIP to unzip it. This will create a directory called swift.300 with subdirectories for the source, documentation, and test cases.

Compiling

Swift must be compiled with a Fortran 90 compatible compiler. Compaq Visual Fortran works well, but I don't have enough experience to give step-by-step instructions. Please let me know if you can adapt my Makefile or have any information about other compilers. In general, using Developer Studio:

Go to swift.300/src

Open a new Fortran console application

Edit modules.f90 and change the maximum array size if desired (see *Parameter Statement below.*)

Set compile options for full optimization and no debug tables

Compile modules.f90 and mut1d.f90 first

Build swift

Important Note: Modules.f90 and mut1d.f90 contain several Fortran 90 modules that are used within other routines. They must be compiled before any other routines are compiled. If you ever need to recompile, be sure to compile modules.f90 and mut1d.f90 first.

Parameter Statement

Swift uses dynamic memory allocation for most arrays to avoid redimensioning for most problems. However, for programming convenience the maximum size of many work arrays are set using a Fortran module defined in modules.f90.

```
      module maxsize
   ! Maximum dimensions of q-vector & small arrays
   ! Must be compiled first
   ! Change & recompile for larger grids

      save
      integer,parameter::ni=255,nj=54,nk=63
      integer,parameter::maxjk=max(nj,nk)

      end module maxsize
```

This grid size (255 x 54 x 63) is large enough for most problems but can be increased to any size as needed.

Running Swift

To run the Goldman turbine vane test case, open a DOS window (Start/Programs/Accessories/Command prompt.)

```
      cd swift.300\gold
```

Edit the TCGRID input file gold.int and set *iopen=1* in namelist 3. This will cause all files to be opened with the default names given in Table 2 on page 11. Now run TCGRID.

```
      c:\tcgrid.300\src\tcgrid < gold.int
```

The grid should run in a few seconds. The output can be redirected to a file if desired. The grid and index file will be named grid.xyz and index.dat respectively.

Edit the Swift input file gold.ins and set *iopen=1* in namelist 3. The input should already be set up to run 50 iterations of the central-difference scheme. Now run Swift.

```
      ..\src\swift < gold.ins > gold.out
```

The 50 iterations should run fairly quickly. You will need to run about 1500 iterations to get a converged solution. The output solution file is written to q_out.q in binary format. Rename it to q_in.q.

```
      rename q_out.q q_in.q
```

Edit the input file, set *iresti=1* and *itmax=1450*. Then run Swift again as before.

Unit	Default name	Description	Reference
fort.1	grid.xyz	grid file from TCGRID	Grid XYZ-File, pp. 27
fort.2	q_in.q	binary input solution file, read if *iresti=1*	Solution Q-File, pp. 27
fort.3	q_out.q	binary output solution file, written if *iresto=1*	Solution Q-File, pp. 27
fort.10	index.dat	ASCII index file, required	Index File Variables, pp. 22
fort.7	kw_in.kw	binary input k-ω file, read if *ilt = 4 or 5*	Turbulence Model k-w File, pp. 27
fort.8	kw_out.kw	binary output k-ω file, written if *ilt = 4 or 5*	Turbulence Model k-w File, pp. 27
fort.13	profile_in.dat	ASCII input qin file, read if *iqin = 1*	Inlet and Exit Profiles, pp. 27
fort.14	profile_ex.dat	ASCII output pex file, read if *ipex = 1*	Inlet and Exit Profiles, pp. 27
fort.15	profile_out.dat	ASCII output span file, written if *ispan = 1*	Inlet and Exit Profiles, pp. 27

Table 2 — Files used by Swift

File Names

The namelist input file for Swift is read from Fortran unit 5 (standard input.) Printed output from Swift is written to Fortran unit 6 (standard output.) Files linked to Fortran units 1-3, 7, 8, 10, and 13-15 may be used in the execution of Swift, depending on input options. The files are described in Table 2 above.

All input files read using unformatted read statements, i.e., read(5,*), so you don't have to worry about getting the data in the right columns.

If *iopen=0* the files are not explicitly opened in the code. You must link the files to the appropriate Fortran unit manually under unix.

If *iopen=1* all files are opened using the default names given above. This will be most useful under Windows.

Note: It is also possible to input your own file names using the namelist input. Edit subroutine openfile.f and uncomment the one line that reads namelist 7 and recompile.

```
!      read (5,nl7)
```

Now add namelist & nl7 to your input file, and reset the prefix of any default file names using character strings, e.g.,

```
&nl7 grid='gold.xyz' q_in='gold.0050.q' &end
```

Any file names not reset retain their default names.

Ref. State	English Units	SI Units
P_{0r}	2116.8 lb_f/ft^2	1.0135 x 10^5 Pa
T_{0r}	519 R	288.3 K
c_{0r}	1116.7 ft/sec	340.39 m/sec
ρ_{0r}	.0765 lb_m/ft^3	1.2246 kg/m^3
$\rho_{0r}c_{0r}$	85.5057 $lb_m/sec/ft^2$	416.8416 $kg/sec/m^2$
μ_{0r}	1.285 x 10^{-5} $lb_m/(ft\ sec)$	1.91 x 10^{-5} kg/(m sec)
renr	6.65 x 10^6 [1/ft] 5.54 x 10^5 [1/in]	2.182 x 10^7 [1/m] 2.182 x 10^4 [1/mm]

Table 3 — Standard reference quantities usually used for nondimensionalization.

Nondimensionalization

The grid *xyz*-file may be input in arbitrary units of length. The input parameters to Swift and the variables in the output *q*-file are strictly nondimensional, with the exception of lengths which must have the same units as the grid.

All quantities are nondimensionalized by an arbitrary reference stagnation state defined by stagnation density ρ_{0r}, sonic velocity c_{0r}, and viscosity μ_{0r}, where μ_{0r} is defined at the stagnation temperature $T_{0r} = c_{0r}^2 / (\gamma R)$. Standard atmospheric conditions, given in table 2 above, are often used for the reference state. However, *any* self-consistent state may be used as long as the units of length are consistent with the grid units.

Input pressures and temperatures are nondimensionalized by P_{0r} and T_{0r}, respectively. Within the code pressures are usually nondimensionalized by $\rho_{0r}c_{0r}^2 = \Upsilon P_{0r}$. Inlet pressures and temperatures are nondimensionalized similarly, so that $P_{0in} = T_{0in} = 1$ for cases in which the inlet is at standard conditions. However, P_{0in} and T_{0in} can also be set arbitrarily using the initial condition input (see "Initial Condition Input", pp. 21) or a *qin* file (see "Inlet and Exit Profiles", pp. 27.) Input velocities are sometimes nondimensionalized by c_{0r} and sometimes input as a Mach number.

The reference state defines a reference Reynolds number *Renr* which must be input to Swift (see "&nam5 - Viscous Parameters", pp. 18.) *Renr* is given by $renr = \rho_{0r}c_{0r}/\mu_{0r}$ and has units of [1/grid units.] *Renr* remains the same for all cases with the same reference state and grid units.

Output quantities should be self explanatory, except for the mass flow. The mass flow may be output with the residual history (see "&nam6 - Output Control", pp. 20, variable *mioe*.) Mass flow is also output in the tables labeled "theta-averaged quantities," at the bottom of the column labeled "% mdot." In either case, the mass flow is nondimensionalized by $\rho_{0r}c_{0r}$ and has units of [grid units]^2. The mass flow through the full annulus is given (rather than mass flow per passage,) so that the printed mass flow should be constant through a multistage machine.

Calculations for liquids

Nondimensionalization

Nondimensionalize using conditions for liquids, but calculate c_{0r} as if for air.

R	$= 1716.58 \text{ ft}^2/(\text{sec}^2 \text{ R})$	ideal gas constant
γ	$= 1.4$	C_p / C_v
T_{0r}	$= 60 \text{ F} = 519 \text{ R}$	
c_{0r}	$= 1116.7 \text{ ft / sec}$	
ρ_{0r}	$= 62.37 \text{ lb}_m / \text{ft}^3$	
P_{0r}	$= \rho_{0r} R T_{0r} = 1{,}725{,}644 \text{ lb}_f / \text{ft}^2$	
ν_{0r}	$= 1.217 \times 10^{-5} \text{ ft}^2 / \text{sec}$	kinematic viscosity for water at 60 F
vispwr	$= 1$	laminar viscosity ~ T
renr	$= c_{0r} / \nu_{0r} = 7.646 \times 10^6 / \text{in}$	(convert to appropriate grid units)
om	$= \text{omega \{rad / sec\}} / c_{0r}$	(convert to appropriate grid units)

Initial Conditions

Calculate the inlet and exit velocities $V_{1,2}$ from the flow rate Q and areas A using:

$$V = Q / A$$

Approximate the Mach numbers for the initial conditions using:

$$M \approx V / c_{0r}$$

Calculate the total pressure rise ΔP_0 from the head rise H using:

$$H \Delta P_0 = \rho g H$$

Calculate the pressure ratios for the initial conditions using:

$$P_{02} / P_{01} = (P_{01} + \Delta P_0) / P_{01} .$$

Calculate the temperature ratios for the initial conditions using:

$$T_{02} / T_{01} = P_{02} / P_{01}$$

Calculate the static pressure rise ΔP using:

$$\Delta P = \Delta P_0 - 0.5 \rho (V_2^2 - V_1^2)$$

Calculate *prat* using:

$$prat = P_2 / P_{01} = (P_1 + \Delta P) / P_{01}$$

This should give a solution close to the correct flow rate, but you will probably have to run several cases with varying *prat* to get the flow rate exactly.

Running Swift

You will have to use preconditioning to get a converged solution, (see "Preconditioning", pp. 5.) It may be hard to get the preconditioning started, so run Swift 200 – 500 iterations with preconditioning turned off using:

icdup=0, nstg=2, avisc2=1, avisc4=1, cfl=2.5, eps=1.5, ibcinu=1, ipc=0.

Then restart with preconditioning turned on

icdup=0, nstg=2, avisc2=1, avisc4=1, cfl=2.5, eps=1.5, ibcinu=1, ipc=1, refmr=.15, pck=.30.

If you have stability problems or notice errors at the exit, set ibcext0=0 to use a simpler exit boundary condition.

The AUSM+ scheme should work well at low speeds, but I don't have much experience with it. Try

icdup=1, nstg=2, cfl=2.5, eps=1.5, ibcinu=1, ipc=1, refmr=.15, pck=.30, ausmk=0.3.

Swift Input

Namelist input is used for most variables. Many variables have defaults assigned and can be defaulted (not input.) Defaults are given in angle brackets, <Default=value> or <default.> If no default is given the value MUST be input.

Title

itile   An alphanumeric string of 80 characters or less printed to the output. The character string must be enclosed in single quotes.

** *&nam2* - Algorithm Parameters**

nstg   Number of stages for the Runge-Kutta scheme, usually 4, but can be 2-5. <default = 4>

ndis   Number of evaluations of artificial viscosity per stage. More than one evaluation usually improves robustness but increases CPU time. <default = 1>

                ndis > 1 gives 2 evaluations at stages 1 and 2 for *nstg = 4*.

                ndis > 1 gives 3 evaluations at stages 1, 3, and 5 for *nstg = 5*.

icdup   Flag for the type of differencing scheme.

                = 0   Central-difference schemes, requires *avisc2* and *avisc4*. <default>

                = 1   AUSM$^+$ scheme, requires *ausmk*, *refmr* and/or *refms*.

                = 2   H-CUSP scheme, requires *hcuspk*, *refmr* and/or *refms*.

cfl   Courant number, typically 5.6 (see Multistage Runge-Kutta Scheme, pp. 4.) If $ivtstp = 0$, *cfl* is the maximum Courant number, usually located somewhere near the leading edge at the blade surface. If $ivtstp = 1$, the Courant number will equal *cfl* everywhere. <default = 5.0>

avisc1   First-order artificial dissipation coefficient. <u>Not recommended</u>, but can sometimes be used to stabilize a solution that blows up at startup. Set $avisc1 = 1.$ for the first 50 or so iterations if necessary, but be sure to set $avisc1 = 0.$ as soon as the solution is running stably. (see "Artificial Viscosity", pp. 4.) <default = 0.0>

avisc2   Second-order artificial dissipation coefficient. Typically 0. - 2. Use 0. for purely subsonic flow or 1. for flows with shocks. <default = 0.5>

avisc4   Fourth-order artificial dissipation coefficient. Typically 0.25 - 1.5. Start at 1.0 and reduce *avisc4* to 0.5 if possible. <default = 0.5>

irs   Implicit residual smoothing flag. Usually = 1. (See Implicit Residual Smoothing, pp. 5.)

                = 0   No residual smoothing.

                = 1   Implicit smoothing after every Runge-Kutta stage <default.>

                = 2   Implicit smoothing after every other stage. *eps* must be increased for this option to work. Rarely used.

eps   Overall implicit smoothing coefficient based on the 1-D stability limit (see "Implicit Residual Smoothing", pp. 5) Swift will calculate the 1-D limit if *eps* is defaulted.

epi, epj, epk   Implicit smoothing coefficient multipliers for the *i, j,* and *k* directions. (see "Implicit Residual Smoothing", pp. 5) Rarely used. <default = 1.>

itmax   Number of iterations, typically 50-1000 per run, but 1000-3000 may be needed for a converged solution.

ivdt   Variable time step flag.

	$= 0$	Spatially constant time step.
	$= 1$	Spatially variable time step. <default, highly recommended>

ipc   Preconditioning flag, (see Preconditioning, pp. 5.)

         $= 0$ No preconditioning. <default>

         $= 1$ Preconditioning using the Merkel, Choi, Turkel scheme. Should give a substantial speedup for Mach numbers < 0.3.>

         $= 2$ Solves the equations using the preconditioning variable set, but sets the preconditioning matrix to the identity matrix. Used to debug the preconditioning routines.

refms, refmr   Reference relative Mach numbers M'_{ref} used for preconditioning, and the H-CUSP and AUSM$^+$ schemes. *Refms* is an absolute Mach number used for stators and *refmr* is a relative Mach number used for rotors. Should be approximately the largest Mach number expected in the flow. If the code blows up, try increasing *refm* by 0.1.

pck   Constant used to scale M'_{ref} for preconditioning (Turkel's parameter *k*.) The denominator in the preconditioning matrix is limited to be $> pck \times (M'_{ref})^2$. Typically 0.1 - 0.3. Smaller values may improve convergence, but larger values may be necessary for stability. <default = 0.15>

hcuspk   Constant used to scale M'_{ref} for the H-CUSP scheme. In the H-CUSP scheme the low-speed dissipation is scaled by $max(M', hcuspk \times M'_{ref})$, i.e., *hcuspk* sets the minimum value of dissipation. Typical values are 0.05 – 0.10. Smaller values may cause wiggles in the solution. Larger values may improve convergence but will increase predicted losses. <default = 0.05>

ausmk   Constant used to scale M'_{ref} for the AUSM$^+$ scheme. In the AUSM$^+$ scheme the numerical speed of sound is used to calculate the pressure fluxes and the pressure diffusion term. The numerical speed of sound is a function of a reference Mach number, $M_0^2 = min[1, max(M'^2, ausmk \times M'^2_{ref})]$, so ausmk also controls the dissipation of the scheme, but in a less obvious way than *hcuspk*. Typical values are 0.3 – 0.8. Larger values seem to be needed for convergence, but don't seem to hurt accuracy. <default = 0.8>

&nam3 - Boundary Condition & Code Control

     Note: In the following discussion, for linear geometries (igeom=0,) (v_m, v_θ, v_r) should be interpreted as (u, v, w).

Inlet boundary

     At the inlet boundary P_0 and T_0 are held constant. For subsonic flow a Riemann invariant based on v_m is extrapolated from the interior. Previous versions of Swift held v_θ constant and used a single flag, *ibcin*, to determine how v_r was calculated. *Ibcin* is retained for compatibility. If *ibcin* is defaulted, two flags, *ibcinv* and *ibcinw*, determine how v_θ and v_r are determined. Properties that are held constant are either generated from the initial condition data in the input file or are read directly from a qin-file.

ibcinu   Inlet boundary condition flag for v_m.

         $= 1$ Extrapolate the Riemann invariant to the inlet. Used for most problems. <default>

         $= 2$ Extrapolate v_m to the inlet. Recommended for low speed flows, especially with preconditioning.

ibcinv   Inlet boundary condition flag for v_θ.

         $= 1$ v_θ is held constant. <default>

	$= 2$	$v_\theta / v_m = \tan\alpha$ is held constant.

ibcinw   Inlet boundary condition flag for w v_r.

	$= 1$	v_r is held constant. \<default\>
	$= 2$	$v_r / v_m = \tan\phi$ is held constant.
	$= 3$	v_m is held tangent to the meridional grid lines at the inlet. \<default\>

ibcin   Old inlet boundary condition flag, still supported. For all options v_θ is held constant.

	$= 0$	or defaulted: ibcinv and ibcinw are used to set the options as described above.
	$= 1$	v_m is held tangent to the meridional grid lines at the inlet. \<default\>
	$= 2$	Supersonic **meridional** inflow - all quantities are held constant. (Rarely used except for the NASA supersonic throughflow fan project.)
	$= 3$	$v_r / v_m = \tan\phi$ is held constant.
	$= 4$	v_r is held constant. \<default\>

Exit Boundary

    Four primitive variables are extrapolated to the exit. The input parameter *prat* gives the exit pressure. The parameter *ipex* determines where *prat* is specified and determines how the spanwise pressure distribution is calculated.

ibcex   Exit boundary condition flag.

	$= 1$	*Prat* is specified as a constant. Only applicable to linear geometries, or annular geometries with radial outflow.
	$= 2$	Supersonic **meridional** outflow. *P* is extrapolated to the boundary. *Prat* is not used. (Rarely used except for the NASA supersonic throughflow fan project.)
	$= 3$	*Prat* is specified at the exit. The spanwise variation of $\bar{p}$ is found by solving radial equilibrium. p is constant blade-to-blade. \<default\>
	$= 4$	P*rat* is specified at the exit hub or tip. The spanwise variation of p is found by solving radial equilibrium. *P* is found as a perturbation about p using a characteristic boundary condition developed by Giles.

ibcext0

	$= 0$	Velocity components are extrapolated to the exit. Recommended for low speed flows, especially with preconditioning.
	$= 1$	Total enthalpy is extrapolated to the exit. Recommended for most problems, but may have stability or convergence problems at low speeds. \<default\>

ipex   Flag that tells where *prat* is specified. This can have a significant effect on the stability range of compressors. For hub-critical machines *ipex* should be set to 0 to hold the hub pressure constant. For tip-critical machines *ipex* should be set to 1 to hold the tip pressure constant.

    If *igeom = 0, prat* is held constant over the exit.

	$= 0$	*Prat* is specified at the hub \<default.\>
	$= -1$	*Prat* is specified at the tip.
	$= 1$	Exit pex-file is read from unit 14. (see "Inlet and Exit Profiles", pp. 17)

Inlet and Exit Profiles

Inlet profiles of P_0, v_θ, v_r and T_0, and exit profiles of p_{stat} can be specified as boundary conditions for Swift. For convenience, a common file format is used for both inlet and exit (see "Inlet and Exit Profiles", pp. 27.) The profiles are input as ASCII files containing six variables, P_0, v_x, v_θ, v_r, T_0, and p_{stat} at several spanwise locations. Only the variables needed at a particular boundary are used, and the other variables are ignored. The profiles are interpolated linearly to the span of the actual grid, and should resolve the endwall boundary layers.

Inlet and exit profile files for the current solution can be written by setting variable *ispan=1*. The output file, written to unit 15, can be edited to extract inlet or exit profiles that can used for subsequent calculations. In this way a multistage machine can be modeled one row at a time by using the exit profile from one blade row as the inlet profile to the next.

It may also be useful to modify output profiles manually, for example by replacing core flow quantities at a few points while retaining boundary layer properties.

ispan	Flag for writing spanwise profiles to unit 15.
	$= 0$ No output generated. \<default\>
	$= 1$ Spanwise profile output written to unit 15.
iqin	Flag for reading inlet profile.
	$= 0$ Inlet conditions are calculated by *subroutine qincalc* based on the initial condition data, boundary layer thicknesses, etc. in the input file. Current input values are used, so the inlet profiles can be changed at restart if desired. \<default\>
	$= 1$ Inlet qin-file read from unit 13. Used to read an exit profile from a solution of an upstream blade row.
ipex	Flag for reading exit pressure profile, also used to set location of *prat*. (see "Exit Boundary", pp. 16)
	$= 1$ Exit pex-file is read from unit 14.

Code Control

isymt	Top-plane symmetry flag. Used to model the bottom half of a linear cascade with bottom-to-top symmetry.
	$= 1$ Symmetry condition on *k = km.*
	else Solid wall boundary condition on *k = km.* \<default\>
kbcor	Obsolete flag for order of accuracy used in endwall boundary conditions. Still read for compatibility with old input files, but not used.
ires	Iteration increment for writing residuals in the output file. Typically 10. If the code is blowing up, set $ires = 1$ to print the size and location of the maximum residual at each iteration.
iresti	Flag for reading input restart file. Restart files are in PLOT3D format.
	$= 1$ Read restart file from unit 2.
	else No action taken. \<default\>
iresto	Flag for writing output restart file.
	$= 1$ Write restart file to unit 3. \<default\>
	else No action taken.
newkw	Flag for running the k-ω turbulence model from scratch using an unchanging solution. Useful for starting a new k-ω solution from an old Baldwin-Lomax solution.
	$= 0$ Run k-ω model and flow solver. \<default\>
	$= 1$ Run k-ω model from initial guess for *itmax* cycles. Write k-ω file to unit 8 and stop.
kwvars	Number of variables to store in the k-ω file. (See Turbulence Model k-w File, pp. 27.)

= 3   Stores 3 variables $[\mu_{tur}, k, \omega]$. Saves storage but not PLOT3D compatible.

= 5   Stores 5 variables $[\mu_{tur}, k, \omega, Re_t, \mu_{lam}]$. Increases storage but the k-ω file is PLOT3D compatible. <default = 5>

iopen   Flag for opening output files explicitly by name.

= 0 Output files are written to Fortran units without explicitly opening them. <default.>

= 1 Output files are opened by name:

grid.xyz   = main grid file (binary)

index.dat= Swift index file (ASCII)

qin.q

&nam4 - Flow Parameters

igeom   Flag for linear cascade or annular blade row.

= 0   Linear cascade.

= 1   Annular blade row <default.>

ga   Ratio of specific heats γ. <1.4 for air>

om   Normalized blade row rotational speed, Ω/c_0, where Ω is the wheel speed in radians per second, and c_0 has dimensions of [grid units/sec], giving *om* dimensions of [1/grid units]. The *(x,y,z)* coordinate system must be right-handed. Looking in the positive *x*-direction, clockwise rotation is negative and counterclockwise rotation is positive. Ω is negative for most Glenn geometries. <default = 0.>

prat   Ratio of the exit static pressure to the reference total pressure, $prat = p_{exit}/P_{0r}$.

expt   Exponent used to specify the inlet whirl distribution. $M_\theta = \overline{M_\theta}(r/r_{mid})^{expt}$ where M_θ is the mid-span value of M_θ determined from the initial condition input.

= 0   Gives uniform M_θ except within the endwall boundary layer. <default>

= -1   Gives free vortex inflow.

= 1   Gives forced vortex inflow.

&nam5 - Viscous Parameters

*il*t   Inviscid, Laminar, or Turbulent analysis.

= 0   Inviscid. Most other viscous parameters are not used if *ilt*=0.

= 1   Laminar.

= 2   Turbulent using the Baldwin-Lomax turbulence model. <default>

= 3   Turbulent using the Cebeci-Smith turbulence model. This model works well for turbine heat transfer but may overpredict losses for transonic compressors.

= 4   Fully turbulent using the Wilcox baseline k-ω turbulence model.

= 5   Turbulent with transition using the Wilcox low Reynolds number k-ω turbulence model. Note that "low Reynolds number model" refers to modifications made to give reasonable calculations of flat plate transition, and **not** to near-wall modifications needed by k-ε models.

isst	$= 0$ Standard k-ω model if *ilt* = 4 or 5.
	$= 1$ Menter's shear stress transport (SST) k-ω model if *ilt* = 4 or 5. Menter's model has two parts. The first part is based on Bradshaw's assumption that the shear stress in a boundary layer is proportional to *k*. Menter showed that this could be added to the standard k-ω model as a modification to the turbulent viscosity near walls. This part of the model is included in Swift, and has given good results when used with the AUSM scheme for a transonic compressor. The second part of Menter's SST model blends the k-ω and k-ε schemes and is not included in Swift.
itur	The turbulence model is updated every *itur* iterations. Recommended values are *itur=5* for the Baldwin-Lomax or Cebeci-Smith models, and *itur=2* for the k-ω model. If the k-ω model blows up quickly it may help to use *itur=1* for the first 100-200 iterations. <default=5>
renr	Reynolds number per unit length based on reference conditions, $renr = \rho_{0r}c_{0r}/\mu_{0r}$. Must have units of $[1/\text{grid units}]$. Generally much larger that a conventional "free-stream" Reynolds number. For example, for standard conditions:

$$renr = 0.0765\left(\frac{\text{lb}_m}{\text{ft}^3}\right) \times 1116.7\left(\frac{\text{ft}}{\text{sec}}\right)/1.285\times10^{-5}\left(\frac{\text{lb}_m}{\text{ft sec}}\right)$$

$$= 6.65\times10^6/\text{ft}$$

prnr	Prandtl number. <default = 0.7 for air>
tw	Normalized wall temperature, $tw = T_{\text{wall}}/T_0$.
	$= 0$ Adiabatic wall boundary conditions are used.
	$= 1$ $T_{\text{wall}} = T_0$ <default>
	else $T_{\text{wall}} = tw$.
vispwr	Exponent for laminar viscosity power law. <default = 0.667 for air> Use *vispwr=0.0* for water.

$$\mu/\mu_{0r} = (T/T_{0r})^{vispwr}$$

prtr	Turbulent Prandtl number. <default = 0.9>
cmutm	Value of $\mu_{\text{turb}}/\mu_{\text{lam}}$ at which transition is assumed to occur. Baldwin and Lomax recommend 14. Can be increased to move transition downstream or vice-versa. If *cmutm* = 0, the flow is fully turbulent. <default = 14.>
jedge	j-index where the artificial viscosity begins to ramp off near the blade. Also the last j-index searched for the blade turbulent length scale. For the Baldwin-Lomax turbulence model $(ilt = 2)$, *jedge* should be a grid line slightly bigger than the largest expected blade boundary layer. For the Cebeci-Smith turbulence model $(ilt = 3)$, *jedge* should be a grid line slightly bigger than half the largest expected blade boundary layer. <default = 10>
kedgh, kedgt	k-indices where the artificial viscosity begins to ramp off near the hub and tip. Also the last k-indices searched for the hub and tip turbulent length scales. See comments for *jedge*. <default = 10>
iltin	Flag controlling inlet velocity and P_0 profiles.
	$= 0$ Inviscid.
	$= 1$ Laminar.

| | = 2 | Turbulent using Cole's wall-wake profile. <default> |

dblh, dblt   Inlet hub and tip boundary layer thicknesses in grid units.

xrle, xrte   Axial locations at which the hub starts and stops rotating. Rotational boundary conditions are applied on the hub for $xrle < x < xrte$. Stationary conditions are applied elsewhere. Note that *xrle* and *xrte* may not be sufficient to locate the rotating part of the hub in a radial flow machine.Defaults are set to make the entire hub rotate.

irle, irte   i-indices at which the hub starts and stops rotating. In radial flow machines *xrle* and *xrte* may not be sufficient to locate the rotating part of the hub, so *irle* and *irte* can be used instead. This option only works correctly for a single block H-grid, and even then the grid lines may not be straight. Defaults are set to make the entire hub rotate.

tintens   Free-stream turbulence intensity written as a decimal. Used to get the inlet value of k for the k-ω model. <default=0.01, i.e., one percent.>

tlength   Free-stream turbulence length scale. Used to get the inlet value of ω for the k-ω model. Typically $tlength \approx 0.03 \times$ boundary layer height or $tlength \approx 0.001 \times$ pitch. The free-stream turbulent viscosity μ_{tur} is derived from *tintens* and *tlength*, and is printed near the top of the output. Generally μ_{tur} should be ≤ 0.1 to minimize the effects of *tlength*.

hrough   Surface roughness height in grid units. In the Baldwin-Lomax and Cebeci-Smith turbulence models (*ilt* = 2 or 3 respectively) surface roughness is modeled using the Cebeci-Chang roughness model. In the k-ω models (*ilt* = 4 or 5) surface roughness is modeled using Wilcox's wall boundary condition for ω. In any case *hrough* represents some equivalent sand grain roughness height, which is a factor of 2–4 times the RMS height. Smooth surfaces are modeled by setting *hrough* = 0. <default = 0.>

Note: Previous versions of Swift input *hrough* in turbulent wall units (h^+). In Swift version 300 the actual height is used. If $hrough > 4$ Swift assumes old input was used and resets *hrough* to zero.

&nam6 - Output Control

oar   Flag for frame of reference of output q-file. Swift automatically detects the frame of reference of a restart q-file and converts it to the absolute frame for internal use if necessary.

 | = 0. | All blade rows are in the absolute frame of reference. |

 | = 1. | All blade rows are in the relative frame of reference. <default> |

mioe   Flag for output format of mass flow in residual history. For transonic fans the inflow may respond slowly to a change in back pressure, so the inlet mass flow can be monitored for convergence. For turbines the inflow may choke quickly so the outflow can be monitored. In general the mass flow error is a good measure of convergence and accuracy and should converge to a fraction of a percent (e. g., < 0.003).

 | = 1 | Inlet mass flow history is written. |

 | = 2 | Exit mass flow history is written. |

 | = 3 | Mass flow error $1 - \dot{m}_{out}/\dot{m}_{in}$ is written. <default> |

 | = 4 | Eliminates the maximum residual (which always looks like the RMS residual anyway) and prints $\dot{m}_{in}$ and $\dot{m}_{out}$ instead. Calculate the mass flow error yourself with EXCEL if desired. |

iqav   Flag controlling type of θ-averaging used in the output.

 | = 0 | Energy average. Mass average of [ρ, V^2, and T]. Usually the most optimistic average.<default> |

	= 1	Momentum average. Mass average of [ρ, ρV, and e], fairly conservative.
	= 2	Mixed-out average. Formal average of inviscid fluxes gives properties far downstream. Usually the most conservative average.
	= 3	Total pressure average. Converts P_0 to an equivalent T_0, mass averages, then converts back. Often done with experimental data. Usually similar to the energy average.

nko   Number of k-indices for blade surface output, max = 10. <default = 0>

ko   Array of *nko* k-indices separated by commas where blade surface output is desired. <default = 0>

iog   Grid block number where spanwise output is desired. Spanwise output is normally printed at the inlet and exit of each blade grid. Sometimes it is useful to have output from other cross-channel planes, say near the trailing edge. This output can be generated for a single grid block by specifying the block number *iog* and the desired i-indices *io*. <default = 1>

io   Array of up to 10 i-indices where spanwise output is desired. For H-grids output is printed at each i index. For C-grids the i-index and its periodic neighbor are merged. <default = 0>

ismout   Flag for distance coordinate s used in blade surface output.

           = 0   S = arc length around blade. <default>

           else   S = meridional distance along blade.

ileout   Flag for location of leading edge (s=0) used in blade surface output.

           = 0   S = 0 at i = imax/2 index. <default>

           else   S = 0 at smax/2.

           Note: For H-grids, S = 0 at i = ile.

Initial Condition Input

Immediately following the namelist input comes *nrow+2* lines of data used for the initial guess and the inlet boundary conditions. *Nrow* is the number of blade rows and is determined within Swift by counting the number of lines of input.

The first line is ignored and can be used for column labels. Subsequent lines give row number and nominal flow conditions at mid-span. Unformatted reads are used, so all variables must be input.

A sample initial condition input for a seven-block grid is shown in figure 5. A portion of the input is repeated below.

```
row         P0         Mx         Mt         Mr          T0
  0     1.0000      .1330    -.0000       0.      1.0000
  1      .9938      .1692    -.3986       0.      1.0000
etc.
```

The variables are as follows:

row   Integer blade row number. Row number 0 is the inlet. Subsequent row numbers represent the exits of each blade row.

P0   Meanline P_0/P_{0r}.

Mx   Meanline Mach number in the x-direction.

Mt   Meanline Mach number in the θ- or y-direction.

Mr   Meanline Mach number in the r- or z-direction.

T0   Meanline T_0/T_{0r}.

Index File

The index file is an ASCII file that gives the grid sizes, connectivity, and certain boundary condition information for each grid. It replaces namelist block &nl1 in RVC3D.

The first line is ignored and can be used for column labels. Subsequent lines give grid type, dimensions, key indices (like *itl* and *iil* in RVC3D), connecting grid numbers, blade row number, and relative rotational rate for each grid. Negative values are sometimes used to toggle boundary condition options. One line is required for each grid. Unformatted reads are used, so all variables must be input.

For an isolated blade row, TCGRID will produce a complete index file written to unit 10. It may be necessary to modify *nhub* or *ntip* if the simple periodicity clearance model is to be used, or to modify the rotation multipliers *om, omh,* or *omt.*

For multistage calculations the grids for each blade row are generated separately, and merged using the utility code MULTIX. The separate index files must be merged manually using the unix *cat* command or an editor. Extraneous header lines must be removed, and connectivity information must be added manually.

A sample index file for a seven-block grid is shown in figure 5. A portion of the file is repeated below.

```
     grid type    im    jm    km    i1    i2    i3   nin   nex nhub ntip nlr row   om omh
omt
        1    1    17    17    57     0     0     0   999     2    0    0    0   1   0.   0.   0.
        2    2   127    37    57    14    57     0     1    -3    0    0    0   1   0.   0.   0.
     etc.
```

Index File Variables

grid	Grid (block) number, from 1 to number of grids
type	Flag giving type of grid.

> = 1   H grid for upstream
>
> = 2   C grid for blades
>
> = 3   O grid for hub or tip clearances

im	Number of grid points in i-direction.
jm	Number of grid points in j-direction.
km	Number of grid points in k-direction.
i1 (C-grid)	Lower i-index of trailing edge. Upper index is assumed to be periodic.
i1 (H-grid)	Leading-edge index of an H-grid, or flag for the type of H-grid geometry. **Note:** Swift.300 supports inlet H-grids ahead of C-grids, H-grids in blade passages, and H-grids in rectangular ducts.

> = 0 – This is an inlet H-grid ahead of a C-grid
>
> = 1 – This is an H-grid in a rectangular duct
>
> > 1 – i-index of the leading edge for an H-grid around a blade

i2 (C-grid)	Lower i-index of inlet (or last periodic point.) Upper index is assumed to be periodic.
i2 (H-grid)	i-index of the trailing edge for an H-grid around a blade
i3	Unused, set to 0.
nin	Inlet boundary condition flag.

> = 999:   C- or H-grid with conventional inlet boundary condition.
>
> > 0:   C-grid inlet patched to upstream H-grid number nin.
>
> < 0:   C-grid inlet mixed-out from upstream C-grid number nin

nex	Exit boundary condition flag.
	= 999: C-grid with conventional exit boundary condition.
	> 0: H-grid exit patched to downstream C-grid number nex.
	< 0: C-grid exit mixed out to downstream C-grid number nex.
nhub	Flag for hub clearance. **Note:** In Swift.300 hub and tip clearances can be modeled simultaneously.
	> 0: Connecting grid block number for gridded hub clearance.
	= 0: No hub clearance.
	< 0: Simple periodicity hub clearance model between k=1 and k=|nhub|.
ntip	Flag for tip clearance. **Note:** In Swift.300 hub and tip clearances can be modeled simultaneously.
	> 0: Connecting grid block number for gridded tip clearance.
	= 0: No tip clearance.
	< 0: Simple periodicity tip clearance model between k=|ntip| and k=km.
nlr	Unused, set to 0.
row	Integer blade row number between 1 and the number of rows. Corresponds to row number in initial condition input.
om(n)	Rotation multiplier. The rotational speed for this grid is $om \times om(n)$. Usually 0.0 for stators, 1.0 for rotors, or -1.0 for counterrotating rotors. (See &nam5 - Viscous Parameters, pp. 18 for definition of normalized blade row rotational speed *om*.)
omh	Hub rotation multiplier. Rotational speed for k=1 on this grid is $om \times omh$. Usually 1.0 for rotating hubs. Overridden by variables *xrle* and *xrte*, the axial locations at which the hub starts and stops rotating (see "&nam5 - Viscous Parameters", pp. 18.)
omt	Tip rotation multiplier. Rotational speed for k=km on this grid is $om \times omt$. Usually 0.0 for stationary shrouds or 1.0 for rotating shrouds.

Swift Output

Printed output from Swift is written to Fortran unit 6 (standard output.) The output is divided into several sections. The sections may be separated using an editor and plotted using any x-y plotting package that can read ASCII column data.

1. The input variables are echoed back for reference, and any comments or warnings regarding the input are given.
2. Spanwise profiles of θ – averaged flow variables are given at the inlet or exit. These variables are either based on the initial guess or on a restart file, depending on how the code is started. The initial profiles are often useful for identifying grid lines near endwall boundary layers.
3. A convergence history gives maximum and RMS residuals of density, and exit flow properties versus iteration.
4. Spanwise profiles of θ – averaged flow variables are repeated at the inlet and exit for the new solution. Four different averaging schemes are available for computing these profiles. Note that all quantities are evaluated locally, except for the columns labeled "P0 loss" (for stators) or "ad.eff" (for rotors.) These quantities are calculated between the grid inlet and grid exit of each blade row.
5. Blade surface profiles of various quantities are given on selected k grid lines (spanwise locations.) Values of y^+ for the first grid point are given for checking turbulent grid spacing, and maximum values of μ_T are given to identify transition points.

Test Cases

Each test case is stored in a separate directory. Files named *.int are the TCGRID input files. Files named *.ind are Swift index files. Files named *.jcl are unix script files for linking files and running Swift. These files also contain the input data, and may contain comments about how to run the case. Files named *.ins are the raw Swift input if you need to run manually. Files named *.out are the my output files for the case. In some cases I have included the experimental data shown in the plots. This should be self explanatory.

Goldman Turbine Vane

The first test case is for an annular turbine vane tested by Goldman at NASA Glenn Research Center. Computed results were presented previously in references 2, 4, and 13. Current results are shown in figure 6.

The grid is shown at the top left of the figure. The grid size was $97 \times 32 \times 33$, for a total of 102,432 points. The grid size is intentionally coarse is to make the test case run quickly, but the results are still very good.

Solutions were run using the C-D, H-CUSP, and AUSM$^+$ schemes and the Baldwin-Lomax turbulence model. The standard four-stage Runge-Kutta scheme was used with cfl=5.6. The calculations were run 1500 iterations. On an 300 MHz SGI Octane, the C-D solution took about 30 minutes, the H-CUSP solution took about 32 minutes, and the AUSM+ solution took about 45 minutes.

Convergence histories for the C-D solution are shown at the bottom left. The other schemes behave similarly. Exit total pressure converges to three significant digits in about 1000 iterations, and mass flow error $1 - \dot{m}_{out}/\dot{m}_{in}$ converges to about 0.001.

Mach contours at midspan are shown at the top right. The flow is entirely subsonic. The thin boundary layers and wake are evident.

Comparisons with experimentally-measured exit profiles are shown at the bottom right. For total pressure loss, the C-D results show little detail along the span. The H-CUSP results show some detail near the tip but too much loss near the hub. The AUSM$^+$ results show good qualitative agreement with the data along the entire span. All results show higher losses than the data at midspan. The midspan loss does not improve with increasing grid resolution, and may be due to poor modeling of the round trailing edge. For flow angle, the C-D results show nearly uniform flow angle along the span, and the H-CUSP results are only slightly better. The AUSM$^+$ results show excellent agreement with the data along the entire span.

NASA Rotor 67

The second test case is for a low aspect ratio transonic fan denoted NASA rotor 67 that was also tested at NASA Glenn Research Center. Solutions were computed with three different grids: a C-grid, an H-grid, and a multiblock H-C-O grid that resolved the tip clearance. All grids had the same wall spacings and spanwise distributions. All results were computed using the central-difference scheme, but different clearance and turbulence models were used. Computed results have been presented previously in ref. 3. Current results are shown in figure 7.

Multi-block Grid

Multi-block calculations were made using a three-block grid that resolved the tip clearance, as shown at the top left of figure 7. The upstream H-grid had $25 \times 19 \times 49$ points, the C-grid around the blade had $187 \times 37 \times 49$ points, and the O-grid in the tip clearance region had $147 \times 11 \times 7$ points, for a total of 373,625 points. The inlet boundary was approximately one axial chord upstream of the leading edge. The Baldwin-Lomax turbulence model was used for these calculations. The calculations were run 1500 iterations on an SGI Origin 2000 system with 6 processors. The wallclock time was about 1.1 hours.

C-grid

Single-block calculations were made using the same C-grid that was used for the multi-block calculations. The grid size was 339,031 points. The inlet boundary was approximately 0.23 axial chords upstream of the leading edge. The simple periodicity tip clearance model and the k-ω turbulence model were used. The calculations were run 1500 iterations on an SGI Origin 2000 system with 6 processors. The wallclock time was about 1.25 hours because of the k-ω model.

H-grid

Single-block calculations were also made using an H-grid with $151 \times 54 \times 49 = 399,546$ points. H-grids need fewer i-direction points than C-grids, but need more j-direction points to resolve the blade boundary layers. Here the inlet boundary was approximately 0.76 axial chords upstream of the leading edge. The simple periodicity tip clearance model and the k-ω turbulence model were also used here. The calculations were run 1500 iterations on an SGI Origin 2000 system with 8 processors. The wallclock time was about 1.6 hours.

Convergence histories are shown at the bottom left. The three grids converge quite differently, but in each case inlet mass flow and exit total pressure converge to three significant digits in about 1000 iterations.

The pressure ratio prat was chosen to give an operating point near peak efficiency. Mach contours at 90 percent span are shown at the top right. The bow shock system and passage shock can be seen.

Comparisons with experimentally-measured exit profiles are shown at the bottom right. All exit profiles are in good agreement with the experimental data. The results are nearly independent of the grid topology, but do depend on the turbulence model. The Baldwin-Lomax model (3-block grid) predicts somewhat higher pressure ratios and efficiencies than the k-ω model (C- and H-grids.) The k-ω SST may give better results for this problem.

Large Low Speed Centrifugal Compressor

The third test cast is for the large low speed centrifugal impeller tested at NASA GRC by Hathaway, et. al. The original calculations were shown in reference 7. Current results are shown in figure 8. The current results used an H-grid with $127 \times 48 \times 41 = 249,936$ points, shown at the top left. The simple periodicity tip clearance model and Baldwin-Lomax turbulence model were used.

The solution was run with the standard C-D scheme for 500 iterations to get the solution started. Then preconditioning was turned on since the flow is relatively low speed. Convergence histories shown at the bottom left show that the pressure ratio and mass flow error were converged in 2500 iterations. The total wallclock time was about 1.1 hours on an SGI Origin 2000 system with 6 processors.

The experimental results were measured at a flow rate of 66.14 lb/sec. The computed mass flow was 67.43 lb/sec, about two percent high. Computed surface pressure contours are shown at the top right. A comparison of computed and measured blade pressures at mid span is shown at the bottom right. The suction surface pressure distribution is in excellent agreement with the data, but the pressure surface pressures are high past mid chord and far off at the trailing edge. The discrepancy at the trailing edge was noted in ref. 7 and is still unexplained.

Single Stage Turbine

The last test case is for a two-stage turbine tested by Dunn, et. all. The original calculations were shown in reference 8. The current results were done for the first stage only and are shown in figure 9. A three-block grid similar to the one shown in figure 5 was used. The stator grid had $127 \times 37 \times 45 = 211,455$ points. The rotor grid had $127 \times 33 \times 45 = 188,595$ points. And the rotor clearance grid had $95 \times 13 \times 11 = 13,585$ points, for a total of 413,635 points.

For this test case the stator and rotor grids are generated separated, and linked with a utility named multix. The source code multix.f is included. The file run_multix.exe is a unix script that will do almost everything for you. It is liberally commented if you want to do this manually. The file out.ind is the preliminary index file generated by multix. The file stg1.ind has been manually edited to add grid connectivity, inlet and exit flags, hub and tip rotation multipliers, etc. Notice the differences between the two *.ind files to see what you may need to add for other multistage calculations.

The solution was run 200 iterations without preconditioning, then 1800 iterations with preconditioning turned on. The wall temperature ratio was set to 0.7 so that heat transfer calculations could be made. The Cebeci-Smith turbulence model was used. Convergence histories at the bottom left of the figure show that pressure ratio and mass flow error were both converged very well in about 1200 iterations.

Computed surface pressures are shown at the top left. A comparison of computed and measured pressures on the stator at mid span is shown at the top right. The agreement is very good.

Comparisons of computed and measured Stanton numbers (heat transfer coefficients) are shown at the bottom right. Computed Stanton numbers are somewhat high for the stator, but are in very good agreement with the data for the rotor.

File Descriptions

Grid *XYZ*-File

Grids are stored using standard PLOT3D xyz-file format. Grids can be read with the following Fortran code:

```
c       read grid coordinates
        read(1)im,jm,km
        read(1)(((x(i,j,k),i=1,im),j=1,jm),k=1,km),
     &         (((y(i,j,k),i=1,im),j=1,jm),k=1,km),
     &         (((z(i,j,k),i=1,im),j=1,jm),k=1,km)
```

Solution Q-File

Solution files are stored in standard PLOT3D q-file format. Solution files can be read with the following Fortran code:

```
c       read q-file
        read(2)im,jm,km
        read(2)eminf,aldeg,renr,time
        read(2)((((qq(l,i,j,k),i=1,im),j=1,jm),k=1,km),l=1,5)

c       additional geometry data and residual history
        read(2)itl,iil,phdeg,ga,om,nres,igeom,dum,dum,dum
        read(2)((resd(n,l),n=1,nres),l=1,5)
```

The q-variables are:

$$q = \left[\frac{\rho}{\rho_{0r}}, \frac{\rho u}{\rho_{0r}c_{0r}}, \frac{\rho v}{\rho_{0r}c_{0r}}, \frac{\rho w}{\rho_{0r}c_{0r}}, \frac{e}{\rho_{0r}c_{0r}^2} \right]$$

$$e = \rho\left(C_v T + \frac{1}{2}(u^2 + v^2 + w^2) \right)$$

If *oar=1* the relative velocity components are stored, $v' = v - \Omega z$, $w' = w + \Omega z$.

Turbulence Model k-ω File

Restart files for the k-ω turbulence model are stored in standard PLOT3D q-file format. Solution files can be read with the following Fortran code:

```
c       read tmu, k, w
        read(7)im,jm,km
        read(7)dum
        read(7)(((((tkw(l,i,j,k),i=1,im),j=1,jm),k=1,km),l=1,kwvars)
```

The tkw-variables are:

$$tkw = \left[\frac{\mu_{tur}}{\mu_{0r}}, \frac{k}{c_{0r}^2}, \frac{\omega}{c_{0r}}, \frac{\mu_{lam}}{\mu_{0r}}, Re_{tur} \right]$$

Note that the laminar viscosity μ_{lam} and the turbulence Reynolds number Re_{tur} are not used by Swift. They are written to pad the file for PLOT3D compatibility if *kwvars=5*. This results in larger file sizes than necessary. Smaller files may be generated by setting *kwvars=3*, but the files cannot be read by PLOT3D.

Inlet and Exit Profiles

Inlet profiles of P_0, v_θ, v_r and T_0, and exit profiles of p_{stat} can be specified as boundary conditions for Swift. For convenience, a common file format is used for both inlet and exit. The profiles are input as ASCII files containing six variables, P_0, v_x, v_θ, v_r, T_0, and p_{stat} at several spanwise locations. Only the variables needed at a particular boundary are used, and the

other variables are ignored. The profiles are interpolated linearly to the span of the actual grid, and should resolve any desired endwall boundary layers.

A sample profile file can be generated by setting variable *ispan=1*. The output written to unit 15 may be edited manually to extract the desired profile. The format is as follows:

```
&ospan irow = 0 kin = 95 flow = 119.25082 &end
k     s/span     P0/P0i      vx/c0     vth/c0      vr/c0      T0/T0i     ps/P0i
1    0.00000    1.12907    0.00000   -0.65789    0.00000    1.06326    0.83876
2    0.00019    0.90181   -0.07050   -0.31211   -0.01668    1.00000    0.83864
etc.
```

The first line is namelist input. Only *kin* is required.

kin	Number of spanwise points.
irow	Dummy variable not used by Swift, but useful for identifying the desired profile from an output file. *Irow* gives the relative location of the profile, where *irow=0* is the inlet, *irow=1* is the exit of the first blade row, *irow=2* is the exit of the second blade row, etc.
flow	Dummy variable not used by Swift. *Flow* is the non-dimensional mass flow and is included for use by the CSTALL code now under development.

The second line has titles for convenience but is not read. The remaining *kin* lines have the following variables:

k	Spanwise index, not used.
s/span	Normalized spanwise distance, between 0.0 at the hub to 1.0 at the tip.
P0/P0i	Normalized total pressure, used for inlet profiles only.
vx/c0	Normalized axial velocity, not used.
vth/c0	Normalized tangential velocity, used for inlet profiles only.
vr/c0	Normalized radial velocity, used for inlet profiles only.
ps/p0i	Normalized static pressure, used for exit profiles only.

References

1. Baldwin, B. S., and Lomax, H., "Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows," AIAA Paper 78-257, Jan. 1978.

2. Chima, R. V., Yokota, J. W., "Numerical Analysis of Three- Dimensional Viscous Flows in Turbomachinery," AIAA J., Vol. 28, No. 5, May 1990, pp. 798-806.

3. Chima, R. V., "Viscous Three-Dimensional Calculations of Transonic Fan Performance," in CFD Techniques for Propulsion Applications, AGARD Conference Proceedings No. CP-510, AGARD, Neuilly-Sur-Seine, France, Feb. 1992, pp 21-1 to 21-19. Also NASA TM-103800.

4. Chima, R. V., Giel, P. W., and Boyle, R. J., "An Algebraic Turbulence Model for Three-Dimensional Viscous Flows," in *Engineering Turbulence Modeling and Experiments 2*, Rodi, W. and Martelli, F. editors, Elsevier pub. N. Y., 1993, pp. 775-784. Also NASA TM-105931.

5. Chima, R. V., "A k-ω Turbulence Model for Quasi-Three-Dimensional Turbomachinery Flows," AIAA Paper 96-0248, Jan. 1996. Also NASA TM-107051.

6. Chima, R. V., "Calculation of Tip-Clearance Effects in a Transonic Compressor Rotor," J. Turbomachinery, Vol. 120, Jan. 1998, pp. 131- 140. Also NASA TM107216.

7. Tweedt, D. L., Chima, R. V., and Turkel, E., "Preconditioning for Numerical Simulation of Low Mach Number Three-Dimensional Viscous Turbomachinery Flows," AIAA Paper 97-1828, June, 1997. Also NASA TM-113120.

8. Chima, R. V., "Calculation of Multistage Turbomachinery Using Steady Characteristic Boundary Conditions," AIAA Paper 98-0968. Also NASA TM-1998-206613.

9. Chima, R. V., "TCGRID 3-D Grid Generator for Turbomachinery - User's Manual and Documentation," Dec. 1999, available from the author.

10. 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, June 1981

11. Sorenson, R. L., "A Computer Program to Generate Two- Dimensional Grids About Airfoils and Other Shapes by Use of Poisson's Equation," NASA TM-81198, 1980.

12. Turkel, E., "A Review of Preconditioning Methods for Fluid Dynamics," *Applied Numerical Mathematics*, Vol. 12, 1993, pp. 257-284.

13. Chima, R. V., and Liou, M.-S., "Comparison of the AUSM+ and H-CUSP Schemes for Turbomachinery Applications," AIAA Paper AIAA-2003-4120, June, 2003. Also NASA TM-2003-212457.
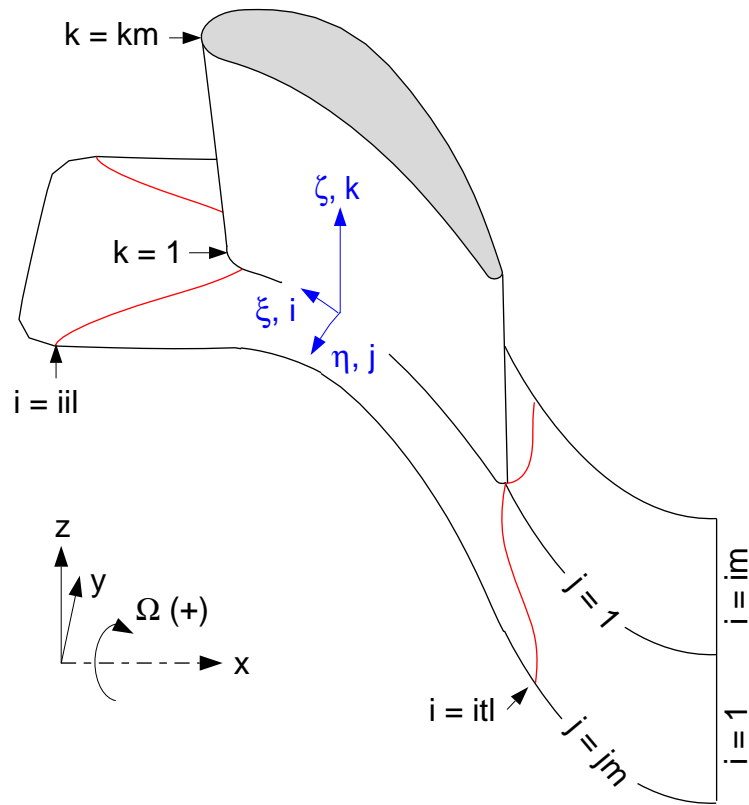
Figure 1 — 3-D coordinate system and grid indexing system
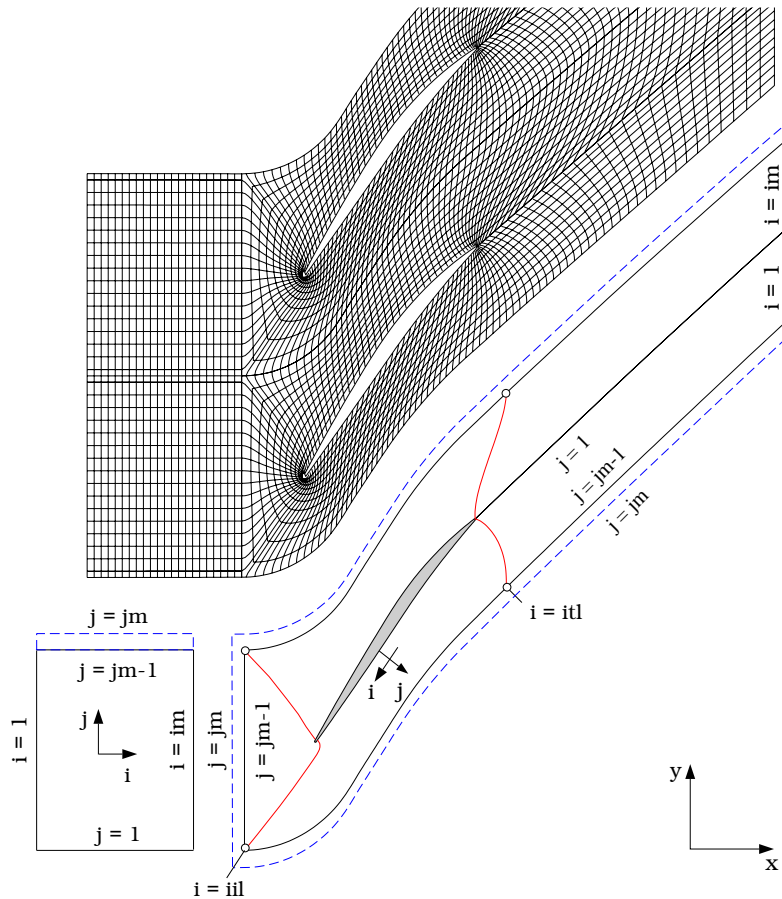
Figure 2 — Indexing system for a C-grid and an upstream H-grid

30
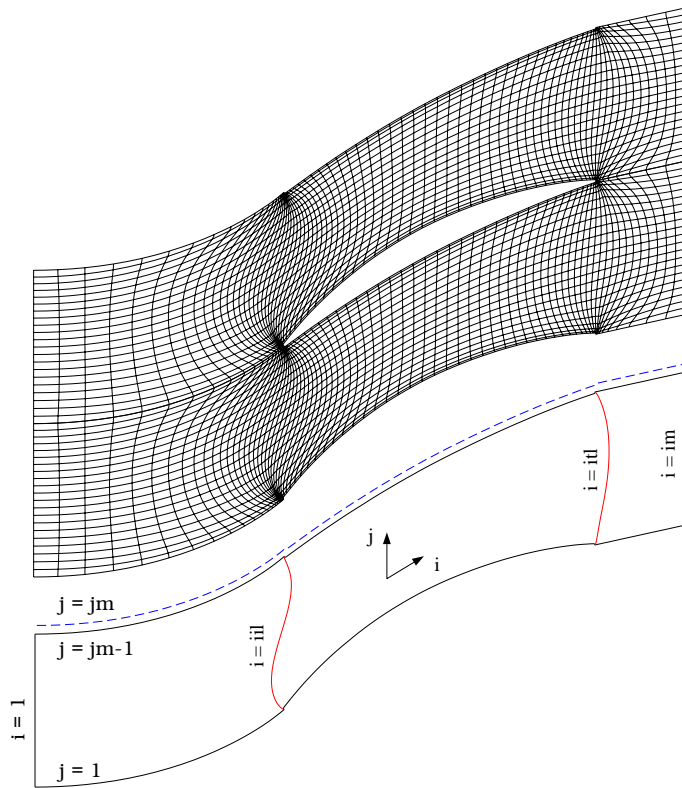
Figure 3 —   Indexing system for an H-grid
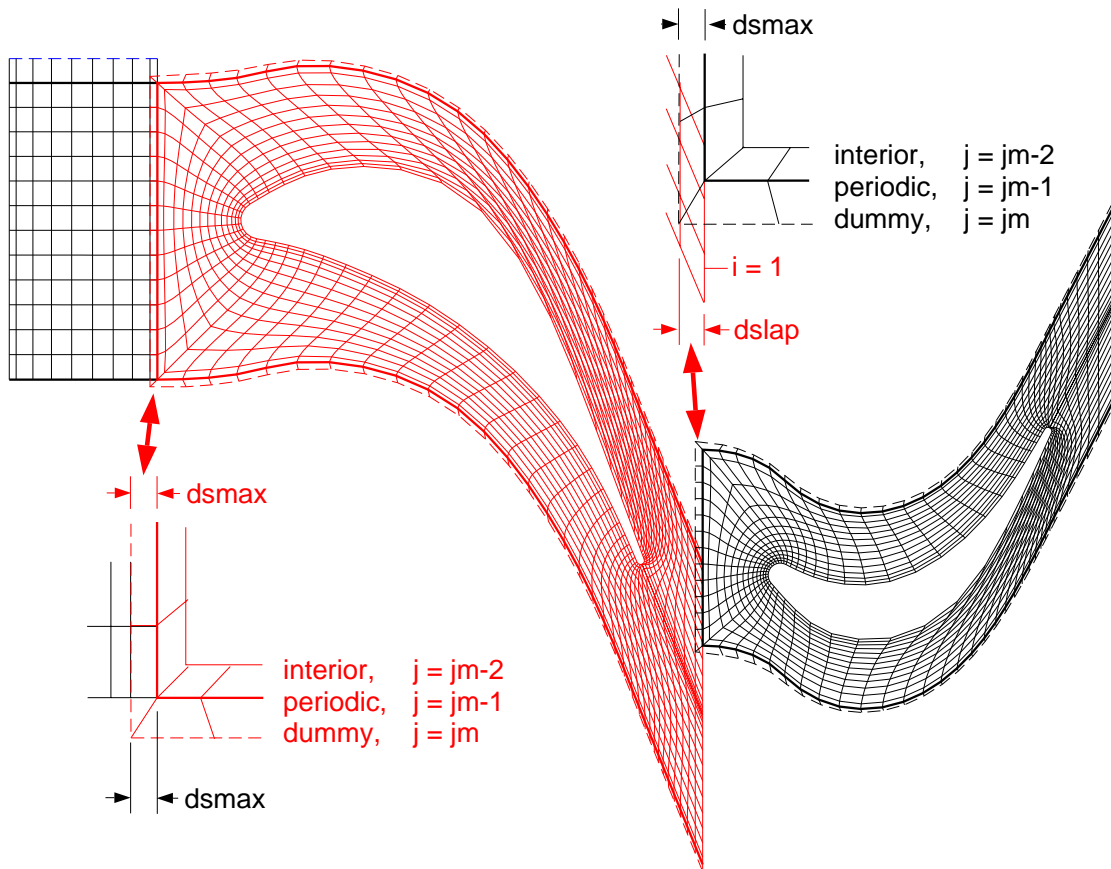
Figure 4 —   Three-block grid for a turbine stage showing overlap regions and dummy grid lines

31
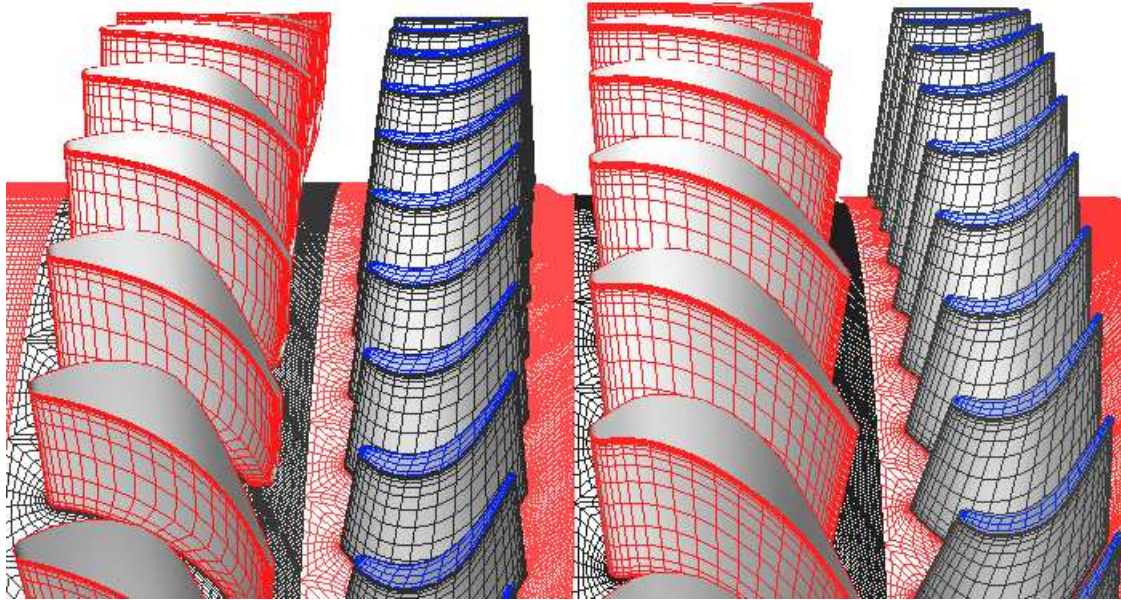
Seven-block grid for a two-stage turbine with rotor tip clearances.
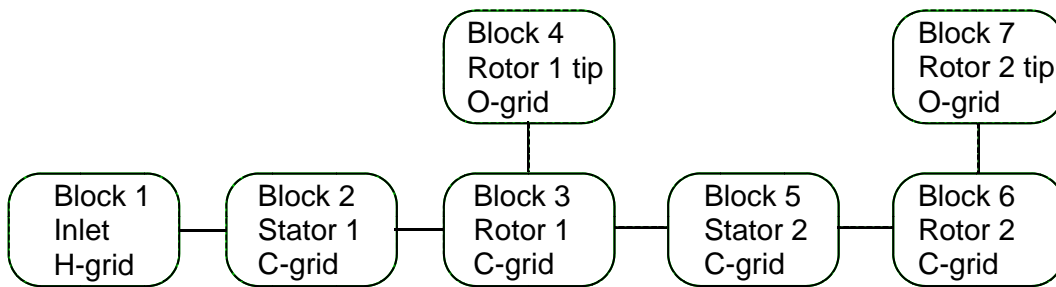
Block diagram of a seven block grid..

```
grid type   im   jm   km   i1   i2   i3   nin   nex nhub ntip nlr row   om omh omt
   1    1   17   17   57    0    0    0   999     2    0    0    0   1  0.  0.  0.
   2    2  127   37   57   14   57    0     1    -3    0    0    0   1  0.  0.  0.
   3    2  127   33   57   17   57   45    -2    -5    0    4    0   2  1.  1.  0.
   4    3   95   13   13    0    0   45     0     0    0    3    0   2  1.  1.  0.
   5    2  127   37   57   14   55    0    -3    -6    0    0    0   3  0.  0.  0.
   6    2  141   33   57   21   64   45    -5   999    0    7    0   4  1.  1.  0.
   7    3  101   13   13    0    0   45     0     0    0    6    0   4  1.  1.  0.
```

Index file

```
row         P0          Mx          Mt        Mr          T0
  0     1.0000       .1330      -.0000       0.      1.0000
  1      .9938       .1692      -.3986       0.      1.0000
  2      .8210       .1984       .0802       0.       .9518
  3      .8112       .1858      -.4175       0.       .9518
  4      .7964       .3693       .0852       0.       .9059
```

Initial condition data.

Figure 5 — Grid, block diagram, index file, and initial condition data for a two-stage turbine with rotor tip clearances.
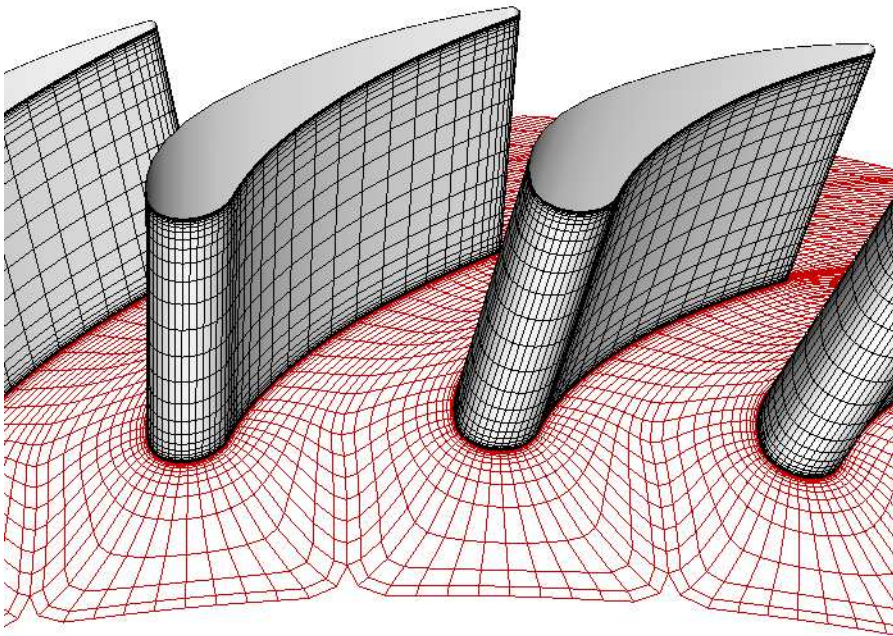
97 x 32 x 33 computational grid
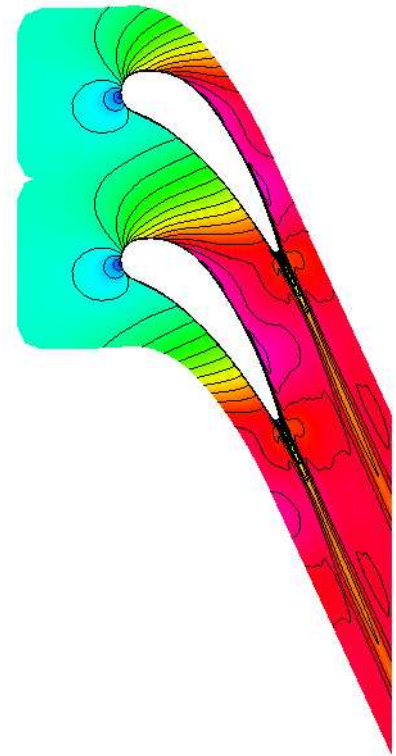
Mach number contours at midspan
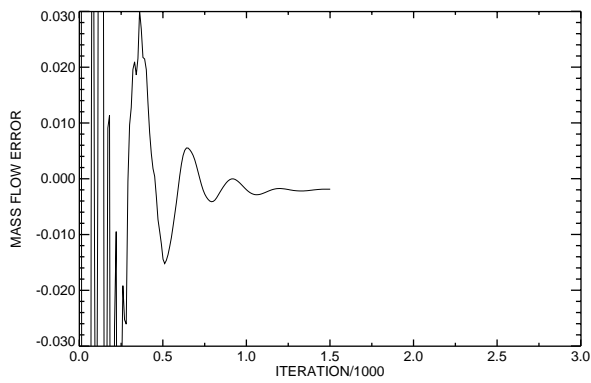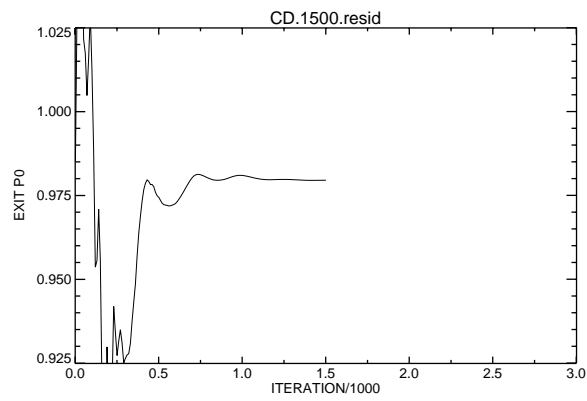
Convergence histories
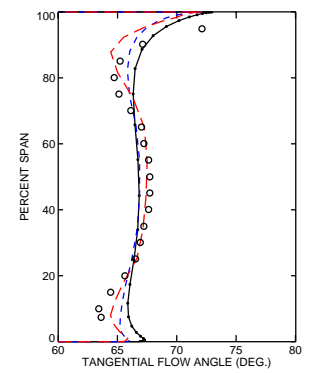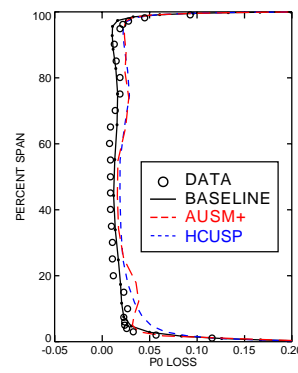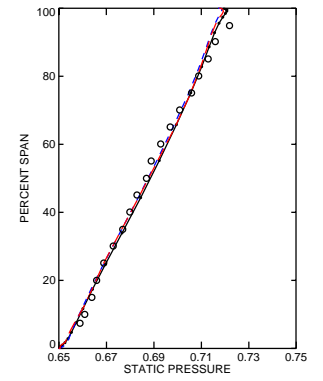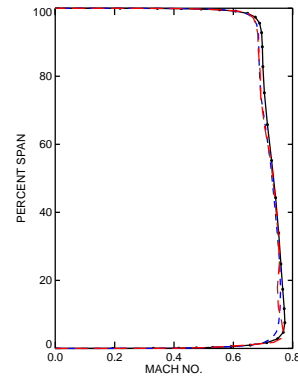
Comparison of computed and measured exit profiles

Figure 6 —   Goldman annular turbine vane test case
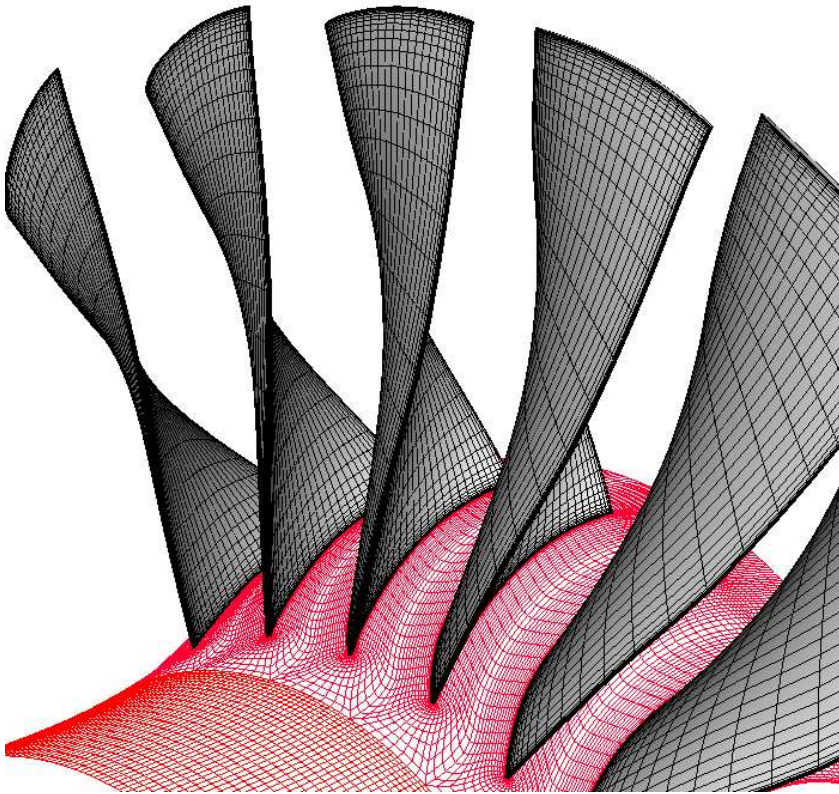
Multiblock grid
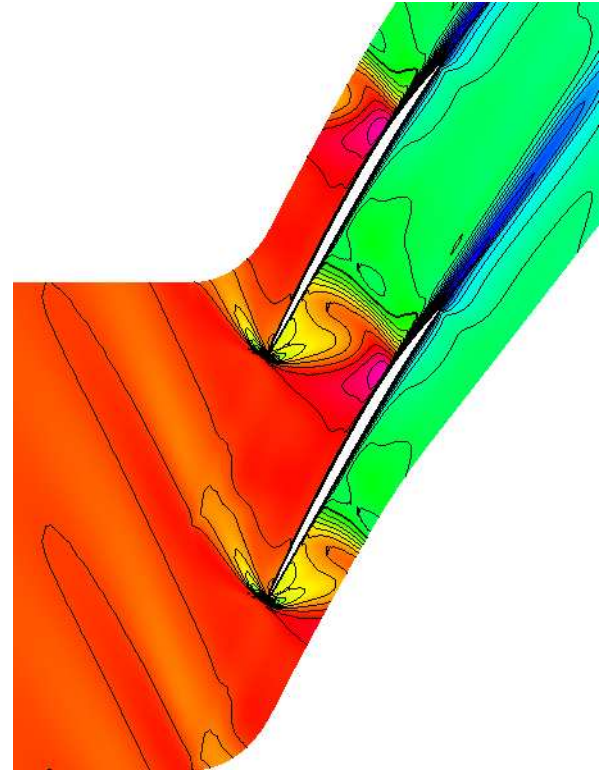
Mach number contours at 90 percent span
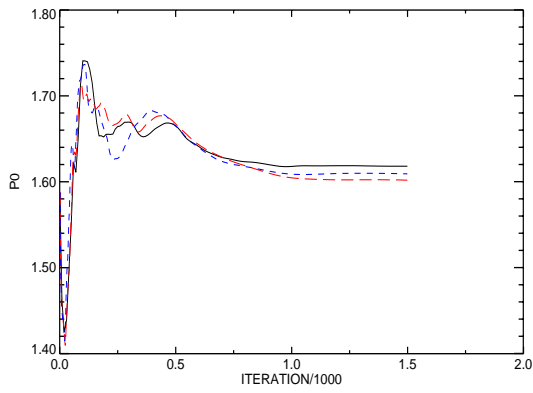
Convergence histories
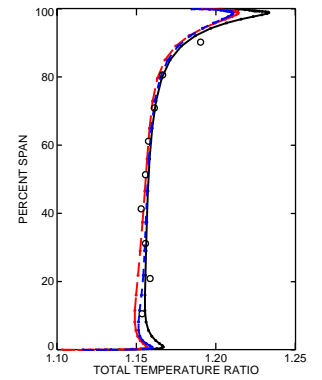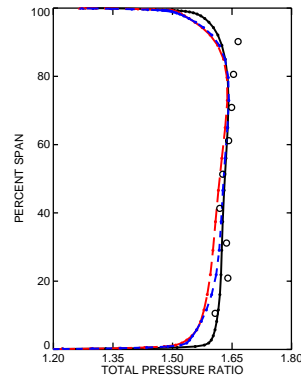
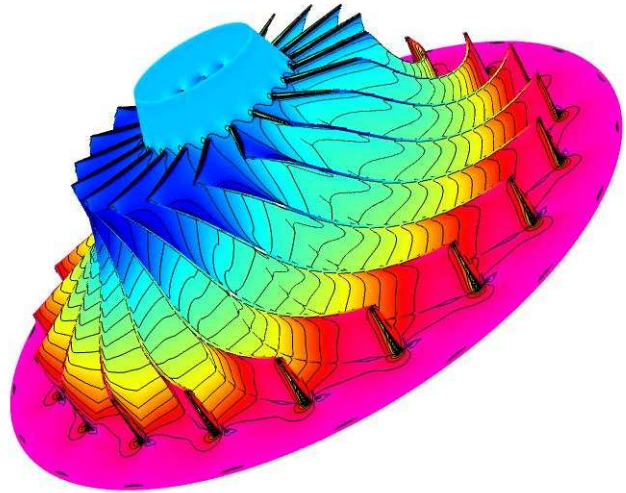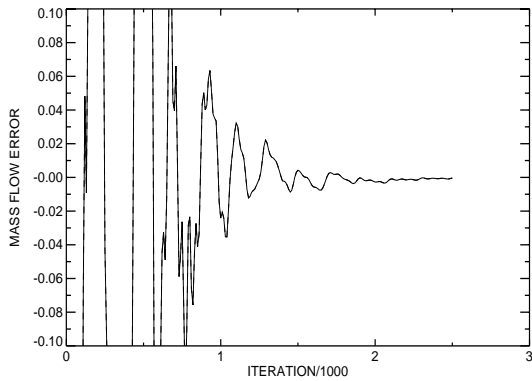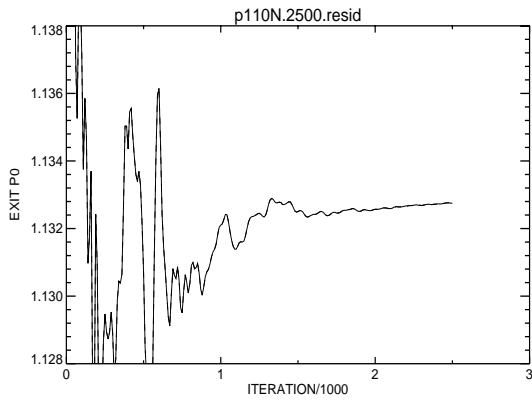Comparison of computed and measured exit profiles
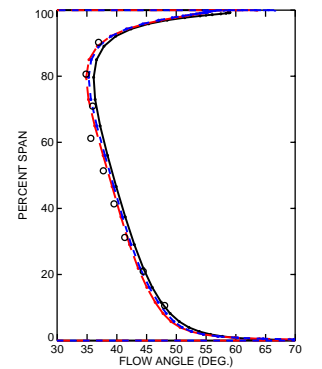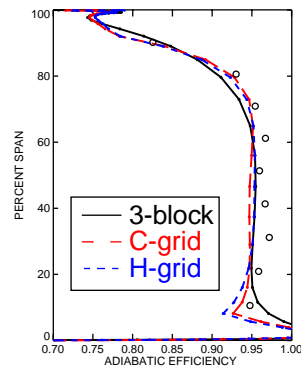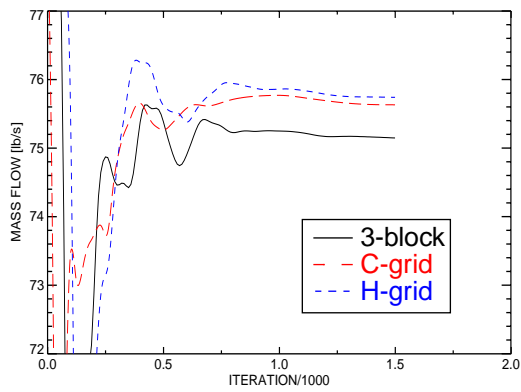
Figure 7 —   NASA rotor 67 test case

34
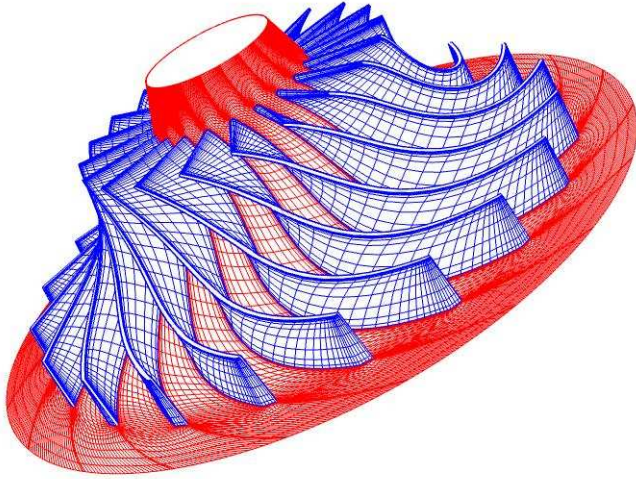
127 x 48 x 41 point H-grid
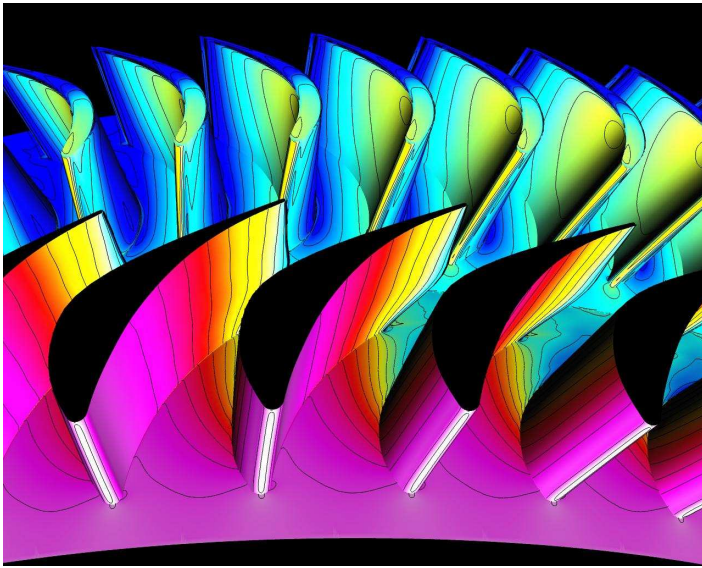

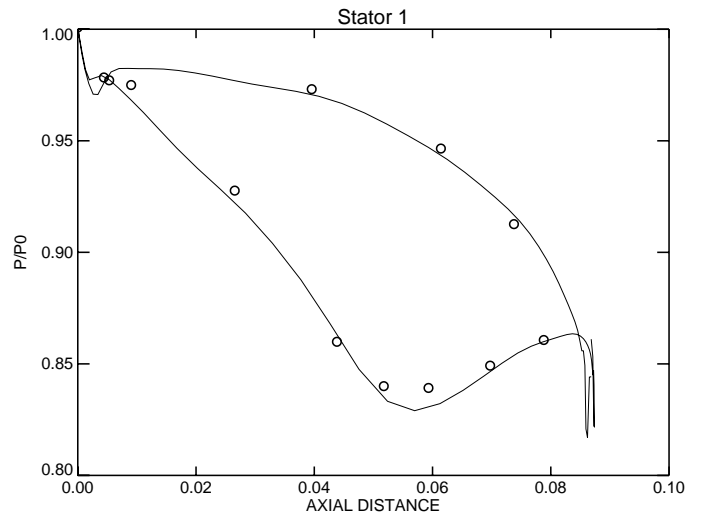Computed surface pressure contours

Convergence histories

Blade pressures at 49 pecent span
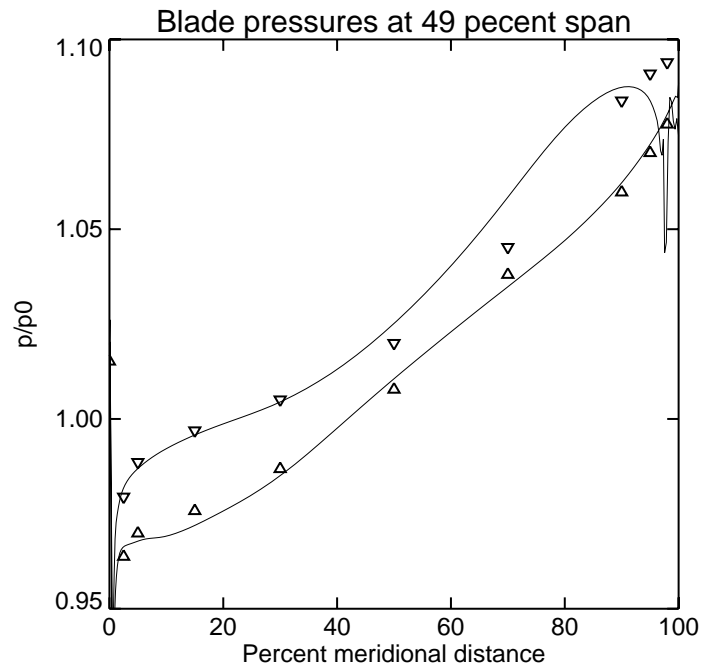
Comparison of computed and measured blade surface pressure distribution near mid span

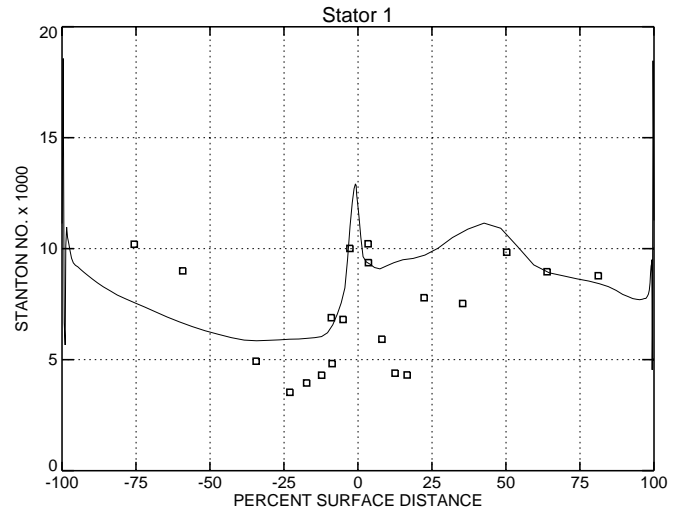Figure 8 —   NASA large low-speed centrifugal impeller test case
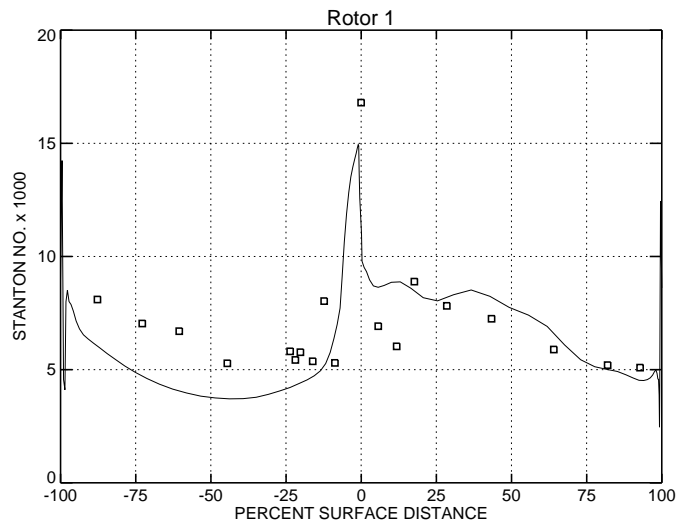
Computed surface pressure contours

Comparison of computed and measured surface pressures on stator 1 at mid span
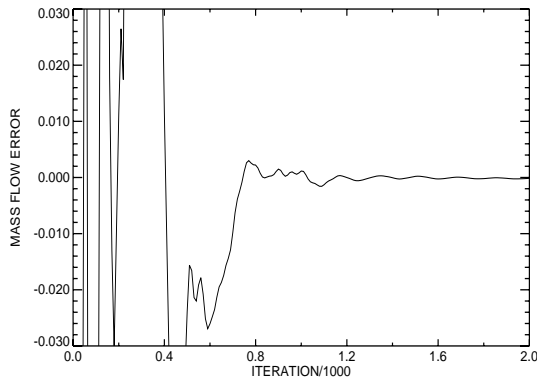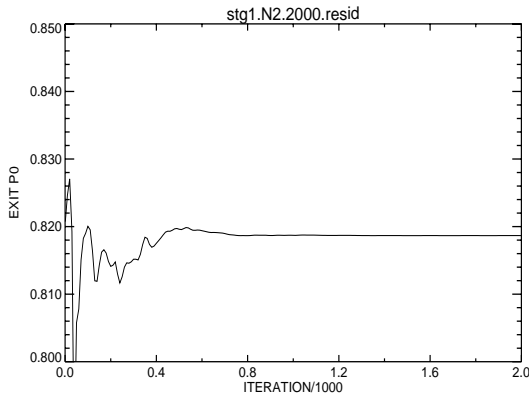
Comparison of computed and measured Stanton numbers on stator 1 at mid span

Convergence histories

Figure 9 — Single stage turbine test case

Comparison of computed and measured Stanton numbers on rotor 1 at mid span

36