

NASA/TM—2017-219418



Progress of High Efficiency Centrifugal Compressor Simulations Using TURBO

Sameer Kulkarni
Glenn Research Center, Cleveland, Ohio

Timothy A. Beach
Vantage Partners, LLC, Brook Park, Ohio

NASA STI Program . . . 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 plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Technical Report Server—Registered (NTRS Reg) and NASA Technical Report Server—Public (NTRS) thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and 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 counter-part 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 co-sponsored 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.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at <http://www.sti.nasa.gov>
- E-mail your question to help@sti.nasa.gov
- Fax your question to the NASA STI Information Desk at 757-864-6500
- Telephone the NASA STI Information Desk at 757-864-9658
- Write to:
NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

NASA/TM—2017-219418



Progress of High Efficiency Centrifugal Compressor Simulations Using TURBO

Sameer Kulkarni
Glenn Research Center, Cleveland, Ohio

Timothy A. Beach
Vantage Partners, LLC, Brook Park, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

March 2017

This report contains preliminary findings,
subject to revision as analysis proceeds.

Trade names and trademarks are used in this report for identification
only. Their usage does not constitute an official endorsement,
either expressed or implied, by the National Aeronautics and
Space Administration.

This work was sponsored by the Fundamental Aeronautics Program
at the NASA Glenn Research Center.

Level of Review: This material has been technically reviewed by technical management.

Available from

NASA STI Program
Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
703-605-6000

This report is available in electronic form at <http://www.sti.nasa.gov/> and <http://ntrs.nasa.gov/>

Progress of High Efficiency Centrifugal Compressor Simulations Using TURBO

Sameer Kulkarni
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Timothy A. Beach
Vantage Partners, LLC
Brook Park, Ohio 44142

Abstract

Three-dimensional, time-accurate, and phase-lagged computational fluid dynamics (CFD) simulations of the High Efficiency Centrifugal Compressor (HECC) stage were generated using the TURBO solver. Changes to the TURBO Parallel Version 4 source code were made in order to properly model the no-slip boundary condition along the spinning hub region for centrifugal impellers. A startup procedure was developed to generate a converged flow field in TURBO. This procedure initialized computations on a coarsened mesh generated by the Turbomachinery Gridding System (TGS) and relied on a method of systematically increasing wheel speed and backpressure. Baseline design-speed TURBO results generally overpredicted total pressure ratio, adiabatic efficiency, and the choking flow rate of the HECC stage as compared with the design-intent CFD results of Code Leo. Including diffuser fillet geometry in the TURBO computation resulted in a 0.6 percent reduction in the choking flow rate and led to a better match with design-intent CFD. Diffuser fillets reduced annulus cross-sectional area but also reduced corner separation, and thus blockage, in the diffuser passage. It was found that the TURBO computations are somewhat insensitive to inlet total pressure changing from the TURBO default inlet pressure of 14.7 psi (101.35 kPa) down to 11.0 psi (75.83 kPa), the inlet pressure of the component test. Off-design tip clearance was modeled in TURBO in two computations: one in which the blade tip geometry was trimmed by 12 mil (0.3048 mm), and another in which the hub flow path was moved to reflect a 12-mil axial shift in the impeller hub, creating a step at the hub. The one-dimensional results of these two computations indicate nonnegligible differences between the two modeling approaches.

Introduction

The High Efficiency Centrifugal Compressor (HECC) is a state-of-the-art centrifugal compressor stage designed and fabricated under a NASA Research Announcement cost-share contract in collaboration with United Technologies Research Center (UTRC). At the time of this writing, component testing of the HECC is underway at the NASA Glenn Research Center, and an in-house computational fluid dynamics (CFD) effort using the three-dimensional unsteady Reynolds-averaged Navier-Stokes (RANS) solver, TURBO Parallel Version 4, is ongoing. The motivation behind this numerical study is to demonstrate a capability to simulate the complex flows inherent to centrifugal compressors, giving insight into the details of the flow field to support the experiment. The goals of this technical memorandum are to document the methodology and the modeling choices made in this CFD effort thus far and to present the results of the initial simulations as compared with existing published results of design-intent simulations (Ref. 1). These design-intent simulations were previously generated with the commercial turbomachinery solver Code Leo (AeroDynamic Solutions).

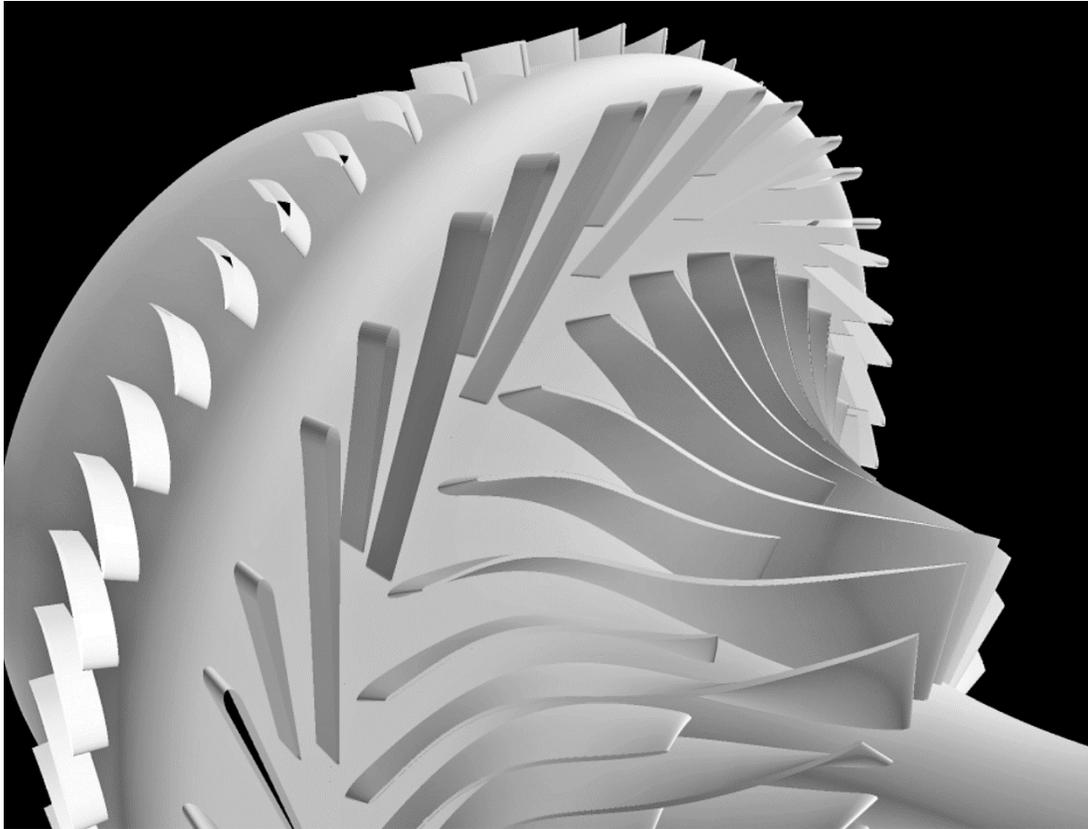


Figure 1.—Computational model of High Efficiency Centrifugal Compressor (HECC).

The HECC stage consists of an impeller, a vaned diffuser, and exit guide vanes (EGVs). The impeller features 15 pairs of main blades and splitter blades with spanwise varying backsweep, lean, and elliptical leading and trailing edges. The diffuser consists of 20 pairs of main vanes and splitter vanes. The EGV row consists of 60 cascade-style airfoils near the axial exit of the stage. A computational model of the HECC stage is depicted in Figure 1. The design point is at 21,789 rpm with an overall total pressure ratio of 5.0 and an exit corrected flow rate of 3.0 lb/s (1.36 kg/s). Further information on design requirements and design strategy is detailed in Reference 1.

The solver selected for in-house CFD simulations is TURBO, an unsteady, viscous, three-dimensional RANS solver designed for axial turbomachines. It employs the two-equation $k-\epsilon$ turbulence model with wall functions developed by the Center for Modeling of Turbulence and Transition (CMOTT) (Ref. 2). The code includes a phase-lag periodic boundary condition to model the unsteady flow from neighboring blade rows in relative motion. This allows for computation of a single blade passage rather than sector periodic or full annulus simulations, which are computationally expensive. Further information regarding the TURBO algorithm is detailed in Reference 3.

In the current effort, changes were made to the TURBO Parallel Version 4 source code to allow for proper modeling of centrifugal compressors. The details of these source code modifications are offered herein. A customized output, similar to the Plot3D format (Ref. 4), was created, and a reader for this format was written for the visualization application ParaView (Ref. 5). Various turbomachinery-related filters and scripts were also written for ParaView to aid in postprocessing. Preprocessing and meshing were done using the Turbomachinery Gridding System (TGS). Modeling choices made in TURBO are discussed herein, and a startup procedure that can serve as a template to produce a converged simulation when starting from scratch is detailed. Finally, some initial results from TURBO are presented alongside the existing design-intent simulations from Code Leo.

Computational Methodology

Changes to TURBO

The TURBO code was designed with axial turbomachines in mind, so changes to certain subroutines were required in order to simulate a centrifugal machine. In rotor passages, TURBO requires a user-specified start coordinate and end coordinate to define the region of hub rotation. In the TURBO source code, these coordinates are assumed to be axial coordinates. In the case of the HECC, the start coordinate is at a constant axial location somewhere upstream of the impeller leading edge, and this assumption is valid. However, the end coordinate is along a constant radius downstream of the impeller trailing edge, and the assumption is invalid. In light of this, a radial end coordinate was supplied in the TURBO input file in place of the axial end coordinate. A source code change was then implemented in the no-slip boundary condition subroutines of the solver. The files *bc.noslip.f*, *bc2.noslip.f*, and *bfull.f* are associated with these subroutines. Specifically, these subroutines contain checks against the variable *xend_hc*, which stores the end coordinate of the region of hub rotation. These checks must be modified so that the variable for the axial coordinate, *x*, is replaced with a newly defined variable for a radial coordinate, *r*, as shown in the following sample code from *bc.noslip.f*.

Original Source Code
<pre>if(x(i,jav,k) > xst_hc(jj,nbr).and. & x(i,jav,k) < xend_hc(jj,nbr)) then</pre>
Modified Source Code
<pre>r = sqrt(y(i,jav,k)*y(i,jav,k) + z(i,jav,k)*z(i,jav,k)) if(x(i,jav,k) > xst_hc(jj,nbr).and. & r < xend_hc(jj,nbr)) then</pre>

A separate change to the TURBO source code was made to write out a custom output file. This custom output file contains flow information similar to the Plot3D format (i.e., pressure, density, momentum, and stagnation energy) but additionally includes values of y^+ and the turbulence quantities k (turbulent kinetic energy), ϵ (dissipation), and eddy viscosity. A file reader to load this new format was written for ParaView, the postprocessing tool used in the current work. This file reader is also able to compute additional quantities, such as total pressure, total temperature, static temperature, Mach number, and relative velocity vectors. Additional details on postprocessing tools used in the current work are included in Appendix A.

Modeling Choices

The two-equation CMOTT k - ϵ turbulence model was used in the current work. This is the only turbulence model available in TURBO Parallel Version 4. The model applies wall functions when values of y^+ are greater than 10.5 and integrates to the wall for lower values of y^+ . An inlet turbulence intensity of 1 percent and an inlet turbulent eddy viscosity ratio of 100 were assumed. These are the default values in TURBO; it is recommended that these values be updated as inlet data becomes available in order to better model the component test. A grid utilizing wall functions ($y^+ > 10.5$) was generated using TGS. The grid consists of a 35-block topology with a single impeller and single diffuser passage (with splitters), and 3 EGV passages. The impeller tip clearance grid includes 18 spanwise points and is set to the design clearance of 12 mil (0.3048 mm) constant along the chord. The ratio of clearance to impeller blade exit height is 1.97. The grid has 85 points from hub to shroud, and the total number of grid points is approximately 9 million. The 35-block base topology is depicted in Figure 2. Computational efficiency was increased by balancing the total number of grid points per block, so these 35 base blocks were further subdivided into 61 total blocks with TGS before the computations were initiated. Further information on block decomposition is detailed in Appendix B.



Figure 2.—High Efficiency Centrifugal Compressor (HECC) base mesh topology.

The inlet boundary condition in TURBO is a user-prescribed spanwise profile of total pressure, total temperature, radial flow angle, and tangential flow angle. The tangential flow angle was set to zero. The near-hub and near-casing radial flow angles were taken as the approximate slopes of the hub and casing flow paths, and the spanwise profile of radial flow angle was then found by linear interpolation between these two values. The total temperature profile was set to approximately 518.4 °R (288 K) along the span. The total pressure profile was set to approximately 1 atmosphere at midspan, with approximately 10 percent boundary layer thickness at the hub and 15 percent boundary layer thickness at the casing. These total pressure and total temperature profiles are the default profiles found in the input file distributed with TURBO Parallel Version 4. It is suggested that these profiles be updated as inlet data becomes available in order to better model the component test.

The phase-lag periodic boundary condition (Ref. 6) was used to reduce the computational size of the simulation. In this mode of analysis, a single impeller and a single diffuser passage are modeled, with a sliding interface between them. A time history of the flow is stored, which supplies the boundary conditions for subsequent time steps. The diffuser main vane and splitter vane, along with three EGV passages, are modeled together in the stationary frame of reference. The solver was run in this time-accurate mode, where 3,000 iterations comprised one full wheel revolution. A row of three struts upstream of the impeller was not included in the simulation, as it was believed to have negligible aerodynamic impact upon the downstream flow field. Any leakages into or out of the flow path were thought to be negligible and were not modeled.

Startup Procedure

The TURBO simulation was initialized on a coarsened grid obtained from TGS. The coarsened grid halved the grid points of the baseline grid in each direction, resulting in a 1.1 million point grid. Working with this coarse grid allowed for faster computation and reduced wall clock run times when initializing the flow field. The TURBO input file (*input00*) was set up to initialize the flow field with uniform flow (*initialize_solution* = 1) (Ref. 7). The exit boundary condition was set to the characteristic variable supersonic outflow (*exit_bc_type* = 4). Temporal accuracy was set to first order and spatial accuracy was set to second order. The wheel speed was set to 2,000 rpm. The solver was run in this manner for the first 100 iterations.

After 100 iterations, spatial accuracy was increased to third order and temporal accuracy was increased to second order. The exit boundary condition was changed to characteristic variable subsonic outflow with radial equilibrium via user-specified pressure imposed at the casing (*exit_bc_type* = -1). This casing pressure was set to 14.8 psi (102 kPa). The simulation was run with these settings for 900 iterations. The wheel speed and backpressure were gradually increased according to the schedule in Table I until the design speed of 21,789 rpm was reached. The mass flow rate convergence history associated with these increases in wheel speed and backpressure is shown in Figure 3.

TABLE I.—TURBO INITIALIZATION PROCEDURE ON COARSE GRID

Iteration number	Wheel speed, rpm	Backpressure, Pa	Iteration number	Wheel speed, rpm	Backpressure, Pa
101 to 1,000	2,000	102,000	8,001 to 9,000	12,000	142,000
1,001 to 1,900	5,000	102,000	9,001 to 10,000	12,000	152,000
1,901 to 3,000	8,000	102,000	10,001 to 11,000	15,252	152,000
3,001 to 4,100	10,000	102,000	11,001 to 12,000	18,521	152,000
4,101 to 5,000	12,000	102,000	12,001 to 13,000	19,610	152,000
5,001 to 6,000	12,000	112,000	13,001 to 14,000	20,700	152,000
6,001 to 7,000	12,000	122,000	14,001 to 15,000	21,789	152,000
7,001 to 8,000	12,000	132,000			

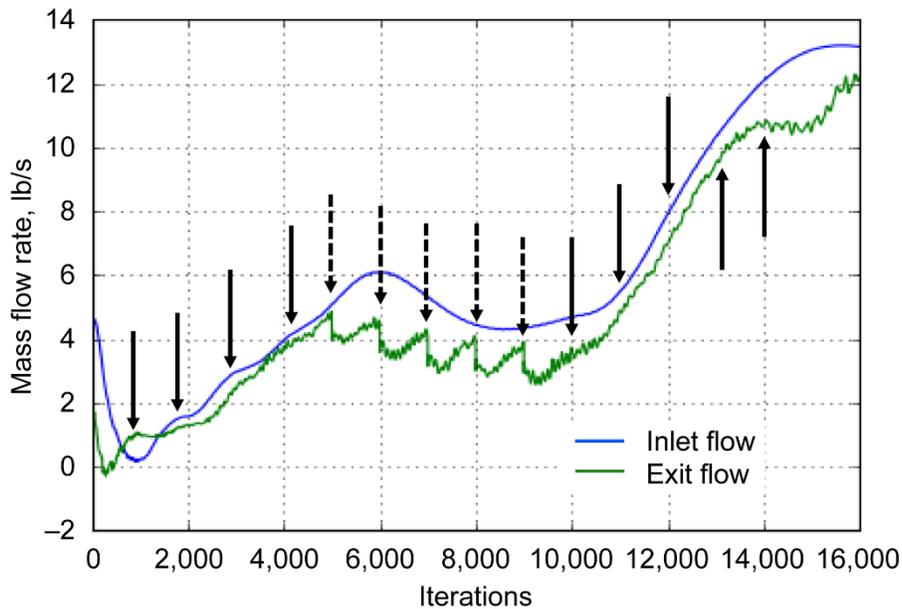


Figure 3.—Convergence history of startup on coarse mesh. Arrows with solid lines mark changes in wheel speed; arrows with dashed lines mark changes in backpressure.

After reaching design speed at a backpressure of 22.05 psi (152 kPa), the flow field of the coarse simulation was interpolated onto the baseline grid. While holding speed constant, the backpressure for this baseline simulation was incrementally increased, allowing the simulation to converge after each increase. The criteria for convergence were inlet and exit physical flow rates equal to within 0.5 percent and negligible change in overall total pressure ratio and adiabatic efficiency over one full revolution.

Computational Results

All TURBO results shown in the current work are converged, time-accurate, phase-lag simulations that have been time-averaged over either one or two revolutions. These time-averaged flow fields are a standard output from TURBO and are loaded into the ParaView visualization software for postprocessing. The TURBO output file contains time-averaged pressure, density, Cartesian momentum vectors, and stagnation energy, from which additional flow quantities, such as total pressure and total temperature, are computed in ParaView. Total pressures and total temperatures were computed at inlet and exit rating stations and then mass averaged. The inlet rating station is downstream of inlet struts (not modeled) and is 4.93 in. (12.52 cm) upstream of the impeller leading edge. The exit rating station is 0.9 in. (2.29 cm) downstream of the EGV trailing edge. Adiabatic efficiency was computed assuming a heat capacity ratio of 1.395.

Baseline Results

A speed line at 100 percent of design speed was generated on the baseline grid, which consisted of about 9 million points. This computation was not taken to numerical stall. Figure 4 shows TURBO unsteady phase-lag results compared with the steady (mixing plane) and unsteady (sector periodic) design-intent Code Leo results from Reference 1.

The baseline TURBO computation overpredicts the choking flow rate by 0.75 percent as compared with the Code Leo unsteady computation. In general, the TURBO computation overpredicts pressure ratio compared with the Code Leo results.

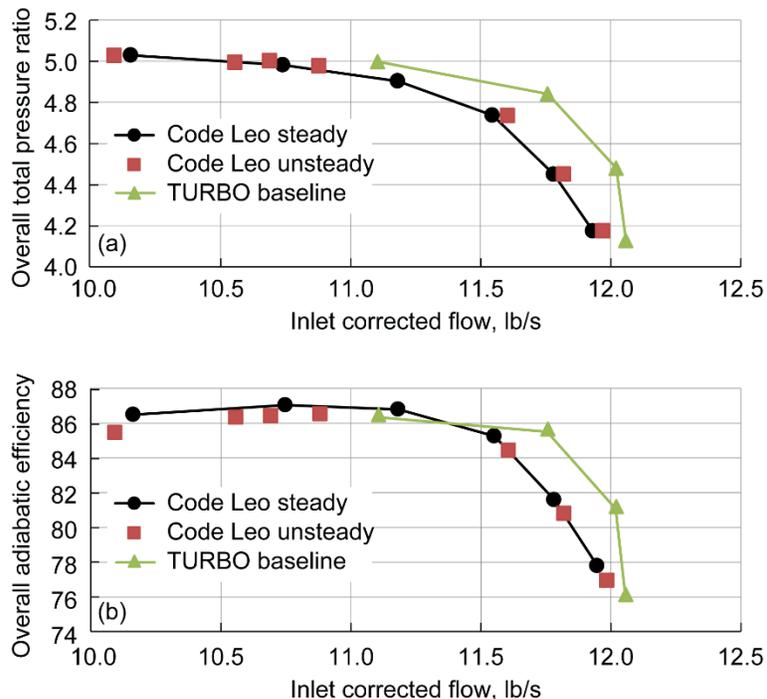


Figure 4.—Computed design speed performance of High Efficiency Centrifugal Compressor (HECC). (a) Pressure ratio. (b) Efficiency.

Effect of Diffuser Vane Fillets

The baseline grid did not include fillets in the geometry definition. Because this compressor chokes in the diffuser and the diffuser fillet area was estimated to be 1.0 to 1.5 percent of the annulus area, a new TURBO grid that included the diffuser vane fillets was generated. It was thought that the differing area might be large enough to impact the choking flow rate. The resulting speed line of the TURBO computation with diffuser vane fillets is shown in Figure 5. This computation was run to near the numerical stall boundary.

The inclusion of diffuser fillets led to a 0.6 percent reduction in the choking flow rate of the compressor. The TURBO computation with fillets was within 0.15 percent of the choking flow rate of the design-intent Code Leo computation. In general, the inclusion of fillets brought the shape of the TURBO speed line closer to the Code Leo speed line. However, TURBO generally overpredicted performance as compared with Code Leo. It was hypothesized that the inclusion of fillets would have a larger impact on the choking flow rate. The relatively small reduction in the choking flow rate may be explained by examining a cross-flow plane near the diffuser throat at a choked operating point, as shown in Figure 6.

The baseline case without fillets (Fig. 6(a)) shows regions of negative radial velocity (i.e., separated flow) in the corner of the pressure side of the main diffuser vane and the hub, as well as in the corner of the suction side of the main vane and the casing. These regions of aerodynamic blockage reduce the effective flow area. After regriding to include fillets (Fig. 6(b)), the pressure-side separation is reduced and the suction-side separation is gone. The inclusion of diffuser vane splitters reduced annulus area but also reduced the aerodynamic blockage caused by these regions of separation. In the baseline case, this separation may have been caused by the sharp corners. These sharp corners were rounded out with the inclusion of fillets, eliminating the flow separation. The computation with fillets more closely matched the design-intent CFD, so the grid with diffuser fillets was used for subsequent computations.

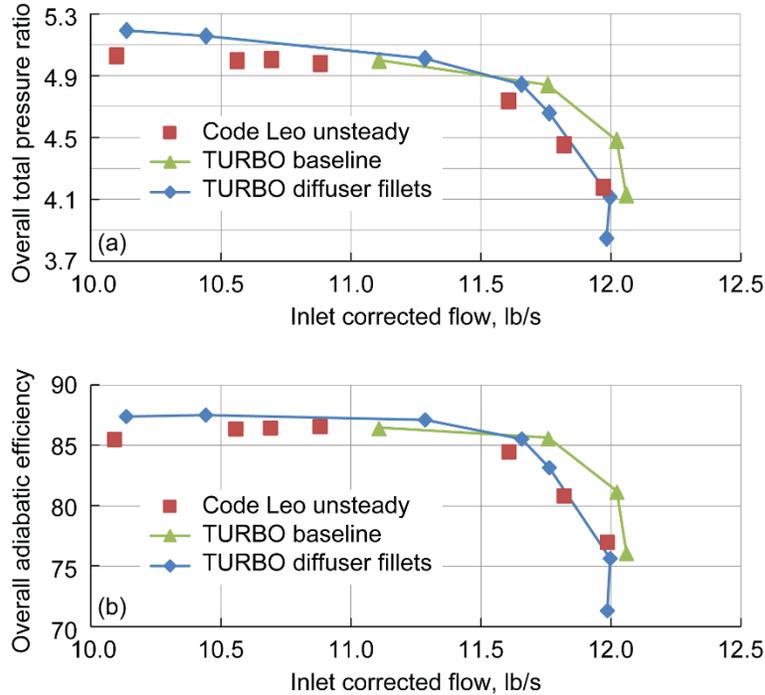


Figure 5.—Computed design speed performance of High Efficiency Centrifugal Compressor (HECC) with and without diffuser fillets. (a) Pressure ratio. (b) Efficiency.

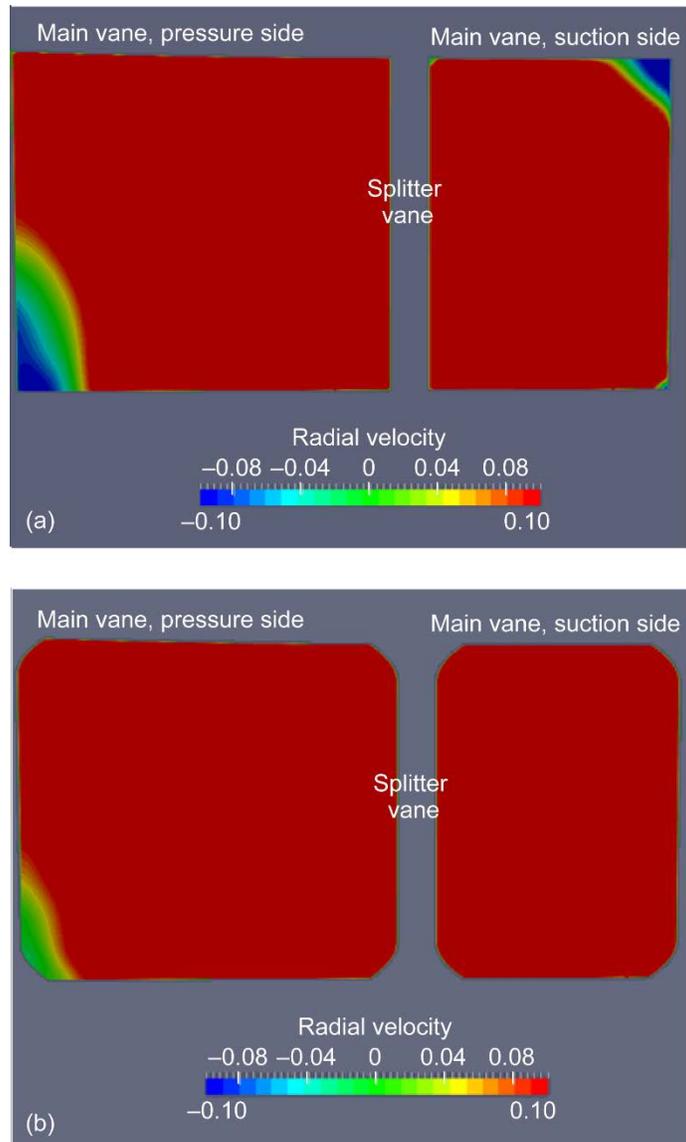


Figure 6.—Cross-flow plane of radial velocity near diffuser throat at choked operating point. (a) Baseline case without fillets. (b) After regriding to include fillets.

Suppressed Inlet Total Pressure

The component testing of HECC is done with inlet total pressure suppressed below its ambient value; during typical operation, inlet total pressure is 10.5 to 11.0 psi (72.4 to 75.8 kPa). In an effort to better model the component test, a TURBO computation at design speed was generated with inlet total pressure suppressed to 11.0 psi, down from the previous value of 14.7 psi (101.35 kPa). The resulting speed line, which was not run to the stall point, is shown in Figure 7.

A comparison of these speed lines indicates that the reduction in inlet total pressure had a negligible impact on both the choking flow rate (less than 0.1 percent change) and the overall performance in the TURBO simulations. This indicates that the modeling choices made in the current effort yield simulations that are relatively insensitive to this magnitude of change in inlet Reynolds number.

Modeling Tip Clearance

A goal of the component test is to collect tip clearance sensitivity data. To support this goal, it is necessary to properly model varying tip clearance in the CFD simulations. The impeller was designed for 12-mil (0.3048-mm) constant clearance along the chord. In the test rig, tip clearance is controlled by shifting the impeller hub in the axial direction. Inherently, this will create a step in the flow path between the impeller hub and the diffuser hub, downstream of the impeller trailing edge. For example, increasing the trailing edge tip clearance from 12 mil (0.3048 mm) to 24 mil (0.6096 mm) will create a 12-mil step in the vaneless space between impeller trailing edge and diffuser main vane leading edge. A computational effort to model this off-design tip clearance was undertaken. In one computation, the trailing edge tip clearance was increased to 24 mil by shifting the impeller hub, as in the test cell. This formed a 12-mil step in the flow path, as shown in Figure 8.

In a second computation, the tip clearance of the entire blade was increased to 24 mil by holding the hub and shroud flow paths constant but trimming the impeller blade tips. Figure 9 compares the results of these two computations with the design-intent tip clearance computation.

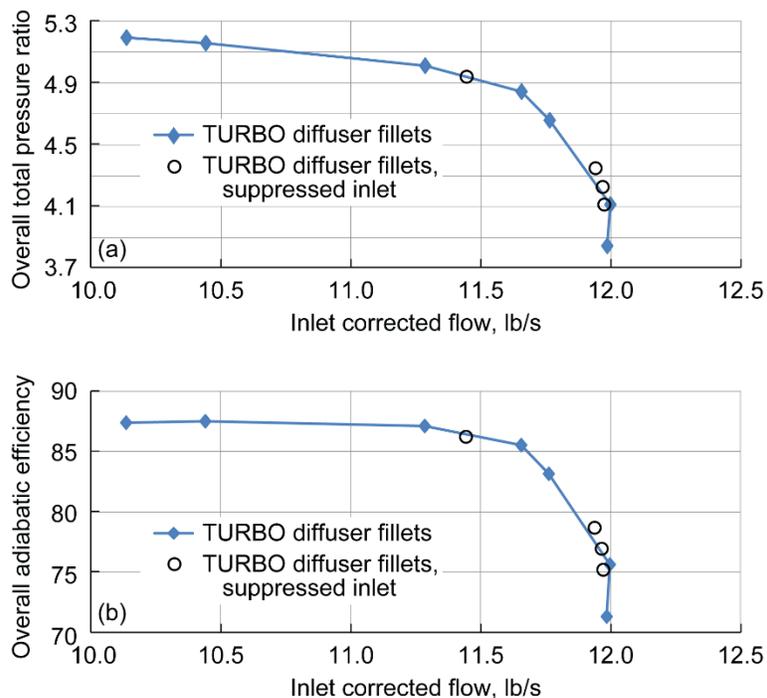


Figure 7.—Computed design speed performance of High Efficiency Centrifugal Compressor (HECC) with suppressed inlet total pressure. (a) Pressure ratio. (b) Efficiency.

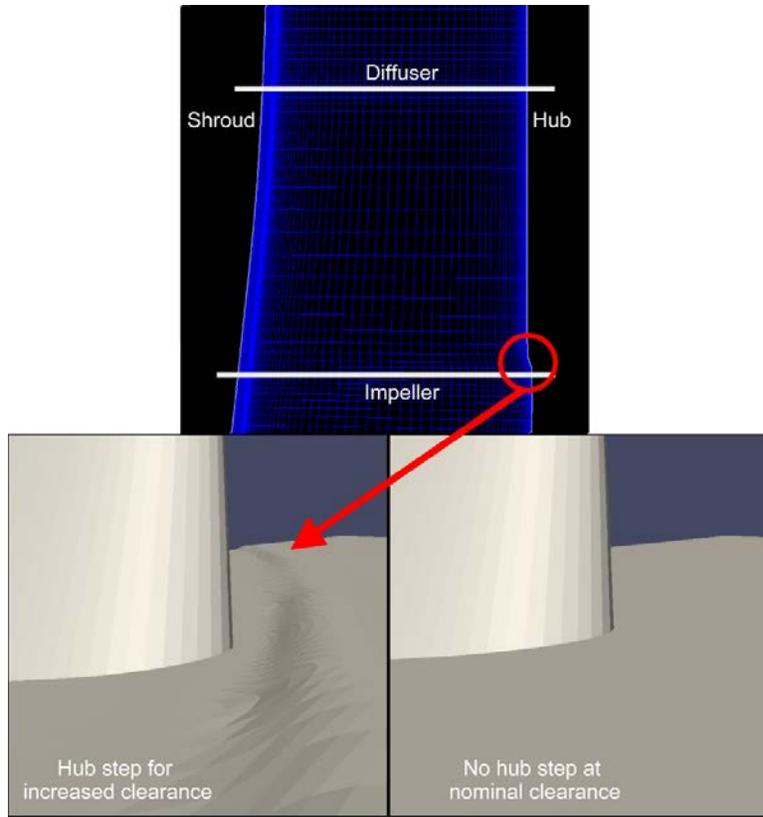


Figure 8.—Step at hub formed by increasing tip clearance at impeller trailing edge.

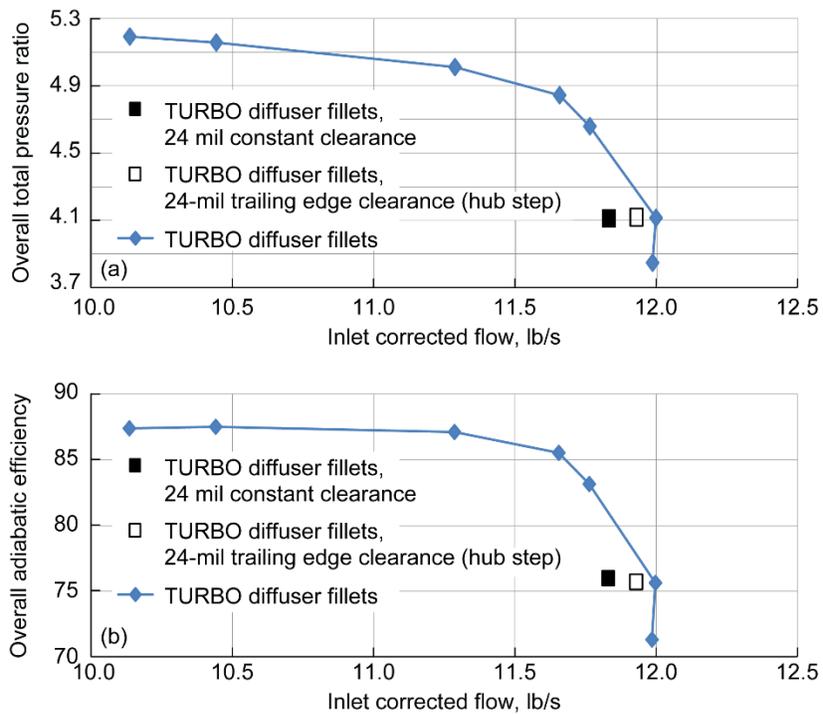


Figure 9.—High Efficiency Centrifugal Compressor (HECC) design-intent tip clearance (12 mil) result compared with 24-mil tip clearance cases. (a) Pressure ratio. (b) Efficiency.

Shifting the impeller hub in order to double the impeller trailing edge clearance resulted in a 0.5 percent reduction in the choking flow rate in the case where the hub was shifted, and a 1.3 percent reduction in the choking flow rate in the case where the blade tip was trimmed. These figures indicate that there are nonnegligible differences in the two ways of modeling the increased tip clearance gap. The complication of regriding the geometry to include the hub step may be necessary to properly model the component rig when simulating off-design tip clearances.

Conclusions and Recommendations

Preliminary design speed computational fluid dynamics (CFD) computations of the High Efficiency Centrifugal Compressor (HECC) stage were generated using the TURBO solver. With the modeling choices made in the present work, the results generally overpredicted the choking flow rate, total pressure ratio, and adiabatic efficiency, as compared with the design-intent CFD results from Code Leo. Inclusion of the diffuser fillet geometry resulted in a reduction of 0.6 percent in the choking flow rate and a better match to design-intent CFD. The current TURBO result was computed with inlet total pressure at the default level of 14.7 psi (101.35 kPa) as well as at 11 psi (75.8 kPa), the level that more closely matched the component test. One-dimensional performance metrics indicated that these TURBO results were insensitive to this difference in inlet total pressure. An off-design tip clearance study was performed with two different computations: one in which the blade tip geometry was trimmed 12 mil (0.3048 mm), and another in which the impeller hub was shifted axially 12 mil (as is done in the component test rig). These two computations differed in choking flow rate by 0.8 percent, indicating that these modeling approaches produce dissimilar flow fields.

For future iterations of these TURBO computations, the authors recommend that inlet boundary conditions be modified as needed to better match the conditions seen in the component testing of the HECC. These inlet boundary conditions include turbulence intensity and spanwise profiles of total pressure, total temperature, and radial and tangential flow angle.

Appendix A.—Interpolating, Integrating, and Averaging TURBO Results Using ParaView

ParaView (Ref. 5) is an open source program for scientific visualization that has been implemented as an important tool for postprocessing TURBO results. ParaView has been shown to be useful for studying high-resolution three-dimensional data. A custom reader has been implemented, allowing easy access to turbomachinery quantities. From the user interface, it is very easy to generate typical graphics such as contours, stream lines, and vector plots. At a slightly deeper level, ParaView offers tools for data interpolation, integration, and averaging. Given herein are examples of using python scripts to generate one- and two-dimensional data from the three-dimensional TURBO flow field.

Several excellent tutorials for interactive sessions are included with the ParaView distribution. Documentation for python scripting is less readily available, but there is a wide user base and much information on the Internet. It is also possible to create a python script using ParaView's trace function, which generates a python script that duplicates the interactive session. This is often the starting point for building more powerful scripts that can be executed as batch scripts using the pvpypython interpreter that binds ParaView functionality into the standard python converter. The example presented here is a script for producing circumferentially mass-averaged profiles of total pressure, static pressure, total temperature, and flow angle at the centrifugal compressor's exit rating plane, which is defined by a constant radius. The result of this script is shown in Figure 10. The averaging computations were performed using functionality in ParaView, and the line plots were generated using Matplotlib (Ref. 8), a standard plotting library for python.

The complete script for producing this plot follows here; comments are included for clarity. The techniques used in this script can be adapted to provide a wide variety of postprocessing tools for studying high-resolution three-dimensional data. The goal is to maintain a library of useful procedures for studying turbomachinery-related datasets.

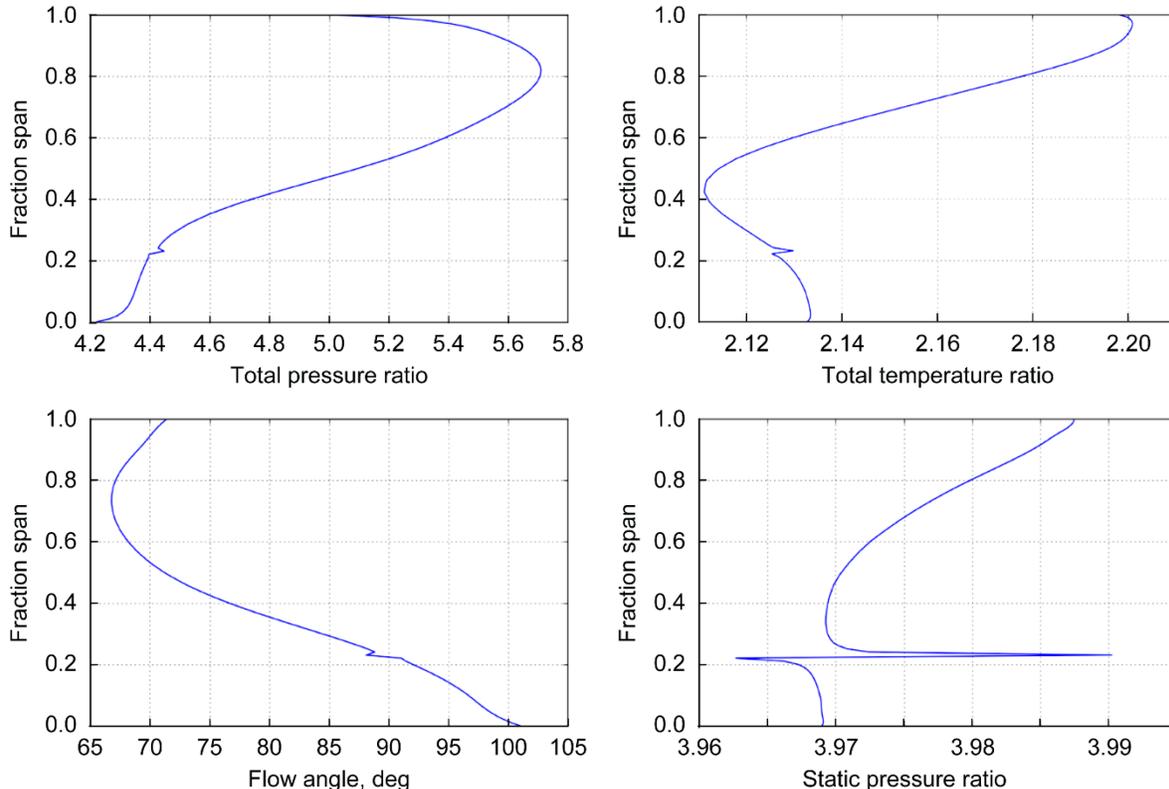


Figure 10.—Mass-averaged profiles produced by pvpypython script for a centrifugal compressor. Profiles at radius ratio 0.31541 averaged along 100 spanwise locations.

Python Script to Produce Circumferentially Mass-Averaged Spanwise Profiles at Centrifugal Compressor Exit

Python Code

```
#
# input definitions for this run
#
radius = 0.31541
gfile = '/media/Data1/RUN/CC3/V9.VANELESS/FINE/cc3_n_12.trb'
qfile = '/media/Data1/RUN/CC3/V9.VANELESS/FINE/cc3_n_128030.q'
steps = 100 # number of integration steps

# import math and pyplot libraries for line plotting
import math
import pylab as pl
from matplotlib.ticker import ScalarFormatter
# import paraview library
try: paraview.simple
except: from paraview.simple import *
paraview.simple._DisableFirstRenderCameraReset()

# read TURBO Grid and Q File
cc3_trb = TURBOReader( FileName=gfile)
# Select Values to load into memory as defined in ParaView
# to be mostly equivalent to plot3d function numbers
# 154 – Mach Number
# 60 – Radius
# 112 – Total Pressure
# 119 – Relative Total Pressure
# 121 – Total Temperature
# 156 – Radial Velocity
# 157 – Meridional Velocity
# 200 – Velocity Vectors
# 280 – Relative Velocity Vectors
cc3_trb.Functions = [154, 60, 112, 119, 121, 156, 157, 200, 280]
cc3_trb.QFileName = [qfile]

# Contour filter at the user-specified constant-radius location
radiusContour = Contour( PointMergeMethod="Uniform Binning" )
radiusContour.ContourBy = ['POINTS', 'Radius']
radiusContour.Isosurfaces = [radius]

# Merge blocks filter necessary since the constant
# radius cut likely cuts through multiple blocks
SetActiveSource(radiusContour)
mergeBlocks = MergeBlocks()

# Calculator filter to get access to X-coordinate information
xCalc = Calculator()
xCalc.AttributeMode = 'point_data'
xCalc.Function = 'coordsX'
xCalc.ResultArrayName = 'x'

# Linear discretization of x-coordinate (which is the spanwise
# direction on a radial cut)
xmin = xCalc.PointData["x"].GetRange()[0]*1.001
xmax = xCalc.PointData["x"].GetRange()[1]*.999
```

```

xlist = []
xstep = (xmax - xmin)/steps
for step in range(steps+1):
    xlist.append(xmin + step*xstep)

# Mesh quality filter calculates cell areas (named "Quality" by default)
meshQuality = MeshQuality()
meshQuality.TriangleQualityMeasure = 'Area'
meshQuality.QuadQualityMeasure = 'Area'

# Cell data to point data filter to convert cell area array
# to point data array
CellDatatoPointData1 = CellDatatoPointData()

# Calculator filter to calculate local mass flow rate at
# each point on the cutting plane
weightCalc = Calculator()
weightCalc.AttributeMode = 'point_data'

# Various weighted average techniques
# Mass Average (normal momentum)
weightCalc.Function = '2*Quality*(Normals.Momentum)'
# Density Average
#weightCalc.Function = '2*Quality*Density'
# Area Average
#weightCalc.Function = '2*Quality'
weightCalc.ResultArrayName = 'weight'

# Loop through every point on the constant-radius cut to
# sum total surface area and total mass flow rate normal to the surface
numPts = servermanager.Fetch(weightCalc).GetNumberOfPoints()
print numPts;
weight = servermanager.Fetch(weightCalc).GetPointData().GetArray("weight")
quality = servermanager.Fetch(weightCalc).GetPointData().GetArray("Quality")

weightTotal = 0
areaTotal = 0

for i in range(0,numPts):
    weightTotal += weight.GetValue(i)
    areaTotal += quality.GetValue(i)*2
#output
print 'Total Area = ' + str(areaTotal)
print 'Total Weight Function = ' + str(weightTotal)

# Loop through discretized x-values (spanwise direction), generating
# a constant-x line along the constant-radius surface. Weighted radial and
# tangential velocities, total pressure, and total temperature are calculated
# and summed at each circumferential point # along the line, then divided
# by the total mass flow along the line. This gives spanwise profiles
# of averaged radial and tangential velocities, P0 and T0.
radVel = []
tanVel = []
P0 = []
T0 = []
Ps = []

```

```

for xpos in xlist:
    SetActiveSource(weightCalc)
    xContour = Contour( PointMergeMethod="Uniform Binning" )
    xContour.PointMergeMethod = "Uniform Binning"
    xContour.ContourBy = ['POINTS', 'x']
    xContour.Isosurfaces = [xpos]
    xLine = servermanager.Fetch(xContour)
    numPtsLine = xLine.GetNumberOfPoints()
    weightLine = xLine.GetPointData().GetArray("weight")
    radVelLine = xLine.GetPointData().GetArray("RadialVelocity")
    tanVelLine = xLine.GetPointData().GetArray("TangentialVelocity")
    P0Line = xLine.GetPointData().GetArray("TotalPressure")
    T0Line = xLine.GetPointData().GetArray("TotalTemperature")
    PsLine = xLine.GetPointData().GetArray("Pressure")
    weightLineTotal = 0
    weightedRadVel = 0
    weightedTanVel = 0
    weightedP0 = 0
    weightedT0 = 0
    weightedPs = 0
    for i in range(0,numPtsLine):
        weightLineTotal += weightLine.GetValue(i)
        weightedRadVel += weightLine.GetValue(i)*radVelLine.GetValue(i)
        weightedTanVel += weightLine.GetValue(i)*tanVelLine.GetValue(i)
        weightedP0 += weightLine.GetValue(i)*P0Line.GetValue(i)
        weightedT0 += weightLine.GetValue(i)*T0Line.GetValue(i)
        weightedPs += weightLine.GetValue(i)*PsLine.GetValue(i)
    radVel.append(weightedRadVel / weightLineTotal)
    tanVel.append(weightedTanVel / weightLineTotal)
    P0.append(weightedP0 / weightLineTotal)
    T0.append(weightedT0 / weightLineTotal)
    Ps.append(weightedPs / weightLineTotal)
    Delete(xContour)

# Calculate span and flow angle
# (here, zero flow angle is in the radial direction)
span = [1-(x-xmin)/(xmax-xmin) for x in xlist]
alpha = [math.atan2(tan,rad)*180/math.pi for rad,tan in zip(radVel,tanVel)]
print radVel
print tanVel
print alpha
print span
print P0
print T0
print Ps

# Delete filters created in this script
Delete(weightCalc)
Delete(CellDatatoPointData1)
Delete(meshQuality)
Delete(xCalc)
Delete(mergeBlocks)
Delete(radiusContour)

# Plot circumferentially averaged profiles
fig = pl.figure()
pl.suptitle("Profiles at radius " + str(radius) +
           "\n averaged along " + str(steps) + " spanwise locations")

```

```
ax1 = fig.add_subplot(2,2,1)
ax1.plot(P0,span)
pl.xlabel("Total Pressure Ratio")
pl.ylabel("Fraction Span")
pl.grid(True)
ax2 = fig.add_subplot(2,2,2, sharey=ax1)
ax2.plot(T0,span)
pl.xlabel("Total Temperature Ratio")
pl.ylabel("Fraction Span")
pl.grid(True)
ax3 = fig.add_subplot(2,2,3, sharey=ax1)
ax3.plot(alpha,span)
pl.xlabel("Flow Angle (deg)")
pl.ylabel("Fraction Span")
pl.grid(True)
ax4 = fig.add_subplot(2,2,4, sharey=ax1)
ax4.plot(Ps,span)
pl.xlabel("Pressure Ratio")
pl.ylabel("Fraction Span")
pl.grid(True)
pl.gca().xaxis.set_major_formatter(ScalarFormatter(useOffset=False))
pl.show()
```


Appendix B.—Modifying TURBO Block Decomposition Using Turbomachinery Gridding System (TGS)

Over the course of this project, a procedure has been developed for altering the block decomposition for TURBO turbomachinery analysis. This can be useful in finding the optimal block size for maximizing computing efficiency. It can also be helpful in postprocessing by reducing the block count. TURBO execution depends on the following file groups:

GUn0—Grid file for each block of the decomposition ($n = 1 \rightarrow n_{\max}$)
QUn0—Flow file for each block of the decomposition
bc.in—File containing physical boundary conditions as applied to each block
dmap.in—File containing the block connections for each of n blocks
turbo.in—File containing information about how blocks are collected into blade rows

The use of this procedure requires that the geometry be meshed and collected into the minimum number of blocks to be considered. For example, Figure 11 shows a mesh for the CC3 impeller (Ref. 9) that includes 11 blocks, including O-mesh blocks around the blades and the tip gap meshes. This is the minimum number blocks for this case when following general practices for TURBO block decomposition. Most of these blocks are too large for actual computation, so they must be subdivided. This process requires that the *bc.in*, *dmap.in*, and *turbo.in* files be created for these basic blocks as if they would be used with TURBO. For this case, *bc.in* contains 37 boundary patches and *dmap.in* contains 27 boundary patches. A TGS script for decomposition will include the following steps:

- (1) Read grid and flow files.
- (2) Read boundary condition files.
- (3) Subdivide each block as desired.
- (4) Write a new set of files for the new decomposition.

An example script with documentation is included at the end of Appendix B. In general, the size of each block is compared with the maximum desired size and divided accordingly. For use in TURBO, each new block will have a minimum of four cells in each direction and will include an even number of cells in each direction. (This is the default, although these rules can be altered.) Running the sample script results yields a TURBO grid of 181 blocks with a maximum block size of 43,000 points, as shown in Figure 12. The file *bc.in* is updated to contain 599 patches, and the *dmap.in* file now contains 377 patches. The *turbo.in* file is not applicable in the case because it is an isolated blade row. A new file called *extract.in* is also created in the process and can be used to reassemble the solution files into the original 11 blocks from the 181-block computation. This is done with the TGS command:

Python Code

```
tgs.merge(TurboData="PATH")  
# PATH is the directory path of the 181-block analysis
```

This case could then be restarted with a different block count. Note that this will not maintain time accuracy or phase-lag continuity. Some boundary values and time history information is not kept. These TURBO block decomposition procedures in TGS can be used to subdivide TURBO meshes to the desired number of blocks.

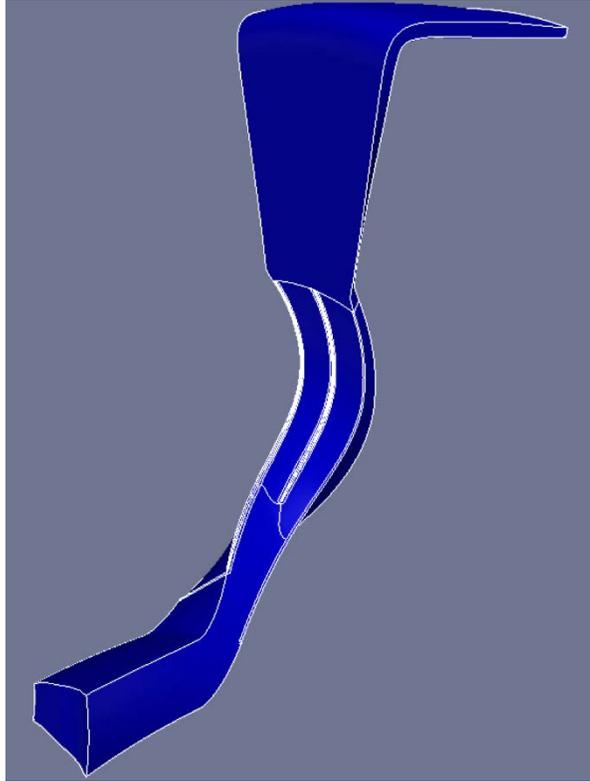


Figure 11.—CC3 impeller with minimum 11 blocks.

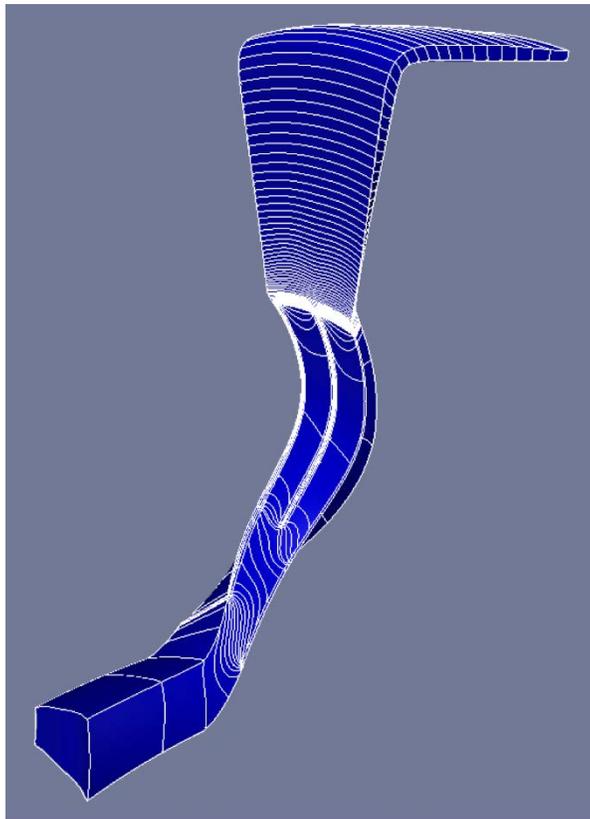


Figure 12.—CC3 impeller with 181 blocks.

TGS Python Script for TURBO Block Decomposition

Python Code

```
#!/usr/bin/python

# set size limit for decomposition
size_limit=43000

# import tgs
import tgs

# initialize tgs and get block count in baseline directory
tgs.init(File="decomp.inp")
nb = tgs.count_gu_files(Path="../BASE")

# define lists and read grid and flow files
gn=[]
qn=[]
name=[]
qname=[]

for n in range(nb):
    f = "../BASE/GU"+str(n+1)+"0"
    qf = "../BASE/QU"+str(n+1)+"0"
    name="GRID"+str(n)
    qname="Q"+str(n)
    gn.append(name)
    qn.append(qname)
    tgs.read(Part="grid",Name=name,File=f,Type="turbo_gu",Form="unf")
    tgs.read(Part="data",Grid=name,Name=qname,File=qf, Type="turbo_qu",Form="unf")

# read bc and dmap files. turbo.in not required for single blade row
tgs.read(Part="bc",File="../BASE/bc.in",Type="turbo_bc", Grid="GRID",Name="GU")
tgs.read(Part="bc",File="../BASE/dmap.in",Type="turbo_dmap", Grid="GRID",Name="GU")

# set blade row and passage number of existing blocks
for n in range(nb):
    tgs.boundaries(Grids=gn[n],BoundaryFunction="set_blade_row", BoundaryData="1")
    tgs.boundaries(Grids=gn[n],BoundaryFunction="set_blade_passage", BoundaryData="1")

# loop over block and check size
rotorgrids=""
rotordata=""
for n in range(nb):
    g = "GRID"+str(n)
    q = "Q"+str(n)
    # get grid size
    nx = tgs.get_grid_size_x_by_name(g)
    ny = tgs.get_grid_size_y_by_name(g)
    nz = tgs.get_grid_size_z_by_name(g)
    size=nx*ny*nz
    print "grid size @ ",n," = ",size

# compare block size with limit
if size > size_limit:
    # get number of splits required to achieve grid size
    ns = int(1 + float(size)/float(size_limit))
    # call turbo grid splitter to split in l direction
```

```

tgs.utility(Grid=g,Data=q,Function="turbosplit", N1=str(ns),N2=str(1))
# add grid children to list of grids (add -> to grid name)
rotorgrids=rotorgrids+g+"->"
rotordata=rotordata+q+"->"
tgs.delete(Grid=g)
tgs.delete(Data=q)
else:
# add grid to list if already small enough
rotorgrids=rotorgrids+g
rotordata=rotordata+q
if n < nb-1:
rotorgrids=rotorgrids+", "
rotordata=rotordata+", "

print "rotor grids = ",rotordata

# build a group for export
tgs.group(Name="ROTOR",Data=rotordata)

# write TURBO files in the current directory
tgs.write(Type="turbo_gu_old",Form="unf",Name="TTT",Groups="ROTOR")

tgs.free()

```

References

1. Lurie, E.A., et al.: Design of a High Efficiency Compact Centrifugal Compressor for Rotorcraft Applications. Presented at the American Helicopter Society 67th Annual Forum, Virginia Beach, VA, 2011.
2. Zhu, J.; and Shih, T.H.: CMOTT Turbulence Module for NPARC. NASA CR-204143, 1997. <http://ntrs.nasa.gov>
3. Chen, J.P.; and Briley, W.R.: A Parallel Flow Solver for Unsteady Multiple Blade Row Turbomachinery Simulations. ASME Paper No. 2001-GT-0348, 2001.
4. Plot3d File Format for Grid and Solution Files. NPARC Alliance CFD Verification and Validation Web Site, July 17, 2008. <https://www.grc.nasa.gov/WWW/wind/valid/plot3d.html>. Accessed Feb. 10, 2017.
5. Ahrens, James; Geveci, Berk; and Law, Charles: ParaView: An End-User Tool for Large Data Visualization. Visualization Handbook, Elsevier, New York, NY, 2005.
6. Chen, Jen Ping; and Barter, Jack: Comparison of Time-Accurate Calculations for the Unsteady Interaction in Turbomachinery Stages. AIAA-98-3292, 1998.
7. MSU TURBO. Online Documentation, Mississippi State University, MS, 2002.
8. Hunter, J.D.: Matplotlib: A 2D Graphics Environment. Comput. Sci. Eng., vol. 9-3, 2007, pp. 90-95.
9. McCain, T.F.; and Holbrook, G.J.: Coordinates for a High Performance 4:1 Pressure Ratio Compressor. NASA CR-204134, 1997. <http://ntrs.nasa.gov>

