Development of Quiet-Flow Supersonic Wind Tunnels for Laminar-Turbulent Transition Research: Final Report for NASA Langley Grant NAG-1-1133

Steven P. Schneider
Assistant Professor of Aerodynamics
School of Aeronautical and Astronautical Engineering
Purdue University
West Lafayette, IN 47907-1282

Abstract

This grant supported research into quiet-flow supersonic wind-tunnels, between May 1990 and December 1994. Quiet-flow nozzles operate with laminar nozzle-wall boundary layers, in order to provide low-disturbance flow for studies of laminar-turbulent transition under conditions comparable to flight. Major accomplishments include: (1) the design, fabrication, and performance-evaluation of a new kind of quiet tunnel, a quiet-flow Ludwieg tube; (2) the integration of pre-existing codes for nozzle design, 2D boundary-layer computation, and transition-estimation into a single user-friendly package for quiet-nozzle design; and (3) the design and preliminary evaluation of supersonic nozzles with square cross-section, as an alternative to conventional quiet-flow nozzles. After a brief summary of (1), a description of (2) is presented. Published work describing (3) is then summarized. The report concludes with a description of recent results for the Tollmien-Schlichting and Görtler instability in one of the square nozzles previously analyzed.

1 The Purdue Quiet-Flow Ludwieg Tube

A quiet-flow Ludwieg tube has been designed, constructed, and tested at Purdue over the past 5 years. This is the first low-cost quiet-flow facility constructed with a design that has the potential to reach substantial quiet Reynolds numbers and test-section sizes. It is also the first quiet-flow facility with good optical access (except for the defunct JPL tunnel [9] and the small MSU facility [7]). The facility

is currently in operation with a 3.8 by 4.3 inch Mach 4 test section that is quiet to a length Reynolds number that exceeds 400,000. This Reynolds number already allows testing of receptivity and roughness effects. More importantly, the way is now open to development of substantially higher Reynolds number facilities with short run-times, at low costs. Current plans call for the installation in the Purdue Ludwieg tube of the obsolete first-generation nozzle blocks for the NASA Langley Mach 3.5 pilot tunnel. These blocks, which have a 6 by 10 inch exit, should allow achieving a quiet length Reynolds number of 2.7 million at 30 psia stagnation pressure, according to reference [4].

The design and construction of the facility is documented in references [13], [15], and [16]. The test results were presented in reference [17], and have been accepted for journal publication [14]. More recent results were presented in reference [19].

2 Integrated Software for Design of 2D and Axisymmetric Quiet-Flow Supersonic Wind Tunnel Nozzles

At the commencement of this effort, it was expected that a new nozzle would have to be designed and constructed for the Purdue Ludwieg tube. Although this has not yet been required after all, considerable effort has been expended in developing an integrated, easy to use quiet-nozzle design system from the several separate codes that were originally in use for this purpose at Langley. The following section briefly describes and documents this relatively user-friendly quiet-nozzle design software.

2.1 Method-of-Characteristics Nozzle-Design Code

The state of the art in the design of supersonic wind tunnel nozzles involves the use of 2D or axisymmetric method-of-characteristics (MOC) codes for determining the nozzle shapes that result in uniform exit flow. A good introductory description of the basic problems is presented in sections 15-5 and 16-4 of reference [23]. Although fully three-dimensional MOC codes exist, they would have to be iterated in order to produce uniform flow at the nozzle exit, a basic requirement for wind tunnel nozzles [12]. Thus, consideration is here restricted to flows derivable from 2D or axisymmetric MOC solutions.

The Sivells wind-tunnel nozzle design code was selected for use in 1990. Unlike the custom-modified code used by Chen for the Langley nozzle designs [21], it is fairly well documented in reference [20], and the source code is available. Although it is old FORTRAN-IV code, it is possible to follow much of the logic from the

code itself. In addition, the Sivells code is capable of generating nozzles with the radial-flow regions that are advantageous from a Görtler stability viewpoint [5].

The manual for the code devotes 8 pages to the description of the input deck, which allows for many different combinations of nozzle geometries. To simplify the use of the code, a FORTRAN program was written to automatically generate the input deck from the answers to a few simple questions that are answered interactively. This program, `SIVINPUT`, which is appended, sets most of the parameters in the code. The internal documentation contained in `SIVINPUT` in the form of comments will have to suffice for documentation at the present time.

Numerous modifications have been made to the Sivells code over the past 4 years, although most of these are fairly minor. The mainframe-style input/output structure was modified to a friendlier form, in which the input and output files are automatically generated by adding 3 character suffixes to a user-supplied 8 character rootname. If the rootname is here called `username`, then `SIVINPUT` generates `username.inp` which is then read by `SIVELLS`. A subroutine (`WRTOBL`; see appendix) was added to write selected output in a form suitable for ready translation into Harris code input (file `username.bl`). The `WRTOBL` subroutine also writes data for the conditions along the centerline of the sidewalls to the file `username.cl`, and integrates the tracks of mach lines originating on the sidewall, for determining the width of quiet-flow regions in 2D nozzles. The `WRTOBL` subroutine also writes out the derivatives of the wall contour, as generated in the MOC algorithm, in order to obtain accurate values of the local wall curvature for Görtler computations. Finally, `WRTOBL` was also modified to call the Hopkins-Hill subroutines developed and documented in reference [2], in order to determine the shape of the bleed slot lip upstream of the throat. Although this means that a different algorithm is used to determine the transonic throat shape upstream and downstream of the throat, the two algorithms seem to merge smoothly and accurately into one another. This agreement is not surprising, since both the upstream and downstream transonic flow algorithms are based on near-sonic perturbation theory, so they must agree very well near the throat where the sound speed is nearly sonic, at least for large radius of curvature wind-tunnel-type throats. Error trapping code was inserted to address problems with user-friendliness, when these were encountered. The common block references were made consistent to reduce compiler difficulties. The array sizes allowed by the code were also increased, to improve accuracy. Minor changes were also made to `SIVELLS` to allow restructuring it to structured Fortran-77 using the commercial software package `FOR-STRUCT` (Cobalt Blue, Inc., Roswell, Ga.).

The only significant change to the Sivells internal algorithm was made when it was determined that the code would not generate internal streamlines downstream of the radial flow region, for the nozzle shapes desired (the original code will only generate internal streamlines for the special case where `ETAD` = 60.) This bug was

fixed by duplicating lines 111 and 112 of the `AXIAL` subroutine above line 43 in `AXIAL`.

Since a nozzle with 10 internal streamlines for square nozzle designs can be generated on a 66MHz 486-class PC in less than 3 minutes, the system is very practical for design work.

2.2 Sivells to Harris Interface Code

The program `MAKEBLIN` was written to take the output file from the Sivells code, `username.bl`, and generate an input file for the Harris code, `username.bli`. The Reynolds number scaling information required is read in from the auxiliary file `username.re`, which contains the throat radius, total pressure and temperature, and so on, and must be hand-generated. It uses various defaults to generate a complete input deck for the Harris code, including the streamwise grid-point locations. It is also capable of generating Harris code input for other inviscid-flow generators. The code is heavily commented, as can be seen from the listing in the appendix. The highly automatic generation of the complex input files required for the Harris code make design studies relatively easy to carry out. This code runs in seconds on the 66MHz PC.

2.3 Harris Boundary-Layer Code

The Harris boundary-layer code is documented with a good user's manual [8] and is written in structured Fortran. It is a good and standard finite difference code with which to compute 2D and axisymmetric boundary layers. For quiet-nozzle work, the boundary layers are assumed to be laminar (since the nozzle-wall boundary layer is only of interest up to the point where it becomes transitional). Thus, the turbulence model incorporated in the code is not an issue.

Again, minor modifications were made to this code, to ease input/output. The input/output files are again automatically generated by appending suffixes to the user-supplied rootname (e.g., `username`). The input data is read from `username.bli`, and the standard output is written to the file `username.blo`. In addition, the surface conditions written using `IPRT` commands are written in tabular form to `username.prt`, and the profiles written using `IPRO` commands are written in tabular form to `username.pro`. This allows rapid plotting of selected surface conditions, and eases translation of the output into a form suitable for the transition-estimation code. In particular, selected derivatives of profile quantities are written to `username.pro`, as generated in the program, so that they can be passed to the transition-estimation program with greater accuracy. Error-trapping code was also added, as needed to

ease use. The modified code has been used in required undergraduate course projects with good success.

This code also runs in a few minutes on the 66 MHz PC, so again the system is convenient for design purposes.

2.4 Harris to e**MALIK Translation Code

The program `BLTOSTAB` was written to take the output of the Harris code and translate it into the binary file form used by the e**MALIK code. This relatively long code generates the sophisticated binary input file, `username.bfl`, required by the stability code e**MALIK. It does this by reading the Harris code output files, `username.prt` and `username.pro`, along with the wall-curvature data saved in `username.bl` from the Sivells code, and the Reynolds number scaling data saved in `username.re`. Besides the main binary output file, `BLTOSTAB` generates `username.cur` for checking the surface curvature computations, `username.gor` for printing Görtler number data, and `username.sck` for printing information to check the computations for Sivells input data. `BLTOSTAB` can also read in surface geometry data that is then differentiated numerically to determine radii of curvature; this feature was implemented to allow running the Chen test case for Görtler instability, described below. In this case, the file `username.usk` is generated for checking computations performed using data not obtained from the Sivells code. A listing of `BLTOSTAB` is appended. For lack of resources, this heavily commented listing must suffice for documentation.

2.5 Transition-Estimation Code

The Langley program e**MALIK was used to perform e**N estimates of transition location [11, 10]. Although the manual, reference [11], is old and imprecise, it is apparently the best version in existence. The manual has been supplemented by examination of the source code, and private communications with Mujeeb Malik, Robert Spall, and Scott Anders. Because of the uncertainties of using this complex code with limited documentation, test cases were run for both the T-S and Görtler instabilities, to confirm proper operation.

The e**MALIK code was also modified slightly, again only to change the input/output formatting somewhat, and to add error trapping code. The code reads stability computation instructions from a file piped into the executable with the Unix `<` command, and writes the general output to a file specified with the Unix `>` command. It writes summary data to a file `username.sum`; the string `username` is passed to the code through an additional read statement at the bottom of the input file. The binary-form input data for the boundary-layer profiles is read from

`username.bfl`. The code was also modified to automatically loop through several different frequencies or spanwise wavenumbers in one execution, a property the version furnished to this author in 1991 did not have, although the manual suggested that it did. This is the most CPU-intensive code in this quiet-nozzle design system. The author has most recently been executing it on his department server, a 4-CPU Sun Sparcstation 1000 that has 256 megabytes of RAM. A half-dozen different frequencies can be studied over few dozen streamwise stations in typical runs that can usually be carried out overnight. Thus, although this code is CPU-intensive, it remains practical to perform design studies with it in a modern workstation environment.

2.5.1 Test Case for T-S Instability Computations

Use of the e**MALIK code for computation of T-S instabilities was benchmarked back in the summer of 1990, using test case 6 from reference [10]. Test case 6 is for the flow on a flat plate at Mach 4.5, $R = \sqrt{U_e x/\nu_e} = 1500$, adiabatic wall, and a total temperature of 1100 Rankine. Table IX on p. 407 of reference [10] gives spatial case results for a nondimensional frequency $\omega = 0.23$. There, the eigenvalue α is given as (0.2534081, -0.0024932) for the most accurate computation. The author's version of e**MALIK was first checked by running it with the internal self-similar boundary-layer solver, which produced (0.253397, -0.00250489). This agreement was considered to be very good. However, when the self-similar boundary-layer profiles were generated by the author of this paper, translated into e**MALIK input form by `BLTOSTAB`, and the stability was recomputed, the results were not as good. In this case, at a slightly different $R = 1495$, α was found to be (0.25195,-0.00229). This 10 percent difference was doubtless caused by imprecise generation of the boundary-layer profiles; the overall agreement indicates that the `BLTOSTAB` code generates the proper e**MALIK input. At the time of this test, the `BLTOSTAB` code differentiated the Harris velocity profiles numerically, instead of using the internally-generated Harris derivatives, so agreement would no doubt be better at the present time.

2.5.2 Test Case for Görtler Instability Computations

Benchmark data for the Görtler test-case was generously supplied by Frank Chen, along with data for the nozzle coordinates and pressure distribution. The data is for the Mach 6 NTC nozzle currently in use at Langley [6], at a total pressure of 100 psia and a total temperature of 360F, under adiabatic wall conditions. The summary output file printed by the e**MALIK code was supplied, although detailed data on the boundary-layer computations performed by Chen were no longer available. This test-case tests both the boundary-layer computations (carried out in both cases

with the Harris code) and the use of the e**MALIK code. Since the Chen nozzle design uses a modified Nelms code, and the system described here uses the Sivells code, the MOC nozzle-design code itself is not tested. Since the Chen data for the surface coordinates are given to four places, without derivatives, auxiliary code was added to the BLTOSTAB routine to allow reading in this non-Sivells coordinate data. The Chen coordinate data was differentiated using the same LaRC routine used by Chen, CSDS, which was obtained from LaRC computing center personnel.

Figure 1 shows the radius of curvature, R, as a function of the axial location z, which is zero at the throat. Note that z is the coordinate along the centerline of the tunnel, not the arclength s along the nozzle wall. Smooth distributions are obtained in the concave region downstream of about $z = 8$ inches, but the distributions become noisy near the downstream end of the nozzle. Additional smoothing in the differentiation might have reduced this effect. The precise curvature distributions computed by Chen were unavailable. The Harris code was run using 870

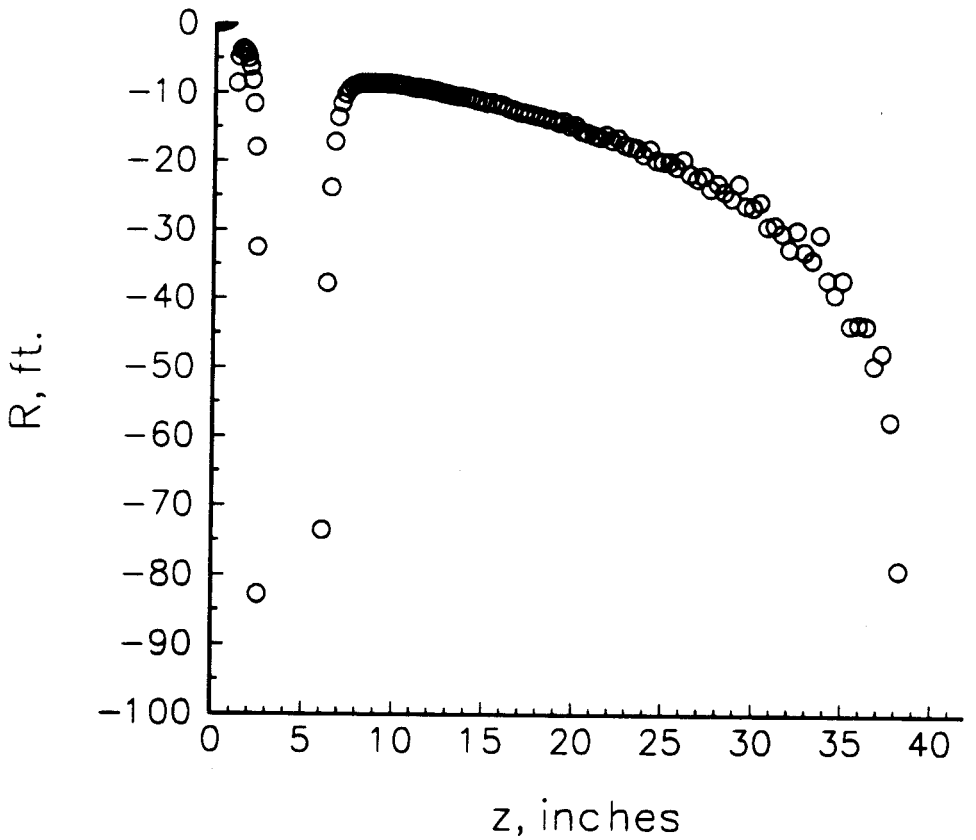

Figure 1: Radius of Curvature for Mach 6 Test Case

streamwise stations and 51 points in the wall-normal direction. Figure 2 shows the edge Mach numbers, M_e, interpolated by the Harris code to the Chen-supplied data. The limited precision of the data supplied causes the Harris code to generate some

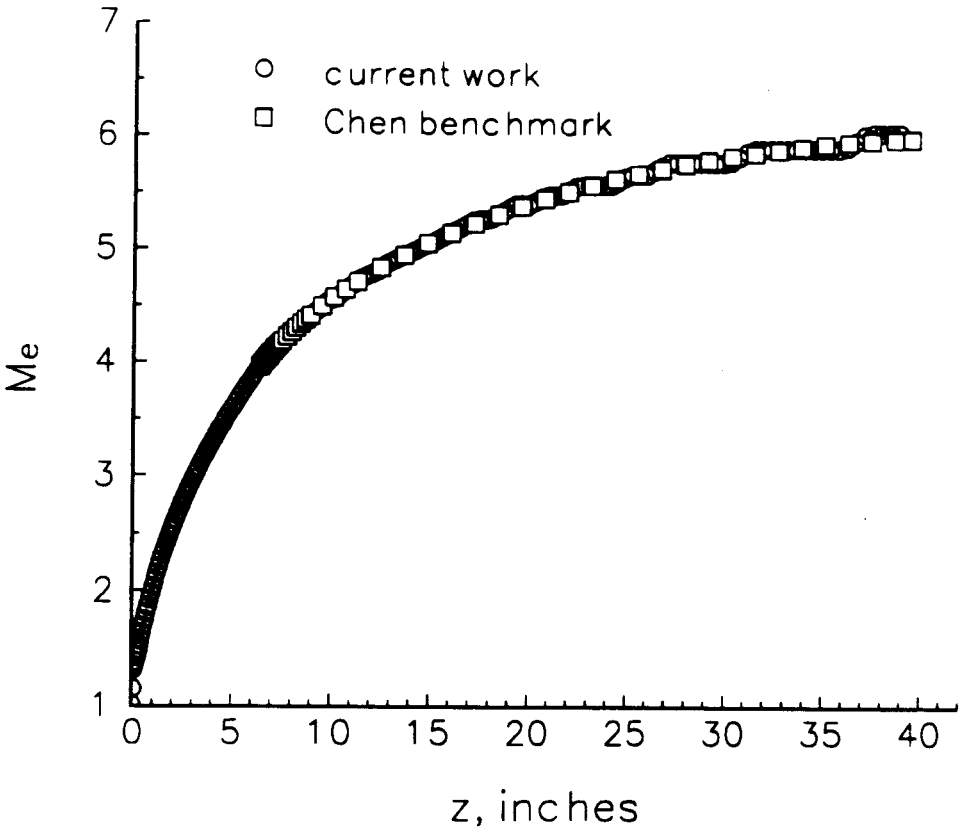

Figure 2: Comparison of Edge Mach Number for Mach 6 Test Case

waviness while interpolating. Figure 3 shows the boundary-layer momentum thickness, Re_θ, comparison. The waviness in the M_e interpolation clearly causes some waviness in the momentum thickness Reynolds number, Re_θ, and the current results are somewhat above those used by Chen. The cause of the small difference is difficult to determine, since the details of the Chen computation are no longer available. Both the wall radius of curvature and the momentum thickness are reflected in the Görtler number computations, presented in Figure 4. Here, the Görtler number G_θ is based on the momentum thickness θ. Overall, the agreement is good, although the interpolations and differentiations required in the re-analysis of the Chen data clearly cause additional scatter. No smoothing has been applied to these computations. Finally, Figure 5 shows the results of the N-factor computations carried out

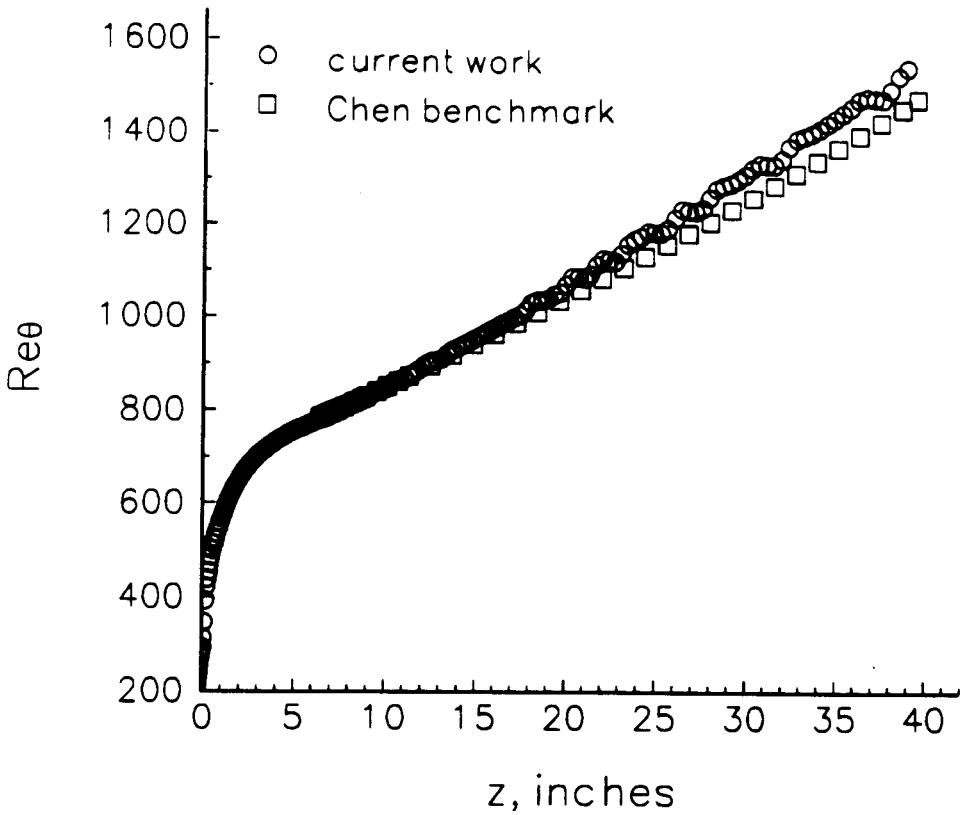

Figure 3: Comparison of Re_θ for Mach 6 Test Case

using the e**MALIK code, for the case with 125 waves around the circumference of the nozzle, which is nearly the most amplified case. Here, s is the arclength along the nozzle wall, with $s = 0$ being located at the throat. Clearly, the operation of the Harris, BLTOSTAB, and e**MALIK codes is fundamentally correct, for the results agree well. Although the close agreement is clearly in some part fortuitous, given the minor disagreements in the previous plots, the test-case shows that this part of the integrated and fairly automatic software produces the proper results.

2.6 Summary of Software Status

This software is now working and fairly well tested. With this semi-automatic software, it is possible to generate nozzle designs in seconds for 2D and axisymmetric nozzles, although square nozzles take longer. The boundary layers on 2D and axisymmetric nozzles can be computed in minutes on a PC, and then a few overnight computations on a modern workstation can be performed to estimate the location

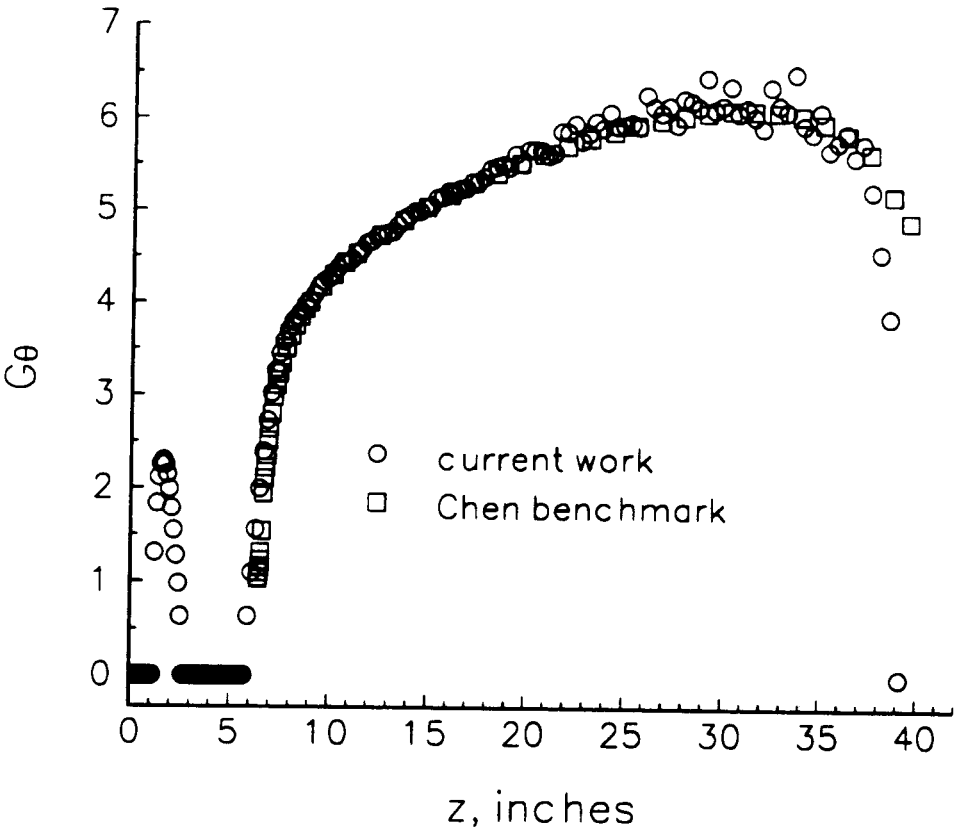

Figure 4: Comparison of G_θ for Mach 6 Test Case

of transition. Thus, one complete iteration of a quiet nozzle design can be carried out in a workstation environment in a few days, using only an hour or two per day of engineering labor.

3 Design and Preliminary Evaluation of Supersonic Wind Tunnel Nozzles with Square Cross Sections, for Use in Quiet-Flow Facilities

3.1 Introduction

This effort began in September 1992, and continued intermittently until the termination of the grant. Sivell's code was again used for the method-of-characteristics (MOC) design of the nozzles [20]. Although reference [20] suggests that the code

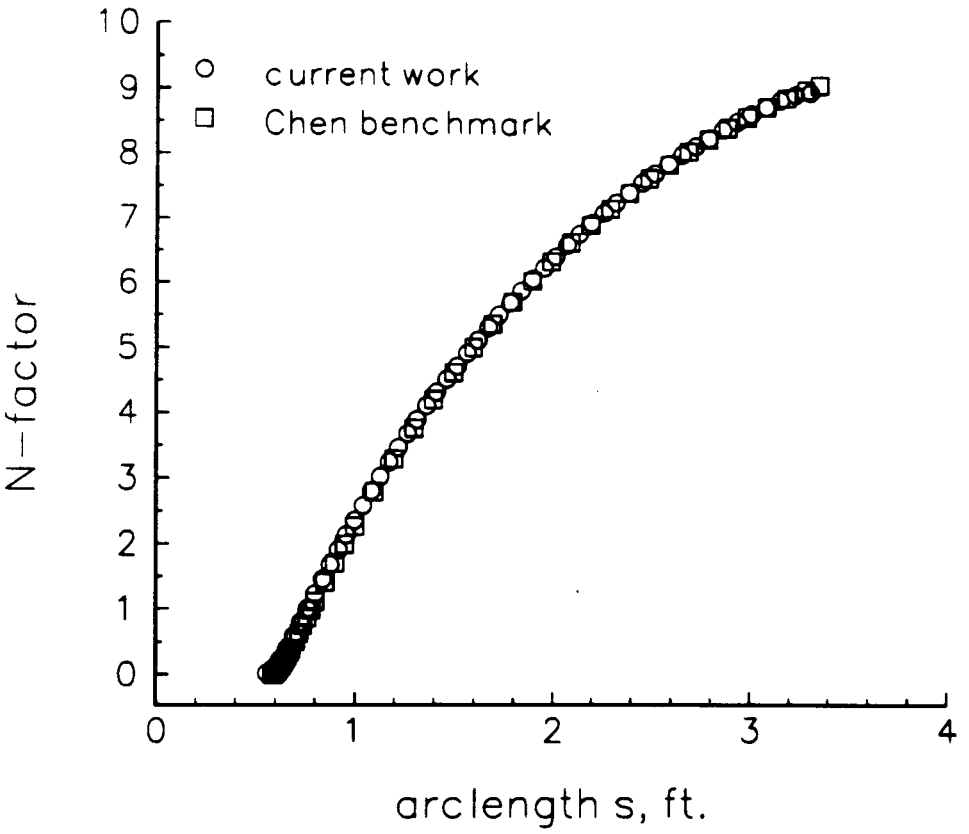

Figure 5: Comparison of Görtler N-factors for Mach 6 Test Case

will generate internal flow streamlines for all the types of nozzles which the code can generate, minor modifications proved to be necessary in order to generate internal streamlines for nozzles that included radial-flow regions. A post-processing code was then written to trace the streamlines upstream from a square cross-section drawn in the nozzle-exit plane, in order to generate near-square cross-sections from the exact MOC results, between the exit and the throat. This post-processing code was completed in fall 1992 and documented in reference [18]. The Hopkins-Hill technique for designing the transonic region of the nozzles (up to the bleed slot lip) still needed to be implemented in a reasonably convenient way. This work was completed during early summer 1993 and documented in reference [2]. Graduate student Timothy Alcenius carried out the Hopkins-Hill work, and then performed 3D Navier-Stokes computations of the mean flow in the square nozzles. The position of transition due to the crossflow instability was estimated using these computations and the cross-flow Reynolds number technique. Most of the Navier-Stokes results are available in

flow Reynolds number technique. Most of the Navier-Stokes results are available in reference [1], which appeared in June 1994. Complete results are available in reference [3], which will be appended to the final report for NAG-1-1607. A summary of the results has been submitted for publication in the AIAA Journal of Aircraft.

Although the above analyses suggested that crossflow instability would be the dominant factor in transition of the boundary layers on the square nozzle walls, computations of the Tollmien-Schlichting instability and the Görtler instability were also carried out using the e**MALIK code [10]. Although the actual nozzle-wall boundary layer is three-dimensional, these computations were carried out using the results of a 2D boundary-layer computation of the sidewall boundary layer in the symmetry plane. Analyses were only carried out for the long Mach 2.4 nozzle described in [1], since this was thought to be the worst case for these instability modes, and since the short Mach 2.4 nozzle appeared less practical from the standpoint of the crossflow instability. A detailed description of these computations forms the remainder of this section.

3.2 Computations of the Centerplane Boundary Layer in the Long Mach 2.4 Square Nozzle

The coordinates of the long Mach 2.4 nozzle analyzed by Alcenius are given on p. 129 of the appendix of reference [3]. These coordinates were obtained by Alcenius using `SIVINPUT` and the Sivells design program [20]. The input parameters are specified on p. 23 of reference [3] (Nozzle 2 of Table 1). It should be noted that the nozzle design codes produce a nozzle with a bleed slot lip that protrudes far upstream of the throat. Alcenius then cut off the upstream extent of the bleed slot, according to specifications provided by Ivan Beckwith from NASA Langley, in order to produce a bleed slot originating 0.175 meters (0.574 ft.) upstream of the throat (p. 129 of reference [3]). For the computations presented here, the bleed slot was cut off 0.612 ft. upstream of the throat; interpolation to the exact position used by Alcenius was not performed. Figure 6 shows that the coordinates are identical, except for the difference in leading and trailing extent (Alcenius's last point is at 4.197 ft., the last point used here is 4.165 ft.). Here, z is again the coordinate along the nozzle centerline, beginning at the throat, and the y-axis is normal to the z-axis. Figure 7 shows the displacement thickness ($\delta^\star$) of the 2D boundary layer calculated using the centerplane pressure distribution and the Harris code [8]; the close agreement shows that the methods used were the same. Although the small discrepancy about 3 ft. downstream of the throat is somewhat troubling, this slight variation in a strong favorable pressure gradient should not have much effect on the stability. Although the point is not discussed in reference [3], it should be noted that both computations

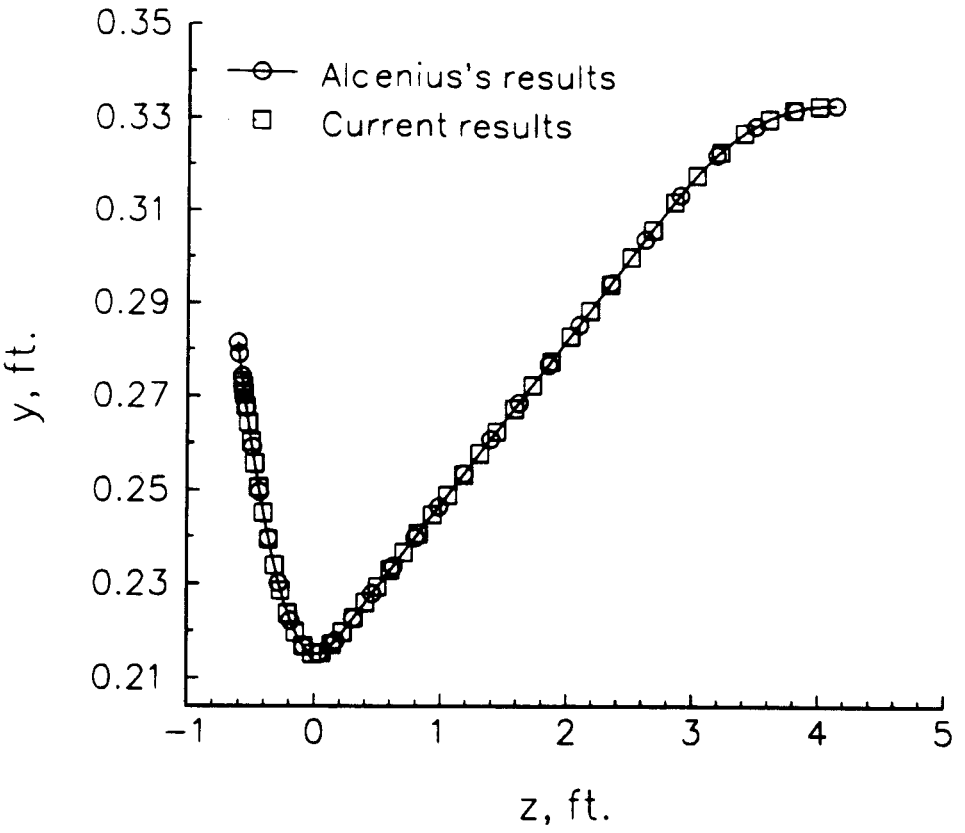

Figure 6: Coordinates of Long Mach 2.4 Square Nozzle

computations. Also, both computations presented here were carried out assuming an isothermal wall temperature of 520 Rankine. Since this is nearly equal to the stagnation temperature, 540 R, and the Mach number is not high, the conditions are nearly adiabatic. The stagnation pressure is taken as 100 psia, as in Alcenius's work.

3.3 Estimates of Transition in the Long Mach 2.4 Square Nozzle due to the Görtler Instability

This instability was analyzed with the e**MALIK code, using the nozzle-design software described above. The boundary-layer data were read in through the external binary file generated from the Harris code output by the translation program `BLTOSTAB`. The body was assumed 2D, 71 grid points were used in global computations and 141 in local computations, and computations were begun at the location

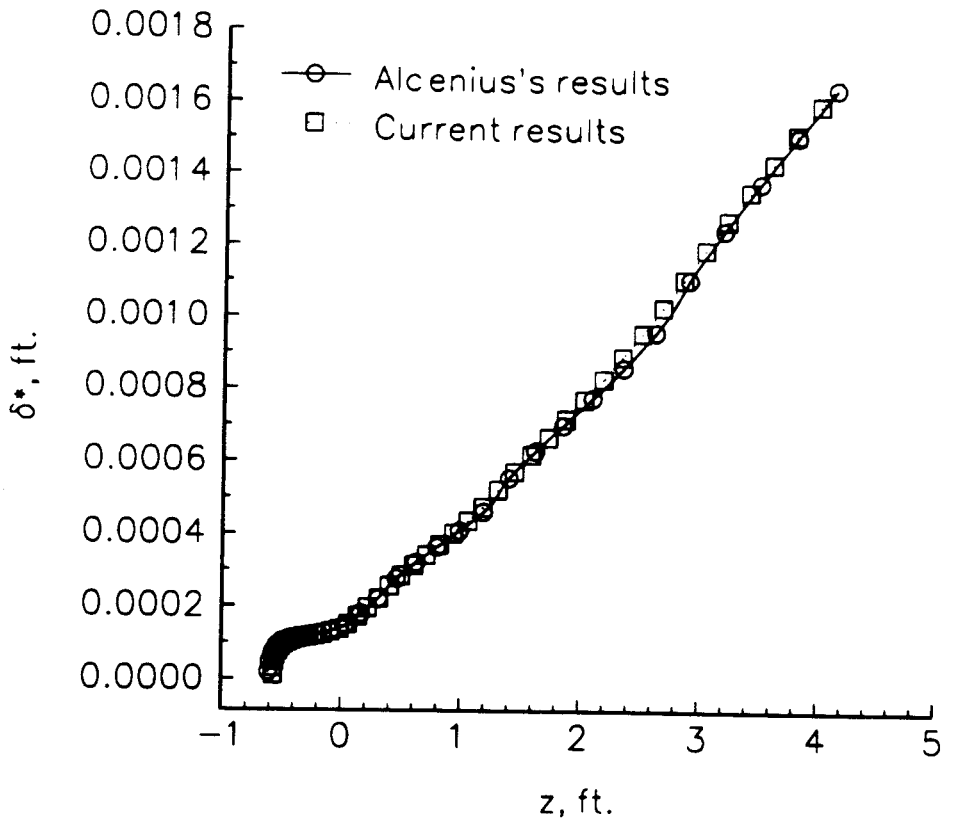

Figure 7: Comparison of Centerplane δ^* Values for Long Mach 2.4 Square Nozzle

where the curvature becomes concave. Since **ibeta** was set to 0, the input parameter **betx** is actually the dimensional wavenumber, k, per ft. (note that the e**MALIK manual [11] is erroneous in this regard; the correction was provided verbally by M. Malik on 16 July 1991). This wavenumber parameter was varied through 500, 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800, 3000, and 4000 per ft. Since the momentum thickness θ of the boundary layer at the first concave station was about 2×10^{-4} ft., and the wavelength $\lambda = 2\pi/k$, this corresponds to spanwise wavelengths λ/θ ranging from 60 to 8. The corresponding values of **BETA** output by the code range from 0.19 to 1.9; these spanwise wavenumber values are non-dimensionalized by the distance from the leading edge and the edge values of the fluid properties. The spanwise wavelength of the Görtler waves is maintained constant as the amplification is integrated downstream. Figure 8 summarizes the integrated amplification between the streamwise distances of 2.80 and 4.17 ft. The peak amplification is about 5.7, which occurs at a **BETA** of 0.9. Based on the Langley

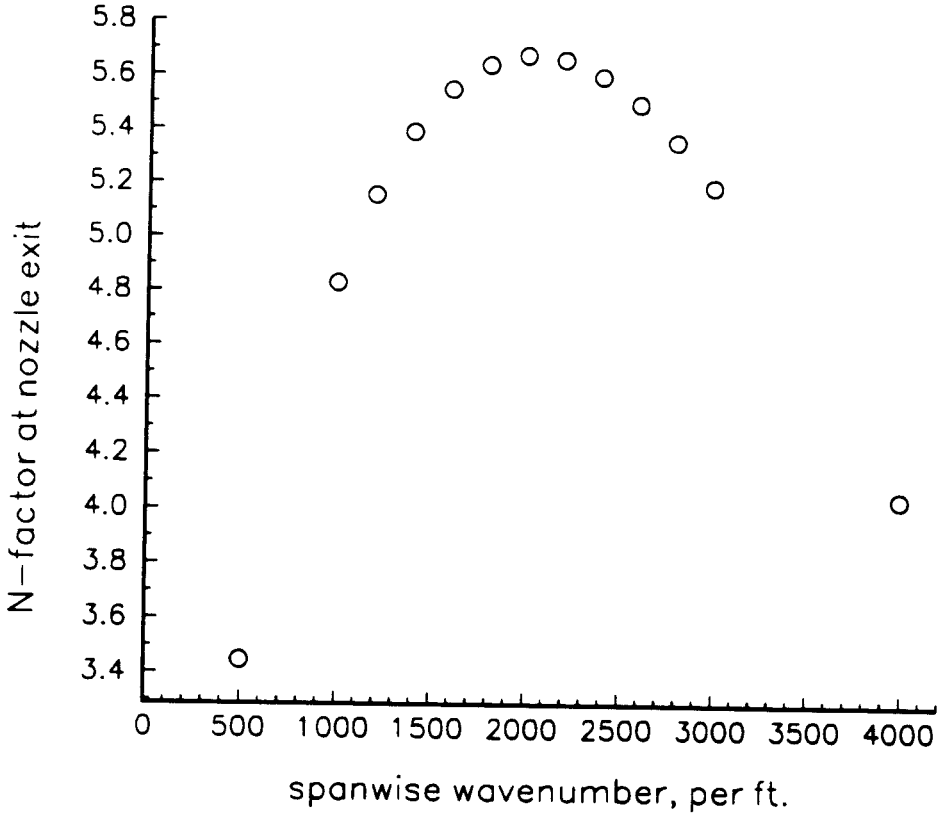

Figure 8: Görtler N-factors at Exit of Long Mach 2.4 Square Nozzle

N-factor criterion of 7.5 for Görtler transition [22], this instability is not expected to cause transition. Of course, this estimate neglects interactions with roughness or other instabilities.

3.4 Estimates of Transition due to the T-S Instability

The T-S computations were performed with the e**MALIK code, and with the same boundary-layer profiles used for the Görtler computations. Many computations had to be performed in order to find the unstable range of frequencies. The results are shown in Figure 9. Frequencies in the neighborhood of 6 kHz are most unstable. These have a wave-angle PSI of about 67 degrees, which appears to be reasonable. The nondimensional frequency OMEGA is about 0.009 for these waves. Since the integrated N-factor is less than 2, it appears unlikely that T-S instability would contribute substantially to transition in this nozzle.

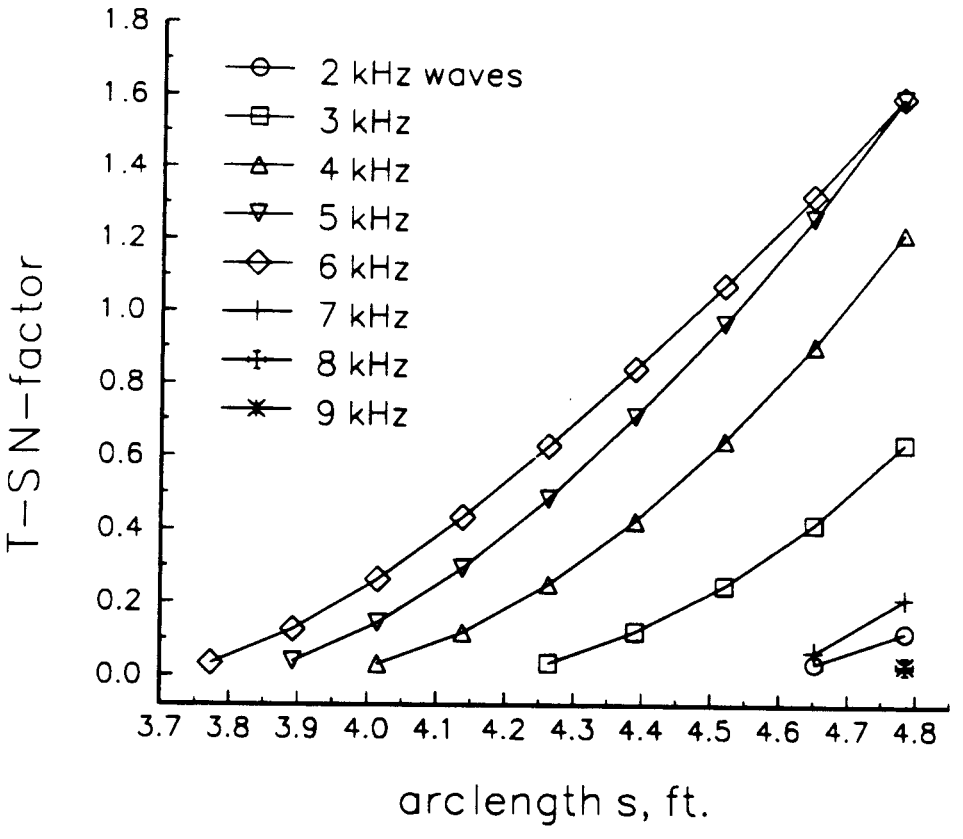

Figure 9: T-S N-Factors in Long Mach 2.4 Square Nozzle

3.5 Summary of Square-Nozzle Design Effort

The design, construction, and shakedown of high speed quiet-flow nozzles is a difficult task in which many poorly understood tradeoffs have to be made. All of the difficult high-speed transition phenomena that the nozzles are intended to study must be estimated in order to best design the nozzles. Fabrication costs are difficult to estimate in advance.

The square nozzle concept put forward by Ivan Beckwith of NASA Langley was suggested as a possible method of obtaining the advantages of both axisymmetric and rectangular (2D) nozzles. The walls are flat downstream of the expansion, facilitating the use of windows. The nozzle can be disassembled to facilitate polishing and maintenance of the critical throat region. High Mach number square nozzles can be machined without excessive difficulties with throat tolerances, and contour flaws do not focus to the centerline. All four walls see the same accelerating flow, so it was hoped that crossflow would not be a major problem.

facilitating the use of windows. The nozzle can be disassembled to facilitate polishing and maintenance of the critical throat region. High Mach number square nozzles can be machined without excessive difficulties with throat tolerances, and contour flaws do not focus to the centerline. All four walls see the same accelerating flow, so it was hoped that crossflow would not be a major problem.

The studies carried out to date do not support the high hopes initially conceived. Estimates based on the crossflow Reynolds number (Alcenius) suggested that crossflow would be a major difficulty in these nozzles, even in the throat. These estimates have apparently been confirmed by unpublished e**N estimates carried out by the High Technology Corp. under the SBIR program. It remains possible that the throat-region crossflow-transition problem identified by these computations could be ameliorated by moving the bleed slot downstream. The cost of machining such a nozzle remains a topic of speculation. It may still be that square nozzles will prove to be more cost-effective than axisymmetric or 2D designs. The research to date seems somewhat discouraging but far from conclusive.

4 Measurements of Crossflow Instability on the Sidewalls of the LaRC Mach 3.5 Low Disturbance Tunnel

Part of the proposed crossflow investigations involved measurements of the flat sidewall boundary layers in the Mach 3.5 low disturbance tunnel at Langley. These were to be carried out during summer 1994 by Christine Haven, as part of her M.S. thesis at Purdue. One and a half days of tunnel access were provided in the middle of the summer, after about a month of preparation. Although the hot wires, traverse system, and controller software worked acceptably, major difficulties with electronic noise were encountered. Although the cause of these difficulties was later determined, the problem could not be solved in time to allow obtaining low-noise data. The high noise level and the limited amount of data that could be obtained has precluded any attempt to draw definite conclusions from this work. A full report will be provided with the final report on Langley grant NAG-1-1607, which should be forthcoming in a few months.

5 Summary

This grant enabled the successful development of a new kind of low cost quiet-flow wind tunnel at Purdue. Since this wind tunnel is the basis for current AFOSR-

supported research, the outcome of the grant has involved successful technology-transfer to AFOSR. The grant also supported the development of relatively efficient and user-friendly software for quiet nozzle design, although the full potential of this software has yet to be put to use. Finally, the grant supported the design and preliminary evaluation of the new square nozzle concept for quiet-flow wind tunnels. Although the preliminary evaluations of this concept are somewhat discouraging, further research might yet show that square nozzles will be an advantageous choice in certain circumstances.

6 Acknowledgements

Part of the work described here was carried out by Timothy Alcenius and Christine Haven, whose M.S. thesis research was in part supported under NASA Langley grant NAG-1-1607. The progress described here would not have been possible without the close cooperation and encouragement of Ivan Beckwith, Steve Wilkinson, Dennis Bushnell, Frank Chen, and other Langley personnel. Construction of the Purdue Ludwieg tube was supported in part by a gift in memory of K. H. Hobbie. And, although Langley support for this work is at present terminating, the Purdue work in the area of high-speed transition is able to continue, thanks to support from AFOSR under grants F49620-94-1-0067 and F49620-94-1-0326, monitored by Dr. L. Sakell.

References

[1] Timothy Alcenius, S.P. Schneider, Ivan E. Beckwith, and John J. Korte. Development of square nozzles for high-speed low-disturbance wind tunnels. Paper 94-2578, AIAA, June 1994.

[2] Timothy Alcenius and Steven P. Schneider. Development of a code for wall contour design in the transonic region of axisymmetric and square nozzles. Contractor Report CR-194857, NASA, January 1994.

[3] Timothy John Alcenius. Development of square nozzles for high-speed, low-disturbance wind tunnels. Master's thesis, School of Aeronautics and Astronautics, Purdue University, December 1994. Included as an appendix to NASA CR-198040.

[4] I.E. Beckwith and W.O. Moore III. Mean flow and noise measurements in a Mach-3.5 pilot quiet tunnel. Paper 82-0569, AIAA, March 1982.

[5] F.J. Chen, M.R. Malik, and I.E. Beckwith. Advanced Mach-3.5 axisymmetric quiet nozzle. Paper 90-1592, AIAA, June 1990.

[6] F.J. Chen, S.P. Wilkinson, and I.E. Beckwith. Görtler instability and hypersonic quiet nozzle design. Paper 91-1648, AIAA, June 1991.

[7] A. Demetriades, R.L. Mueller, D.C. Reda, and L.S. King. Boundary-layer studies in a 2-d supersonic nozzle for quiet-tunnel applications. Paper 94-2507, AIAA, June 1994.

[8] J.E. Harris and D.K. Blanchard. Computer program for solving laminar, transitional, or turbulent compressible boundary-layer equations for two-dimensional and axisymmetric flow. Technical Report NASA-TM-83207, NASA, February 1982.

[9] James M. Kendall. Experimental methods and results on supersonic boundary-layer stability and receptivity, 1995. To appear in the notes of a 1994 ICASE short course on stability and transition, to be published by Oxford University Press, edited by T. Corke.

[10] M.R. Malik. Numerical methods for hypersonic boundary layer stability. *Journal of Computational Physics*, 86:376–413, 1990.

[11] Mujeeb R. Malik. e**Malik: A new spatial stability analysis program for transition prediction using the e**N method. Technical Report HTC-8902, High Technology Corporation, March 1989. See also HTC-9203, which is almost identical.

[12] Victor H. Ransome, H. Doyle Thompson, and Joe D. Hoffman. Three-dimensional supersonic nozzle flowfield calculations. *J. Spacecraft and Rockets*, 7:458–462, 1970.

[13] S. P. Schneider. A quiet flow Ludwieg tube for study of transition in compressible boundary layers: Design and feasibility. Contractor Report CR-187374, NASA, November 1990. Semiannual Status Report for NASA Grant Number NAG-1-1133. See also Status Report, June 1991, NASA CR-188499.

[14] S. P. Schneider and C. E. Haven. Quiet-flow Ludwieg tube for high-speed transition research. *AIAA Journal*, 33(4):688–693, April 1995.

[15] S.P. Schneider. A quiet-flow Ludwieg tube for experimental study of high speed boundary layer transition. Paper 91-5026, AIAA, December 1991. Gives details of design procedures.

[16] S.P. Schneider. A quiet-flow Ludwieg tube for experimental study of high speed boundary layer transition. Paper 92-3885, AIAA, July 1992. Gives detailed design information for facility as constructed.

[17] S.P. Schneider and C. E. Haven. Mean flow and noise measurements in the Purdue quiet-flow Ludwieg tube. Paper 94-0546, AIAA, January 1994.

[18] Steven P. Schneider. Method of characteristics design of a supersonic wind tunnel nozzle with square cross-section. Contractor Report CR-194359, NASA, January 1993.

[19] Steven P. Schneider, Christine E. Haven, Joseph B. McGuire, Steven H. Collicott, Dale Ladoon, and Laura A. Randall. High-speed laminar-turbulent transition research in the Purdue quiet-flow Ludwieg tube. Paper 94-2504, AIAA, June 1994.

[20] J.C. Sivells. A computer program for the aerodynamic design of axisymmetric and planar nozzles for supersonic and hypersonic wind tunnels. Technical Report AEDC-TR-78-63, Arnold Engineering Development Center, December 1978.

[21] S. P. Wilkinson, S. G. Anders, and F.-J. Chen. Status of Langley quiet flow facility developments. Paper 94-2498, AIAA, June 1994.

[22] S.P. Wilkinson, S.G. Anders, F.-J. Chen, and I.E. Beckwith. Supersonic and hypersonic quiet tunnel technology at NASA Langley. Paper 92-3908, AIAA, July 1992.

[23] M.J. Zucrow and J.D. Hoffman. *Gas Dynamics, Volume II: Multidimensional Flow*. Robert Krieger, Malabar, Florida, 1985.

A Source Code for SIVINPUT

```
*  PROGRAM SIVINPUT.  Fortran-77.  For generating input deck to
*  Sivells nozzle design code (AEDC-TR-78-63).
*
*  emailed back 'From moen Wed Nov 25 12:00:09 1992' after use by him.
*
c     this program writes an input file in the correct format for
c     sivells code; sps 6-27-90
c     modified 9-30-92 for the square nozzle work with streamlines sps
c     modified to produce starting streamlines on nozzle wall.
*     mod sps 11-30-92 to make mp=5
*     mod 11-30 sps to allow choices of IN, XC, IX
*     add header info. 1-16-95, this is the current version used by Alcenius
*       for his MS thesis and square nozzle computations (see appendix of
*       Alcenius MS thesis).
c
      character*10 title
      character*20 sivfile
c
      write (*,*) 'enter a rootname to write sivells input file:'
      read (*,10) sivfile
      title = sivfile
      ileng = index(sivfile,'  ') -1
      sivfile(ileng+1:ileng+4) = '.inp'
      write (*,*) 'opening file-',sivfile,'-for output'
      open(unit=2,file=sivfile,status='new')
c
*     write (*,*) 'enter title of run (10 characters): '
*     read (*,20) title
      write (*,*) 'enter jd (-1=2D, 0=axisym): '
      read (*,*) jd
      write (2,30) title,jd
*     write (*,*) 'enter sfoa:'
*     read (*,*) sfoa
      sfoa=0.   !use 3rd or 4th degree distribution
      gam = 1.40
      ar = 1716.563
      zo=1
c     following three used in bl computations, not used here
      ro=1
      visc=1
      vism=1
c
      xbl=1000.  !gives values at evenly spaced intervals
      write (2,40) gam,ar,zo,ro,visc,vism,sfoa,xbl
c
      write (*,*) 'enter etad,rc,bmach,cmc: '
```

```
      read (*,*) etad,rc,bmach,cmc
      xc=0.  !so 4th degree distribution, change?
*     write (*,*) 'enter xc, in: '
*     read (*,*) xc,in
*     write (*,*) 'xc,in= ',xc,in
      fmach=0.  !this sets distribution, change?
      sf = 0.   !nozzle throat radius = 1.0
      pp = 0.   !coordinates given relative to throat
      write (2,50) etad,rc,fmach,bmach,cmc,sf,pp,xc
c      write (*,*) 'enter mt,nt,ix,in,md,nd,nf,mp,jc,lr,nx:'
c      read (*,*) mt,nt,ix,in,md,nd,nf,mp,jc,lr,nx
      mt=61    !pts on char. EG, max 125
      nt=31    !pts on axis IE, max 149-LR
*     write (*,*) 'enter ix: '
*     read (*,*) ix
      ix=0     !is 3rd deriv matched?  change?
      in=10    !use Mach no. distrib on BC, makes 2nd deriv match rad. flow
c                         change?
      iq=0     !calls for complete contour
      md=61    !pts on char. AB, max about 125, odd
      nd=15    !pts on axis BC, max about 150,
c                    changed from 49 to 15 sps 7-2-90
      write (*,*) 'enter -1 for smoothing, 1 for no smoothing: '
      read (*,*) ismooth
      nf=ismooth*81  !pts on characteristic CD.  Neg calls for smoothing
      mp=5    !pts on GA, conical section, if Fmach ne Bmach
      jc=0     !if not 0, used to print intermediate characteristics
      lr=31    !pts on throat char., - prints out transonic soln
      nx=13    !spacing of pts on axis upstream, this no. recc.
      mq=0     !pts downstream of D
      jb=-1    !neg for no BL computation
      jx=1     !pos calls for streamlines
      it=0     !jack points, not used
      write (2,60) mt,nt,ix,in,iq,md,nd,nf,mp,mq,jb,jx,jc,
     >               it,lr,nx
c
      if (ismooth .eq. -1) then
      noup=10  !smoothing parameters, arbitrary
      nodo=10
      npct=90
      write (2,70) noup,npct,nodo
      end if
c
c     gives streamline distribution that corresponds
c     to the half wall for conversion to a square nozzle.
c     note that the number of streamlines requested will
c     be reduced by one because Sivells automatically
c     calculates the wall streamline.  Sivells output
```

```
c     will have the actual number of streamlines requested.
c     (moen 10-92)
c
      write (*,*) 'How many streamlines along halfwall?'
      read (*,*) nstream
      nstream=nstream-1
      dx=1.0/(float(nstream)*sqrt(2.0))
      ycnt=1.0/sqrt(2.0)
      do 100 istream = 0,nstream-1
        etadstr = etad*sqrt((istream*dx)**2+ycnt**2) !see btm page 59
        qm = sqrt((istream*dx)**2+ycnt**2)
        xj = 1  !look for more streamlines
        write (2,90) etadstr,qm,xj
        if (ismooth .eq. -1) then
          write (2,70) noup,npct,nodo    !must have for each!!
        end if
  100 continue
c
      close(2)
      stop
   10 format(a20)
   20 format(a10)
   30 format(1x,a10,2x,i2)
   40 format(8(1x,f9.3))
   50 format(8(1x,f9.3))
   60 format(1x,i4,15(i5))
   70 format(1x,i4,2(i5))
   90 format(3(1x,f9.4),1x)
      end
```

B Output Subroutine Written for Sivells Code

The following is the subroutine added to the Sivell's code to generate output files for transfer to the boundary-layer code.

```
* subroutine wrtobl.for
* this subroutine modified from perfc by s schneider 6-90 to write sivells
*   output to a file to be read directly by a bl program.
* also computes the upstream contour from halls assumptions and rc
*    assumes throat is 0,0, and uses formulas from Hall, (3) and
* assumes sf=0.0 at beginning so throat radius is 1 unit
* modified 7-31-91 to give full reference to block CONTR same as elsewhere
* modified 9-29-92 to output multiple calls when writing streamlines,
*      also start z write at Mach > 1
* dimension arrays in parameter statement.  Note arrays must match with
*  other routines. Change small from 0.05 to 0.01
```

```
* modified 9-30-92 to find streamline shapes in transonic region
* mod 11-25-92 to increase array sizes and use parameter statements sps
* mod 6-4-93 to call hophill to do hopkins-hill upstream of throat, sps
* mod 9-1-94 to write out WALTAN and sd with contour for Gortler, sps
      SUBROUTINE wrtobl                                                      1
C                                                                            4
      parameter (mwr=300,maxtpt=150,mpt=500,mwk=400)
      IMPLICIT REAL*8(A-H,O-Z)                                               5
      character*20 blfile,clfile
      logical lopen
      dimension pep0(maxtpt)
* following single for passing to hophill
      real*4 rap,thetap,gamp,xendp,ytp
      real*4 xp(maxtpt),yp(maxtpt),amachp(maxtpt) !single for passing
*  (upstream extension of contour into subsonic-transonic region)
      save icall,xend,numpts,rap,thetap,gamp,xendp
      common /centerl/clx(mwr),clmach(mwr)
      COMMON /GG/ GAM,GM,G1,G2,G3,G4,G5,G6,G7,G8,G9,GA,RGA,QT                 6
      COMMON /CLINE/ AXIS(5,mwk),TAXI(5,mwk),WIP,X1,FRIP,ZONK,SEO,CSE        7
      COMMON /COORD/ S(mpt),FS(mpt),WALTAN(mpt),SD(mpt),WMN(mpt),TTR(mpt     8
     1),DMDX(mpt),SPR(mpt),DPX(mpt),SECD(mpt),XBIN,XCIN,GMA,GMB,GMC,GMD      9
* following not used here, messy common block, avoid if possible
*     COMMON /WORK/ A(5,150),B(5,150),FINAL(5,150),WALL(5,mpt),WAX(mpt),     10
*    1WAY(mpt),WAN(mpt)                                                     11
      COMMON /PROP/ AR,ZO,RO,VISC,VISM,SFOA,XBL,CONV                         12
      COMMON /PARAM/ ETAD,RC,AMACH,BMACH,CMACH,EMACH,GMACH,FRC,SF,WWO,WW     13
     1OP,QM,WE,CBET,XE,ETA,EPSI,BPSI,XO,YO,RRC,SDO,XB,XC,AH,PP,SE,TYE,XA     14
*     COMMON /TROAT/ FC(6,51)                                                15
*     COMMON /CONTR/ ITLE(3),IE,LR,IT,JB,JQ,JX,KAT,KBL,KING,KO,LV,NOCON,     16
*    1IN,MC,MCP,IP,IQ,ISE,JC,M,MP,MQ,N,NP,NF,NUT,nr,lc,md,mf,mt,nd,nt        17
* this is full common block reference to CONTR taken from routine AXIAL
      COMMON /CONTR/ ITLE(3),IE,LR,IT,JB,JQ,JX,KAT,KBL,KING,KO,LV,NOCON,AXI  12
     1IN,MC,MCP,IP,IQ,ISE,JC,M,MP,MQ,N,NP,NF,NUT,NR,LC,MD,MF,MT,ND,NT   AXI  13
*
* following common block filled by call to tcoeff, gives coefficients of
*   transonic series solution - sps
      common /transc/gr,gs,gt,gv,gk,u42,u22,u63,u43,u23,up2,up0,
     >  v42,v22,v02,v63,v43,v23,v03
      data tiny/1.0e-5/,eangle/10.0/ !degrees, must be gt. 0
* (allow 1 percent error in computation of upstream contour; this is
*     a small size of mach number....)
* Note that a choice of 0.05 for small gives amstar2 > possible in upstream
*  part of transonic solution near centerline, 9-29-92
* change small from 0.01 to 0.002 10-21-92, was giving problems when
*  Mike Moen was running mach 3.5 test cases in axisym
      data icall/1/
C
      if (icall .eq. 1) then
```

```
* (first call to wrtobl)
       inquire (unit=3,name=blfile,opened=lopen)
       if (lopen) then
          write (*,*) 'writing nozzle summary for bl read to ',blfile
       else
          write (*,*) 'no file open for writing bl info to!!!???'
          return
       end if
       inquire (unit=4,name=clfile,opened=lopen)
       if (lopen) then
          write (*,*) 'writing nozzle summary for cl to ',clfile
       else
          write (*,*) 'no file open for writing cl info to!!!???'
          return
       end if
     else
       write (*,*) 'wrtobl, call ',icall,' for streamline dump'
     end if
*
*   get some things needed for transonic case and for cl mach distribution
      if (ie .eq. 0) then
*     (planar)
         sigma = 0.0
      else
*     (axisymmetric)
         sigma = 1.0
      end if
      s1 = rc + 1
      alambd = sqrt( (1+sigma)/( (gam+1)*s1 ) )
*
* write out the cl mach number distribution info, for first call only
*   subsequent calls are to write streamline data
*
      if (icall .eq. 1) then
* first, compute the number of points already in the array:
*   (this is necessary because of overlap and half-filling in method)
        numpts = 1
        do 10 i = 2,mwr
          if (clx(i) .gt. clx(numpts)) then
*           (a real point)
            numpts = numpts + 1
            clx(numpts) = clx(i)
            clmach(numpts) = clmach(i)
          else
* (should be a zero point)
            if (clx(i) .ne. 0.) then
              write (2,*) 'WRTOBL: dropping cl point which is ',
     >                                            i,clx(i),clmach(i)
```

```
            end if
          end if
   10   continue
        write (4,95) ITLE,XBIN,XCIN,SF,frip
        write (4,*) 'z computation begins at first point where M>1'
        write (4,*) ' following line is npts, then x,machno.,z,mu '
        write (4,*) numpts + 2*nt
* first, write the points up to the radial flow region:
        z = 0.0
        do 20 i1 = 1,numpts
          if (clmach(i1) .le. emach) then
            if (clmach(i1) .gt. 1.0) then
              amu = dasin(1.0/clmach(i1))
            else
c             write (*,*) 'mach number on centerline lt 1, skip when'
c             write (*,*) '  computing sidewall mach lines'
c             write (*,*) 'for clmach,i1=',clmach(i1),i1
              amu = 0.0
            end if
            if (i1 .gt. 1) then
              z = z + tan(amu)*(clx(i1)-clx(i1-1))
            end if
            write (4,15) clx(i1),clmach(i1),z,amu
   15       format(4(e14.7,2x))
            i1last = i1
          else
            go to 21
*           *(exit the loop)*
          end if
   20   continue
   21   continue
*  now, write the radial flow region
        if (abs(xbin-(xb*sf+frip)) .gt. tiny) then
          write (*,*) ' xbin,xb= ',xbin,xb
          write (*,*) ' sf,frip= ',sf,frip
          stop 'problem with xbin'
        end if
        xein = xe*sf+frip
c       write (*,*) 'xbin,xein= ',xbin,xein
        deltam = (bmach - emach)/(2*nt)
        gam1 = 2.0/(gam+1)
        gam2 = (gam-1)/(gam+1)
        gam3 = (gam+1)/(2.0*(gam-1))
c       write (*,*) 'gam,1,2,3=',gam,gam1,gam2,gam3
        do 30 i2 = 1,2*nt
          xmach = emach + i2*deltam
* (following implements eqn 29 of sivells report)
          rhs = ((gam1 + gam2*xmach**2)**gam3)/xmach
```

```
          xsorce = rhs**(1.0/(1.0+sigma))
          xin = xsorce*sf + frip
          amu = dasin(1.0/xmach)
          if (i2 .gt. 1) then
            z = z + tan(amu)*(xin-xinold)
            xinold = xin
          else
*           write (4,*) 'z,amu,clx(i1last),xin=',z,amu,clx(i1last),xin
            z = z + tan(amu)*(xin-clx(i1last))
            xinold = xin
          end if
          write (4,15) xin,xmach,z,amu
   30   continue
c       write (*,*) 'sigma=',sigma
c       write (*,*) 'xin,xmach,xsorce,rhs=',xin,xmach,xsorce,rhs
* now, write out the points beyond the radial flow region
        do 40 i3 = i1,numpts
          if (clmach(i3) .gt. bmach) then
            amu = dasin(1.0/clmach(i3))
            z = z + tan(amu)*(clx(i3)-clx(i3-1))
            write (4,15) clx(i3),clmach(i3),z,amu
          end if
   40   continue
        close (unit=4)
*  (done with writing cl mach distribution) *
      end if
*
* following gives coefficients of transonic soln, needed for pressure
      call tcoeff(gam)
*
      if (pp .ne. 0) stop 'coords should be computed rel. to throat'
* (checks to see if coords really computed relative to throat)
*
* now compute the transonic extension upstream:
*   (note that Sivells array below will begin at about the throat!)
*   (following uses only that the throat has a radius of curvature of
*    rc at the throat.  Upstream of the throat, the shape of the entrance
*    is arbitrary, as far as the small perturbation transonic solution in
*    the throat is concerned.)
*   first, determine farthest upstream can reasonably compute:
*     (let second term in contour be small compared to first)
*   (this only works for the nozzle contour, which is arbitrary and determines
*    the interior streamlines)*
      if (icall .eq. 1) then
        write (*,*) 'Using Hopkins-Hill nozzle shape in upstream tr.'
        write (*,*) 'enter entry angle, degrees, gt0: '
        read (*,*) eangle
        write (*,*) 'using nozzle entry angle of ',eangle,' degrees'
```

```
* now compute a series of points down to the throat:
*   6-28-90 skip the point at the throat, end up with two, gives trouble
*  (compute same number of points as used in throat characteristic)
        numpts = abs(lr)
        if (numpts+1 .gt. maxtpt) stop 'too many points in wrtobl array'
        yth = 1.0 !throat radius is 1.0, eangle is degrees
        xend = -1*tiny
* (!small fraction of radii, upstream near throat, not overlap)
* (corrected sign 9-2-94 sps)
* Note that hophill is single precision!! must convert!!
        rap = rc
        thetap=eangle
        gamp = gam
        xendp = xend
        ytp = yth
        call hophill(rap,thetap,gamp,xendp,ytp,numpts,xp,yp,amachp)
* this routine returns numpts values of x,y,amach along streamline
*  upstream from eangle to xend.  Sets xbegin for later calls.
      else
        xthroat = s(1)   !these are the first points in Sivells streamline
        ytp = fs(1)  !must connect to transonic streamline
        if (xthroat .ne. 0.0) then
          write (*,*) 'xthroat= ',xthroat,' first sivells pt'
*         pause 'WRTOBL: x should be 0 at throat'
* first point on streamline MUST be greater than 0, for sonic line
* bows downstream!
        end if
        call hophill(rap,thetap,gamp,xendp,ytp,numpts,xp,yp,amachp)
      end if
      do 60 itx = 1,numpts !get pep0, convert back to double
        amach2 = amachp(itx)**2
        denom = (1. + amach2*(gam-1.)/2.0)**(gam/(gam-1.))
        pep0(itx) = 1.0/denom
   60 continue
*
* now write out the bl data to a special file in easily read form
*
      if (icall .eq. 1) then
        write (3,95) ITLE,XBIN,XCIN,SF,frip
 95     FORMAT (1H1,3A4,' sivells, xbin= ',F11.6,
     >    ' xcin=',F11.6,', sf= ',f11.6,' frip=',f11.6)
        write (3,*) 'next is total pts. and no. upstream of throat,'
        write (3,*) 'then x,y, pe/p0, dy/dx, d2y/dx2 are: '
        write (3,*)
     >  'waltan=dy/dx and sd=d2y/dx2 only written downstream of throat'
* note that these two are given in PERFC, format statement 89, 9-94 sps
        write (3,*) king+numpts, numpts
        write (3,103) (xp(k),yp(k),pep0(k),k=1,numpts)
```

```
        write (3,105) (s(k),fs(k),spr(k),waltan(k),sd(k),k=1,king)
 103    format(3(e14.7,1x))
 105    format(5(e14.7,1x))
      else
        write (3,*) 'next is streamline for icall= ',icall
        write (3,*) 'totpts and no. pts upstream of throat,'
        write (3,*) ' then x,y, pe/p0: '
        write (3,*) 'downstream of throat, also dy/dx and d2y/dx2'
        write (3,*) king+numpts, numpts
        write (3,103) (xp(k),yp(k),pep0(k),k=1,numpts)
        write (3,105) (s(k),fs(k),spr(k),waltan(k),sd(k),k=1,king)
      end if
*
      icall = icall + 1
      return
*
* following all format statements saved for comparison
*
*     write (3,84) RC,ETAD,AMACH,BMACH,CMACH,EMACH,MC,AH                    476
*     write (3,90) (K,S(K),FS(K),WALTAN(K),SD(K),WMN(K),DMDX(K),SPR(K),D    478
*    1PX(K),K=1,KING)                                                       479
C                                                                           493
*  save original format statements from perfc for reference ********************
*84   FORMAT (1H ,4H RC=,F11.6,3X,5HETAD=F8.4,4H DEG,3X,6HAMACH=F10.7,3X    494
*    1,6HBMACH=F10.7,3X,6HCMACH=F10.7,3X,6HEMACH=F10.7,3X,A4,2HH=F11.7/)    495
*89   FORMAT (1H ,9X,5HPOINT,7X,5HX(IN),9X,5HY(IN),9X,5HDY/DX,8X,7HD2Y/D    501
*    1X2,7X,8HMACH NO.,7X,5HDM/DX,9X,5HPE/PO,11X,6HDPR/DX/)                 502
*90   FORMAT (10(10X,I3,2X,0P6F14.7,1P2E16.5/))                            503
*92   FORMAT (1H ,' RC=',F11.7,',    STREAMLINE RATIO=',F11.8,',    TEST    505
*    1 CONE BEGINS AT',F12.7,' IN.' / )                                     506
*95   FORMAT (1H1,3A4,45H INVISCID NOZZLE CONTOUR, RADIAL FLOW ENDS ATF1    509
*    11.6,25H IN., TEST CONE BEGINS ATF11.6,19H IN., SCALE FACTOR=F9.4/)    510
*******************************************************************************
      END                                                                   527
```

C Sivells-to-Harris Interface Code

```
* PROGRAM MAKEBLIN.FOR.
* Steven P. Schneider Purdue University 317-494-3343
* this is a program to read in output from the sivells code,
*   add specifics for Re, and write in a form readable
*   by the Harris code for bl.
* specific for the nozzle block problem sps 6-90
* add some code for the contraction computation 12-5-90 sps
* add code for output of file for arbitrary shape using modified Newtionian thy
*    sps 3-6-91
```

```
* allow for arbitrary power-law distribution of points 3-6-91 sps,
*   and for easy change of jsolve
* add code for reading in pressure data from euler code, and shock location
*   data, and writing an output file for the harris code 7-13-91 sps
* modified to fix bug with error code on different case 7-28-91 sps
* modify a bit to use with whole nozzle, not just sivells part 2-4-92 sps
* modify to make print stations go to te of nozzle, not just near. 4-8-93 sps
* modify 8-94 to make thermal BC explicit, add bullet solution
* modify 9-8-94 to skip 4 lines in sivells for new sivells radcur format, sps
*
      parameter(maxpts=1000,jsolve=1)
* jsolve is the number of solution stations per station of input data
*   can increase to make finer resolution solution possible
      character*80 text
      character*20 root,infile,outfile,parmfl,tawfile
*  most of these are the number of stations along the moc soln
*  ss is the number of stations in bl soln to get
      dimension x(maxpts),y(maxpts),pepo(maxpts),s(maxpts),theta(maxpts)
      dimension proval(maxpts),prntval(maxpts),ss(jsolve*maxpts)
      dimension tw(maxpts)
      dimension xsh(maxpts),ysh(maxpts)  !shock location
      common /param/pstar   !for passing to subroutines for arbitrary shapes
      data pi/3.1415926535/,ksprnt/1/,ksprof/2/
*      *(print info every ksprnt'th soln station; print profile info
*      * (every ksprof'th soln station)
*     need dense printing of soln for gortler work to get good values
*     for derivatives of wall height to get streamwise curvature!
*     need lots of profiles for roughness work also
*
      write (*,*) 'this is the PC version of makeblin'
      write (*,*) 'unix version differs in namelist format'
      write (*,*) 'enter root filename for transfer: '
      read (*,5) root
    5 format(a20)
      ileng = index(root,'  ') - 1
      write (*,*) 'enter 0 if this is a Sivells (or nozzle) test, '
      write (*,*) 'enter 1 if this is a flat plate test, '
      write (*,*) 'enter 2 if this is a Lees modified newtonian test: '
      write (*,*) 'enter 3 if Euler output for body is to be read: '
      write (*,*) 'enter 4 is this is a round cone at zero AOA: '
      read (*,*) imodel
**********************************************************************
      if (imodel .eq. 0) then
        write (*,*) 'Sivells test or other nozzle test'
        infile(1:ileng) = root(1:ileng)
        infile(ileng+1:ileng+3) = '.bl'
        write (*,*) 'reading input data from file ',infile
        open(unit=1,file=infile,status='old')
```

```
        read (1,10) text
   10   format(a80)
        read (1,10) text
        read (1,10) text
        read (1,10) text
        read (1,*) numpts
        if (numpts .gt. maxpts) stop 'too many points'
        do 50 i=1,numpts
          read (1,*) x(i),y(i),pepo(i)
   50   continue
        close (unit=1)
        write (*,*)
     >  ' wall. temp. BC? Enter 1 if isothermal, 2 adiabatic, '
        write (*,*) ' 3 if Tw/Taw = const: '
        read (*,*) itflag
        write (*,*) 'read itflag as: ',itflag
*       (done reading in info from sivells output file)
*****************************************************************
      else if (imodel .eq. 1) then
        write (*,*) 'enter numpts, mach, gam, plend: '
        read (*,*) numpts,amach,gam,plend
        gmexp = gam/(gam-1)
        gmfact = (gam-1)/2.0
        denom = (1.0 + gmfact*amach**2)**gmexp
        pep0 = 1.0/denom
        write (*,*) ' gives pep0= ',pep0
        do 60 i = 1,numpts
          x(i) = plend*float(i-1)/float(numpts-1)
          y(i) = 1.0  !not 0.0, messes up computations
          pepo(i) = pep0
   60   continue
*************************************************************************
      else if (imodel .eq. 2) then
        write (*,*) 'this is a modified newtonian test;'
        write (*,*) 'you must enter the shape in source code'
        write (*,*) 'enter numpts, mach: ' !, pstar, plend: '
        read (*,*) numpts,amach   !,pstar,plend
        gam = 1.4 !air
        if (numpts .gt. maxpts) stop 'too many points'
* now compute pressure ahead of shock, ratio to total pressure
*   in stilling chamber
        gmexp = gam/(gam-1)
        gmfact = (gam-1)/2.0
        denom = (1.0 + gmfact*amach**2)**gmexp
        pinfp0 = 1.0/denom
        write (*,*) ' gives pinfinity/p0= ',pinfp0
* now compute stagnation or total pressure behind normal shock,
*   ratio to p_infty ahead of shock(see Anderson p. 54, 3.17)
```

```
        denom1 = 4*gam*amach**2 - 2*(gam-1)
        piece1 = ( (gam+1)**2 * amach**2/denom1 )**(gmexp)
        piece2 = (1-gam+2*gam*amach**2)/(gam+1)
        ptpinf = piece1*piece2
        ptp0 = ptpinf*pinfp0
        write (*,*) 'pstag on body/p infty is ',ptpinf
        write (*,*) 'gives pstag on body/pstag in stilling=',ptp0
        cpmax = (2.0/(gam*amach**2)) * (ptpinf - 1) !Anderson 3.19
        write (*,*) 'cpmax computed as ',cpmax
* now compute body shape
*    for non-sphere shape, do this in subroutine
c       write (*,*) 'passing to blunts; pstar,plend=',pstar,plend
* blunts commented out 12-93, find subroutine to bring back
c       call blunts(plend,numpts,x,y,theta)
* following does a sphere
        do 70 i = 1,numpts
          x(i) = -1.0 + float(i-1)/float(numpts-1)
          y(i) = sqrt(1.0 - (x(i))**2)
          if (y(i) .gt. 0.0) then
            dydx = -1*x(i)/y(i) !needed for newtonian thy
            theta(i) = atan(dydx)
          else
            theta(i) = pi/2.0
          end if
   70   continue
        write (*,*) 'last x,y are ',x(numpts),',',y(numpts)
        do 80 i = 1,numpts
* now compute ratio of pe to ptotal ahead of shock, using Lees
*     modified newtonian thy - formula derived using p. 54
          pepo(i) = (ptp0 - pinfp0)*(sin(theta(i)))**2 + pinfp0
          if (pepo(i) .ge. 1.0) pepo(i) = 0.99999 !so not singular
 80     continue
*****************************************************************
      else if (imodel .eq. 3) then
        write (*,*) 'working for euler data for blunt body'
        write (*,*) 'enter mach, gamma: '
        read (*,*) amach,gam
* now compute pressure ahead of shock, ratio to total pressure
*   in stilling chamber
        gmexp = gam/(gam-1)
        gmfact = (gam-1)/2.0
        denom = (1.0 + gmfact*amach**2)**gmexp
        pinfp0 = 1.0/denom
        write (*,*) ' gives pinfinity/p0= ',pinfp0
* now compute stagnation or total pressure behind normal shock,
*   ratio to p_infty ahead of shock(see Anderson p. 54, 3.17)
        denom1 = 4*gam*amach**2 - 2*(gam-1)
        piece1 = ( (gam+1)**2 * amach**2/denom1 )**(gmexp)
```

```
        piece2 = (1-gam+2*gam*amach**2)/(gam+1)
        ptpinf = piece1*piece2
        ptp0 = ptpinf*pinfp0
        write (*,*) 'pstag on body/p infty is ',ptpinf
        write (*,*) 'gives pstag on body/pstag in stilling=',ptp0
        write (*,*) 'reading euler data for blunt body: '
        infile(1:ileng) = root(1:ileng)
        infile(ileng+1:ileng+4) = '.eul'
        write (*,*) 'reading input data from file ',infile
        open(unit=1,file=infile,status='old')
        read (1,90) lineskip
   90   format(i6)    !number of lines to skip
        do 92 i = 1,lineskip  !skip 'lineskip' lines of text
          read (1,91) text
   91     format(a80)
   92   continue
        read (1,*) numpts
        if (numpts .gt. maxpts) stop 'too many points'
        do 95 i=1,numpts
          read (1,*) x(i),y(i),pepinf !read euler data for pressure
          pepo(i) = pepinf*pinfp0
   95   continue
*    Now read shock location data:
        read (1,90) lineskip
        do 96 i = 1,lineskip  !skip 'lineskip' lines of text
          read (1,91) text
   96   continue
        read (1,*) numshpts
        if (numshpts .gt. maxpts) stop 'too many points'
        do 97 i=1,numshpts
          read (1,*) xsh(i),ysh(i) !read euler data for shock location
   97   continue
        write (*,*) 'done reading from file'
*******************************************************
      else if (imodel .eq. 4) then
        write (*,*) ' cone: enter numpts, gam, axial length: '
        read (*,*) numpts,gam,axleng
        write (*,*) ' enter half-angle (deg.), shock angle, pep0: '
        read (*,*) anghalf, wave, pep0
        anghalf = anghalf*pi/180.0
        do 100 i = 1,numpts
          x(i) = axleng*float(i-1)/float(numpts-1)
          y(i) = tan(anghalf)*x(i)  !not 0.0, messes up computations
          pepo(i) = pep0
  100   continue
      else
          stop 'imodel must be 0,1,2,3, or 4'
      end if
```

```
*******************************************************
* now check computed values
      do 150 i = 1,numpts
        if (imodel .eq. 2 .or. imodel .eq. 3) then
          if (pepo(i)-ptp0 .gt. -0.01*ptp0) then !arbitrary nearness
            write (*,*) 'pepo(',i,')= ',pepo(i)
            write (*,*) 'ptp0= ',ptp0
            write (*,*) 'if first gt second when computed by VGBLP,'
            write (*,*) 'this will give surface static pressure larger'
            write (*,*) ' than the total pressure on the surface'
            write (*,*) 'this would be a fatal error in VGBLP'
            if (pepo(i) .gt. ptp0) then
              write (*,*) 'reducing pepo(',i,') to 0.99999*ptp0'
              write (*,*) ' to forstall error in VGBLP'
              pepo(i) = 0.99999*ptp0
            end if
          end if
        end if
  150 continue
*
*  now read in parametric info from parameter file
*
      parmfl(1:ileng) = root(1:ileng)
      parmfl(ileng+1:ileng+3) = '.re'
      write (*,*) 'reading reynolds number scaling info from ',parmfl
      open(unit=1,file=parmfl,status='old')
      read (1,5) outfile    !read filename to write to
      read (1,*) throat      !throat radius in feet
* (assumes input scaled so throat radius is one unit) *
* (for more general shapes, just treats 'throat' as scaling parameter
*   for lengths)
      read (1,*) prandtl
* ptotal,ttotal,xmach are conditions at infinity - see p. 44
*   for nozzle are stagnation chamber conditions
* for non-nozzle, xmach seems to affect mostly the computations
*   involving the flow behind the shock.  Should be freestream values!!
      read (1,*) ptotal
      read (1,*) rgas
      read (1,*) ttotal   !ahead of le shock
      read (1,*) xmachi
      if (imodel .ne. 0) then
        if (xmachi .lt. 1.0) then
          write (*,*) 'xmach given as ',xmachi
          write (*,*) 'should be freestream value ahead of shock,'
          write (*,*) ' not the value at stagnation!!'
        end if
      end if
      close (unit=1)
```

```
*
* now set defaults for input to harris program (besides harris's)
*
      if (imodel .eq. 2 .or. imodel .eq. 3) then
        ibody = 1    !stag pt at nose
        j = 1        !axisymmetric
      else if (imodel .eq. 1) then
        ibody = 2    !no stagnation point at nose
        j = 0        !2D
        write (*,*) 'assuming 2D geometry'
      else if (imodel .eq. 0) then
        write (*,*) 'enter 0 if 2D nozzle, 1 if axisymmetric: '
        read (*,*) j
        if (j .ne. 0 .and. j .ne. 1) stop 'j must be 0 or 1'
      else if (imodel .eq. 4) then
        ibody = 2 !sharp cone, no stag. pt.
        j = 1
      end if
      ie = 51     !from test case number 4
      if (imodel .eq. 3) then
        ientro = 2  !variable entropy calculation
      else
        ientro = 1
      end if
      igeom = 1    !create coords using geometric series; need xk,ie,xend!!
      kodunit=0    !US units
      if (itflag .eq. 1 .or. imodel .eq. 4) then
        kodwal=1     !specify wall temperature distribution (no time to heat)
        write (*,*) ' setting kodwal=1, isothermal wall!!'
        write (*,*) 'setting wall temp equal to total temp, beware!'
* model temp same as total temp, ambient
* note that total temperature behind shock is the same as ahead of shock
*                *(not quite stagnation point temp. but close)*
        twall = ttotal
      else if (imodel .eq. 2) then
        kodwal=1
        write (*,*) 'enter isothermal wall temperature, rankine: '
        read (*,*) twall
        write (*,*) 'using wall temp= ',twall
      else if (itflag .eq. 2 .or. imodel .eq. 3) then
        kodwal=2     !specify adiabatic wall
        write (*,*) ' setting kodwal=2, adiabatic wall!!'
      else if (itflag .eq. 3) then
        kodwal = 1
        write (*,*) 'setting kodwal=1, isothermal wall'
        write (*,*) 'enter file to read, .prt file with Taw/TT1 data: '
        read (*,5) tawfile
        write (*,*) 'reading Taw/TT1 from ',tawfile
```

```
        write (*,*) 'enter const, where Tw/Taw = const: '
        read (*,*) tratio
        write (*,*) 'using tratio= ',tratio
        open (unit=11,file=tawfile,status='old')
        read (11,10) text
        read (11,10) text
        read (11,*) numpts1,prandtl1,j1,omega1,rref1,uref1
        if (numpts .ne. numpts1) then
          numpts = numpts1 !reset and solve at prev. solution stations!
          iend1 = numpts
          throat = 1.0  !don't rescale dimensions!!
        else
          iend1 = jsolve*numpts
        end if
        do 170 i = 1,numpts
          read (11,*) z1,rmi1,s1,ye1,dltast1,theta1,res1,pe1,te1,ue1,
     >       twott1,amache1,amue1,xi1,qsd1,hd1
          tw(i) = ttotal*twott1*tratio
          x(i) = z1
          y(i) = rmi1
          pepo(i) = pe1/ptotal
  170   continue
        close (unit=11)
      else
        stop 'logic error in setting wall thermal BC'
      end if
      if (itflag .ne. 3) then  !otherwise set above when read file
        iend1 = jsolve*numpts  !number of soln stations
      end if
      proinc = 10.0 ! hopefully, none
      prntinc = 10.0
      sst = 1e+20  !no transition on body until then (laminar flow)
      if (imodel .gt. 1 .and. imodel .lt. 4) then
        wave = 90.0  !shock wave angle at s=0, needed for test case 4 type flows
      else if (imodel .eq. 4) then
        write (*,*) 'read shock wave angle at origin as: ',wave
      else
        wave = 0.0  !needed for shockless type flows
      end if
c     xend = 120   !from blasius test case
      xend = 10    !as in test case 4
c     xk = 1.275   !value used in test cases in book, sets grid
c     xk = 1.1     !because value used in text gives hyper-dense
*         grid near wall, which makes for difficulties.
c     xk = 1.0     !like test case number 4
c     xk = 1.05    !because 1.0 gives little near wall for sphere
      xk = 1.1     !because 1.05 gives not great resolution for stability
*
```

```
* compute the arc length along the wall (see (66) of harris paper)
*   (approximate with straight line segments between stations)
      s(1) = 0.0
      if (itflag .le. 2) then
        tw(1) = twall
      end if
      do 180 i = 2,numpts
        s(i) = s(i-1) + sqrt( (x(i)-x(i-1))**2 + (y(i)-y(i-1))**2 )
        if (itflag .le. 2) tw(i) = twall !set above, assumes same along wall
  180 continue
* compute the s stations to get soln at, and to write at, even spacing
*    in sqrt(s) normally, other power for other cases
      if (imodel .lt. 2) then
        power = 2.0
        write (*,*) 'using square root distribution of pts'
      else
* with sphere, have problems with stepsize increasing too rapidly near le
        power = 1.0
        write (*,*) 'using linear distribution of pts, good for blunt'
*       *(try this, since problems with T<0 at le)*like test case 4
      end if
      root1 = 1.0/power
      rsinc = (s(numpts)-s(1))**root1/float(iend1)  !try-
      ss(1) = rsinc**power
      ss(2) = ss(1)  !required
      ss(3) = ss(1)  !also required, actually
      srun = 3*ss(1) !running value of s
      iprnt = 0
      ipro = 0
      do 200 i = 4,iend1
        srunold = srun
        srun = ( (srunold)**root1 + rsinc )**power
        ss(i) = srun-srunold
* add last part to following to have a print station at the end:
        if (mod(i,ksprnt) .eq. 0 .or. i .eq. 1 .or. i .eq. iend1) then
          iprnt = iprnt + 1
          if (iprnt .gt. maxpts) then
            write (*,*) 'iprnt = ',iprnt,' exceeded maxpts'
            stop 'fatal error'
          end if
          prntval(iprnt) = srun
        end if
        if (mod(i,ksprof) .eq. 0) then
          ipro = ipro + 1
          proval(ipro) = srun
        end if
  200 continue
*
```

```
* Now write out NAM1 namelist into file
*
      write (*,*) ' writing bl input to file ',outfile
      open(unit=2,file=outfile)
      write (2,*) '&NAM1'
      write (2,*) 'IBODY=',ibody
      write (2,*) 'IE=',ie
      write (2,*) 'IEND1=',iend1
      write (*,*) 'may need to reset param.inc, note that '
      write (*,*) 'ie is now ',ie,' and iend1= ',iend1
      write (2,*) 'IENTRO=',ientro
      write (2,*) 'IGEOM=',igeom
      write (2,*) 'IPRO=',ipro
      write (2,*) 'IPRNT=',iprnt
      write (*,*) 'ipro and iprnt are ',ipro,iprnt
      write (2,*) 'J=',j
      write (2,*) 'KODUNIT=',kodunit
      write (2,*) 'KODWAL=',kodwal
      if (imodel .ne. 2) then
        phii = (180.0/pi)*atan( (y(2)-y(1))/(x(2)-x(1)) )
      else
        write (*,*) 'setting leading edge angle = 90 degrees'
        phii = 90.0
      end if
      write (2,*) 'PHII=',phii
      write (2,*) 'PR=',prandtl
      write (2,*) 'PRNTINC=',prntinc
      write (2,*) 'PRNTVAL=',(prntval(i)*throat,i=1,iprnt)
      write (2,*) 'PROINC=',proinc
      write (2,*) 'PROVAL=',(proval(i)*throat,i=1,ipro)
      write (2,*) 'PT1=',ptotal
      write (2,*) 'R=',rgas
      write (2,*) 'SST=',sst
      write (2,*) 'TT1=',ttotal
      write (2,*) 'WAVE=',wave
      write (2,*) 'XEND=',xend
      write (2,*) 'XK=',xk
      write (2,*) 'XMA=',xmachi
      write (2,*) '/'    !end of namelist input
* end of writing nam1.  Now compute and write nam2:
      write (2,*) '&NAM2'
      write (2,*) 'NUMBER=',numpts
      write (*,*) 'and NUMBER is ',numpts
      write (2,*) 'PE=',(pepo(i)*ptotal,i=1,numpts)
      write (2,*) 'RMI=',(y(i)*throat,i=1,numpts)
      write (2,*) 'S=',(s(i)*throat,i=1,numpts)
      write (2,*) 'SS=',(ss(i)*throat,i=1,iend1)
      if (kodwal .eq. 1) then  !specified wall temp
```

```
        write (2,*) 'TW=',(tw(i),i=1,numpts)
      else !specified heat transfer
        write (2,500) numpts-1     !make adiabatic wall
  500   format('QW=',i4,'*0.0,0.0') !,0.0 kluge for compiler bug
      end if
      write (2,*) 'Z=',(x(i)*throat,i=1,numpts)
      write (2,*) '/'
* end of writing nam2.  Now write nam3, if required
      if (imodel .eq. 3) then   !doing entropy computations through shock
        write (2,*) '&NAM3'
        write (2,*) 'NUMBER=', numshpts
        write (*,*) 'now numshpts (JL) is ',numshpts
        write (2,*) 'RRS= ',(xsh(i)*throat,i=1,numshpts)
        write (2,*) 'ZZS= ',(ysh(i)*throat,i=1,numshpts)
      end if
      write (2,*) '/'
      close (unit=2)
*
      stop 'end of makeblin'
      end
```

D Harris to e**MALIK Translation Code

```
* BLTOSTAB.FOR
* this is a program to take vgblp data and put in E**MALIK form
* sps 7-2-90
* revised 7-9-90 to fit with input form provided by Maliks vgblp program,
*   different from that implied in preliminary paper
* revised 7-17-90 to fix problem with getting correct matches
*    also revised to give correct scaling of profiles for emalik code
* revised 7-18-90 to scale before passing to utder, makes cutoffs clearer
* revised 7-16-91 to write more info to .cur file sps
* revised 7-16 and 7-17-91 to get better derivatives of surface shape sps
*   and also to skip past unused iterations printed using variable entropy
*   computations
* revised 8-27-91 to use derivative data now output by Harris code
*   harris code outputs FZ and TZ, which is actually the derivs wrt the
*   wall-normal coordinate eta at the next streamwise solution station.
*   However, we do not normally output every single solution station, so
*   cannot just use this at the next solution station output.  So accept
*   the error involved in using the derivs at next solution station in
*   place of derivs at current, for now.
* mod. 5-27-94 to read and compute derivs in double, needed for t profiles
* mod. 9-2-94 sps to read derivs of nozzle contour from .bl sivells file
*  (couldnt get good derivatives of nozzle contour from printed data)
* mod 9-9-94 to interpolate sivells data, stations don't match, sps
```

```
* mod. 10-14-94 to read curvature data from file when not using sivells, sps
* mod 10-24-94 to use Frank Chen's method of getting curvature from nozzle
*    contour data, using old NOS routine, sps.
*
c     implicit double precision (a-h,o-z) !needed for derivs
      parameter (mpt=1000,npt=1000)
      character*81 text  !try changing to 81 from 80 per 7-91 version
      character*20 blfile,outfil,root,profil,curfil,sivfil,refil,gortfl
      character*20 sivck,unsivck
      dimension z(mpt),rmi(mpt),s(mpt),ye(mpt),dltast(mpt),theta(mpt),
     >    res(mpt),pe(mpt),te(mpt),ue(mpt),twtt1(mpt),ame(mpt),
     >    amue(mpt),xi(mpt)
      dimension xsiv(mpt),ysiv(mpt),ssiv(mpt),dydx(mpt),
     >   d2ydx2(mpt),radcur(mpt)
      dimension df(mpt),coef(mpt,4),wk(7*mpt+9),rrmi(mpt)  !for CSDS subroutine
      dimension x(npt),u(npt),u1(npt),u2(npt),t(npt),t1(npt),t2(npt)
      dimension u1deb(npt),etascal(npt),t23b(npt)
      data small/1.0e-4/
      data norder/3/ !order of polynomial interpolation
      data maxiter/5/  !maximum number of variable entropy iterations in file
      data numiter/0/  !number of variable entropy iter written to file
      data nhophill/0/ !used as an offset
      data eps/0.1/ !fractional error acceptable in derivs, this is
*           really a check on the unit conversions
      data huge/1.0e10/ !if radcur infinite
      data df/mpt*1.e-3/ !estimate of standard dev. for CSDS
*
      write (*,*) 'enter root filename for writing and reading: '
      read (*,5) root
    5 format(a20)
      il = index(root,'  ') -1
      curfil(1:il) = root(1:il)
      curfil(il+1:il+4) = '.cur'
      write (*,*) 'opening ',curfil,' for curvature output'
      open(unit=3,file=curfil)
      write (3,*) 'curvature data for this case'
      write (3,*) 'first set: nz,z,s,rmi,radcur,epsxr'
      gortfl(1:il) = root(1:il)
      gortfl(il+1:il+4) = '.gor'
      write (*,*) 'opening ',gortfl,' for gortler number output'
      open(unit=7,file=gortfl)
      write (7,*) 'z,s,theta,res,radcur,gortno for file: ',root
      sivck(1:il) = root(1:il)
      sivck(il+1:il+4) = '.sck'
      unsivck(1:il) = root(1:il)
      unsivck(il+1:il+4) = '.usk'
      blfile(1:il) = root(1:il)
      blfile(il+1:il+4) = '.prt'
```

```
      write (*,*) ' opening ',blfile,' for read printed info'
      open(unit=1,file=blfile,status='old')
      read (1,10) text
   10 format(a80)
      read (1,10) text    !second line of text
* for files with variable entropy computations, several iterations
*   may exist in the file, so the print data is redone for several
*   iterations, following sequential in file.  Skip to last
*  read iprnt, prandtl, j which stays the same
      read (1,*) iprnt,prandtl,jgeom,omega,rref,uref
      if (iprnt .gt. mpt) stop 'too many print stations'
*  now read the first set of iprnt values:
      do 110 iiter = 1,maxiter
        do 100 i = 1,iprnt
          read (1,*) z(i),rmi(i),s(i),ye(i),dltast(i),theta(i),res(i),
     >       pe(i),te(i),ue(i),twtt1(i),ame(i),amue(i),xi(i)
          if (z(i) .lt. 0.0) nupstrm = i !number of pts upstream of throat
  100   continue
        read (1,*,end=120) iprnt2,prandtl2,jgeom2,omega2  !now repeated
        read (1,*,end=120) z(1),rmi(1),s(1),
     >     ye(1),dltast(1),theta(1),res(1),
     >     pe(1),te(1),ue(1),twtt1(1),ame(1),amue(1),xi(1)
        if (s(1) .lt. s(iprnt)) then   !there IS another iteration
          numiter = numiter + 1
          write (*,*) 'read iteration ',numiter,' read next'
          backspace(1)  !back up to before the first line in this iter.
        else
          write (*,*) 'iprnt = ',iprnt
          write (*,*) 's(1)= ',s(1),' s(iprnt)= ',s(iprnt)
          stop 'fatal error, funny business in reading print file'
        end if
  110 continue
      stop 'reached maxiter reading variable entropy data'
  120 continue   !reached eof looking for next iteration, done-
      close (unit=1)
      write (*,*) 'there are ',numiter+1,'iteration sets in file'
* now open profile info file:
      profil(1:il) = root(1:il)
      profil(il+1:il+4) = '.pro'
      write (*,*) 'opening ',profil,' for reading from vgblp'
      open(unit=1,file=profil,status='old')
      read (1,150) text
  150 format(a80)
      read (1,*) ipro
*
      outfil(1:il) = root(1:il)
      outfil(il+1:il+4) = '.bfl'
      write (*,*) 'opening ',outfil,' for writing to e**malik'
```

```
      open(unit=2,file=outfil,form='unformatted')
      write (2) text
      write (2) ipro
*
* Decide if getting curvature data from sivells or elsewhere:
      write (*,*) 'where to get curvature data for gortler? '
      write (*,*) 'enter 1 if sivells data, 2 if by spline diff.: '
      read (*,*) iwherec
*
* beginning of block where use 1 of 2 ways to get curvature data
*
      if (iwherec .eq. 1) then
* now open the .re file to get scaling information for the sivells file
        refil(1:il) = root(1:il)
        refil(il+1:il+3) = '.re'
        write (*,*) 'opening ',refil,
     >    ' to get reynolds scaling for sivells'
        open (unit=4,file=refil,status='old')
        read (4,150) text  !skip first line
        read (4,*) throatrad  !in feet, scales sivells data
        close (unit=4)
*
* now open the sivells output file directly, the XXXX.bl file used
* as input to the harris code.  Pick up the derivatives of the nozzle
*  contour from here.  9-94 sps
*
        sivfil(1:il) = root(1:il)
        sivfil(il+1:il+3) = '.bl'
        write (*,*) 'opening ',sivfil,' to read sivells contour derivs'
        open (unit=4,file=sivfil,status='old')
        read (4,150) text
        read (4,150) text
        read (4,150) text
        read (4,150) text !skip header lines
        read (4,*) ntotal,nhophill  !total num pts, no. of hopkins-hill
        do 160 i = 1,nhophill
          read (4,*) xdum,ydum,pratdum  !skip past these points
          if (i .eq. 1) then
            sarcl = 0.0
            xold = xdum
            yold = ydum
          else
            sarcl = sarcl + sqrt((xdum-xold)**2 + (ydum-yold)**2)
            xold = xdum
            yold = ydum
          end if
  160   continue
        nsiv = ntotal-nhophill
```

```
        do 200 i = 1,nsiv
          read (4,*) xsiv(i),ysiv(i),pratdum,dydx(i),d2ydx2(i)
          xsiv(i) = xsiv(i)*throatrad
          ysiv(i) = ysiv(i)*throatrad
          if (i .eq. 1) then
            ssiv(i) = throatrad*sarcl
          else
            ssiv(i) = ssiv(i-1) +
     >        sqrt((xsiv(i)-xsiv(i-1))**2 + (ysiv(i)-ysiv(i-1))**2)
          end if
          d2ydx2(i) = d2ydx2(i)/throatrad  !change to feet from throat radii
          if (d2ydx2(i) .ne. 0.0) then
            radcur(i) = (1.0+dydx(i)**2)**1.5/d2ydx2(i)
          else
            radcur(i) = huge
          end if
*   !concave is minus for malik!
  200   continue
        close (unit=4)
* (note that must be checked that these are at same stations)
        write (*,*) 'opening ',sivck,' for sivells curvature check'
        open (unit=4,file=sivck)
        write (4,*) 'sivells curvature check: i,xsiv,ysiv,ssiv,radcur'
        do 210 i = 1,nsiv
          write (4,209) i,xsiv(i),ysiv(i),ssiv(i),radcur(i)
  209     format(i4,3(1x,f12.5),1x,1p,e12.5)
  210   continue
        close (unit=4)
        ngor = iprnt - nupstrm !number of pts in .prt file downstream of throat
*
      else if (iwherec .eq. 2) then !get derivs using contour directly
*
        write (*,*) 'opening ',unsivck,' for unsivells curv. check'
        open (unit=4,file=unsivck)
        write (4,*) 'non-sivells curv. ck: z,rrmi,dydx,d2ydx2,radcur'
c From Frank_Chen.AERONAUTICS@qmgate.larc.nasa.gov Mon Oct 17 13:53 EST 1994
c RE>gortler test case. The fragment of the code I used is very simple.
* following uses NOS routine CSDS, see header for this subroutine.
        IPT1=-1
        fnmin = iprnt - (2.0*iprnt)**0.5
        fnmax = iprnt + (2.0*iprnt)**0.5
        fn = (fnmin+fnmax)*0.5 !a guess for what to use
        CALL CSDS(mpt,iprnt,Z,RMI,DF,fn,IPT1,COEF,WK,IERR)
        if (ierr .ne. 0) then
          write (*,*) 'error return from CSDS, ierr= ',ierr
          stop 'halting'
        end if
        rrmi(1) = coef(i,1)  !dh = 0 for these three, the first point
```

```
        if (rrmi(1) .eq. 0.0) then
          write (*,*) 'problem with csds at first pt., rrmi(1)=0'
          write (*,*) 'set to rmi(1)'
          rrmi(1) = rmi(1)
        end if
        dydx(1) = coef(i,2)
        d2ydx2(1) = 2.0*coef(i,3)
        DO 217 I=1,iprnt-1
          DH=Z(I+1)-z(i)
*         * note that rrmi is the interpolated location of rmi from spline fit*
          RRMI(I+1)=((COEF(I,4)*DH+COEF(I,3))*DH+COEF(I,2))*DH+COEF(I,1)
          dydx(I+1)=(3.0*COEF(I,4)*DH+2.0*COEF(I,3))*DH+COEF(I,2)
          d2ydx2(I+1)=6.0*COEF(I,4)*DH+2.0*COEF(I,3)
  217   CONTINUE
* end of frank chen fragment (which has been adapted here)
        do 220 i = 1,iprnt
          if (d2ydx2(i) .ne. 0.0) then
            radcur(i) = (1.0+dydx(i)**2)**1.5/d2ydx2(i)
          else
            radcur(i) = huge
          end if
          write (4,219) z(i),rrmi(i),dydx(i),d2ydx2(i),radcur(i)
  219     format(5(1x,1p,e14.7))
*   !concave is minus for malik!
  220   continue
        close (unit=4)
        ngor = iprnt  !for gortler printout
      else
        stop 'invalid iwherec'
      end if
*
* end of block where get radcur in one of two ways
*
*  now write gortler number output for checking
      do 225 i = 1,ngor
        if (iwherec .eq. 2) then  !s array and radcur array indexed same
          retheta = res(i)*theta(i)/s(i)
          radcur1 = radcur(i)
          i2 = i
        else !using sivells output
          i2 = i + nupstrm
          s1 = s(i2)     !i indexes over .prt array, NOT sivells array
          call locate(ssiv,nsiv,s1,jsiv)
          if (jsiv .ge. norder) then
            call polint(ssiv(jsiv-norder+1),radcur(jsiv-norder+1),
     >        norder,s1,radcur1,errest)
          else
            call polint(ssiv(1),radcur(1),norder,
```

```
     >        s1,radcur1,errest)
           end if
           if (s(i2) .eq. 0.0) then
             write (*,*) 'i2,s= ',i2,s(i2),' s(i2-1)= ',s(i2-1)
             stop 'fatal'
           end if
           retheta = res(i2)*theta(i2)/s(i2)
         end if
         if (radcur1 .lt. 0.0) then
           gortno = retheta * sqrt(theta(i2)/abs(radcur1))
         else if (radcur1 .gt. 0.0) then
           gortno = 0.0  !convex
         else
           gortno = huge
         end if
         write (7,223) z(i2),s(i2),theta(i2),res(i2),radcur1,gortno
  223   format(6(1x,1p,e12.5))
  225 continue
      write (7,*)
     > 'now z(in),s(ft),radcur,gortl,gortno from emalik algorithm: '
*
*   Now have everything need from prntval stations.  Start reading
*     data from proval stations and writing to malik program
*
*   Before read the profiles, skip to the last set of profiles
*    (multiple sets if doing variable entropy computations)
*   Know how many iterations in file from prt file, use this info here
      do 250 iskip = 1,numiter    !numiter is num in file -1
        do 240 istation = 1,ipro  !ipro stations
          read (1,*) nnp,s1
          do 230 j = 1,nnp
            read (1,*) xdum,udum,tdum,u1ndum,t1ndum
  230     continue
  240   continue
  250 continue
*
      open(unit=4,file='bltostab.deb')
*
      do 1000 nz = 1,ipro    !loop over stations
*       first, get general info for station from iprnt file
        read (1,*) nnp,s1  !number of points in profile- see malik documents
        if (nnp .gt. npt) stop 'too many points in profile'
        write (*,*) 'working profile station ',nz,' with ',nnp,'pts'
        do 300 i = 1,iprnt  !now find matching prnt station:
          if (abs(s1-s(i))/s1 .lt. small) then
            jprnt = i
            go to 301
          end if
```

```
  300   continue
        write (*,*) 's1= ',s1
        stop 'no match found'
  301   continue
        write (*,*) 'found match at print station ',jprnt
        write (*,*) 's1= ',s1,' s(jprnt)= ',s(jprnt)
        res1 = res(jprnt)
        rey = sqrt(res1)
        dstz = s1/rey
*
* following uses radii of curvature
*
* following block interpolates contour derivatives for radius of curvature
* from original sivells data file, using derivs output by sivells
        if (z(jprnt) .gt. 0.0 .and. iwherec .eq. 1) then
* aredownstream of throat, so do radcurvature
* note that concave curvature should have a minus sign for Malik!!
* following Numerical Recipes routine finds position of pt in array
*   ssiv that is just below s1, returns in jsiv
          call locate(ssiv,nsiv,s1,jsiv)
* following routine performs norder-pt polynomial interpolation
          call polint(ssiv(jsiv-norder+1),radcur(jsiv-norder+1),norder,
     >      s1,radcur1,errest)
          write (*,*) 'radcurtab= ',radcur(jsiv-norder+1),
     >      ' radcurinterpolated= ',radcur1
          if (radcur1 .lt. huge .and. abs(errest)/radcur1 .gt. eps) then
            write (*,*) 'radcur interpol. err est= ',errest
            write (*,*) ' when radcur= ',radcur1
            pause ' too large? '
          end if
          epsxr = dstz/radcur1
          call polint(ssiv(jsiv-norder+1),dydx(jsiv-norder+1),norder,
     >      s1,dydx1,errest)
          if (abs(errest)/dydx1 .gt. eps) then
            write (*,*) 'dydx interpol. err est= ',errest
            write (*,*) ' when dydx= ',dydx1
            pause ' too large? '
          end if
          drdx = dydx1
          rmi1 = rmi(jprnt)
        else if (iwherec .eq. 1) then !upstream of throat in hopkins-hill region
          epsxr = 0.0  !neglect gortler upstream of throat
          drdx = (rmi(jprnt+1)-rmi(jprnt))/(z(jprnt+1)-z(jprnt))
          rmi1 = rmi(jprnt)
          radcur1 = 0.0 !flag
        else  !iwherec .eq. 2, not sivells, use original data
          radcur1 = radcur(jprnt)  !local value
          epsxr = dstz/radcur1
```

```
          drdx = dydx(jprnt)
          rmi1 = rrmi(jprnt)  !use value interpolated from spline fit
        end if
        write (3,350) nz,z(jprnt),s1,rmi(jprnt),radcur1,epsxr
  350   format(1x,i4,5(1x,1p,e13.6))
c       rmi1 = rmi(jprnt) !use spline fit or not, depends, moved up
        theta1 = theta(jprnt)
        del995 = ye(jprnt)
        retheta = res(jprnt)*theta1/s1
        ame1 = ame(jprnt)
        if (ame1 .gt. 0.) then
          rethm = retheta/ame1
        else
          write (*,*) 'ame1= ',ame1,' nz= ',nz
          stop 'fatal error'
        end if
        te1 = te(jprnt)
        amue1 = amue(jprnt)
        ue1 = ue(jprnt)
        xc = s1
        pe1 = pe(jprnt)
        kodunit = 0  !british units
        igas = 0  !perfect
        displc = dltast(jprnt) !displacement thickness
*  now, write general info to file
        write (2) nz,nnp,dstz,rey,res1,epsxr,drdx,rmi1,theta1,del995,
     >            retheta,rethm,prandtl,kodunit,igas
        write (2) te1,ame1,ue1,xc
* now test gortler number computations vs. emalik style
        if (epsxr .lt. 0.0) then
          gortl = rey*sqrt(abs(epsxr))
          gortth = gortl*(theta1/dstz)**1.5
        else
          gortl = 0.0
          gortth = 0.0
        end if
        write (7,219) 12.0*z(jprnt),s1,radcur1,gortl,gortth
*  now, read the profile info:
*   and at the same time normalize
        xscal = del995*sqrt(res1)/s1   !see maliks version of the harris code
        escal1 = (res1*amue1)/(rref*uref*s1*sqrt(2.0*xi(jprnt))) *
     >                  rmi1**jgeom  !rref and uref added 8-30-91
        phi = atan(drdx)  !changed 9-8-94 sps
* yescal changes d/dytilde derivs to d/dy/ye derivs, see (15)
        yescal = del995/omega
        if (nz .eq. ipro .or. nz .eq. 1) then
          write (4,*) 'debug data for station nz= ',nz
          write (4,*) 'res1,amue1,s1,xi=',res1,amue1,s1,xi(jprnt)
```

```
          write (4,*) 'del995,xscal,escal1= ',del995,xscal,escal1
          write (4,*) 'phi,yescal = ',phi,yescal
          write (4,*) 'rref,uref= ',rref,uref
        end if
        if (rmi1 .le. 0.0) then
          write (*,*) 'bltostab debug: rmi1= ',rmi1
          write (*,*) ' at nz= ',nz,' s1= ',s1,' z(jprnt)= ',z(jprnt)
          write (*,*) ' rmi(jprnt)= ',rmi(jprnt),
     >        ' rrmi(jprnt)= ',rrmi(jprnt)
        end if
        do 400 j = 1,nnp
          read (1,*) x(j),u(j),t(j),u1(j),t1(j) !really FZ and TZ, accept error
*  following rescales eta derivs to y/ye derivs for malik code
*     this next pair is from 23b in harris manual
          t23b(j) = 1.0 + x(j)*ye(jprnt)*cos(phi)/rmi1
*  this next line derived from eqn 24b in manual, changes eta derivs to
*            y/ye derivs
          etascal(j) = escal1 * t23b(j)**jgeom / t(j)
* the xscal factors in the following are to convert to malik code form
          x(j) = x(j)*xscal    !scaling for malik code
          u1(j) = yescal*etascal(j)*u1(j)/xscal
          t1(j) = yescal*etascal(j)*t1(j)/xscal
  400   continue
        write (*,*) 'getting derivatives'
        call scond(x,u1,u2,nnp)  !get second derivs from first
        call scond(x,t1,t2,nnp)
*         *change to use of utder as malik, adapted from maliks
*         call utder(nnp,x,u,t,u1,u2,t1,t2)
*         don't have derivs from harris for first point
* now, write the profile info
        write (2) (x(j),j=1,nnp)
        write (2) (u(j),j=1,nnp)
        write (2) (u1(j),j=1,nnp)
        write (2) (u2(j),j=1,nnp)
        write (2) (t(j),j=1,nnp)
        write (2) (t1(j),j=1,nnp)
        write (2) (t2(j),j=1,nnp)
* for checking
        u1test = (u(2)-u(1))/(x(2)-x(1))
        if (abs((u1test-u1(1))/u1test) .gt. eps) then
          write (*,*) 'nz= ',nz,' u1test,u1(1)= ',u1test,u1(1)
          write (*,*) 'problems with generation of u derivatives'
          pause 'looks like fatal error'
        end if
* change test specs due to profiles being so flat, adiabatic wall effects
        nnptest = nnp/2.0
        t1test = (t(nnptest)-t(nnptest-1))/(x(nnptest)-x(nnptest-1))
        if (t1test .ne. 0.0) then
```

```
          if ((abs((t1test-t1(nnptest))/t1test) .gt. 30*eps)
     >        .and. (abs(t1test-t1(nnptest)) .gt. small)) then
            write (*,*) 'nz= ',nz,' t1test,t1(nnptest)= ',
     >                    t1test,t1(nnptest)
            write (*,*) 'nnptest = ',nnptest
            write (*,*) 'problems with generation of t derivatives'
            pause 'looks like fatal error'
          end if
        end if
*  for debug:
        if (nz .eq. ipro .or. nz .eq. 1) then
          call scond(x,u,u1deb,nnp)
          write (4,*) 'nz= ',nz,' , debug info'
          write (4,*) 'del995 (ye)= ',del995
          write (4,*)
     >       'x,u,u1,u1deb,uratio,t23b,etascal,u2,t,t1,t2= ',
     >       '(as written to bfl file)'
          do 900 i = 1,nnp
            if (u1deb(i) .ne. 0.0) then
              uratio = u1(i)/u1deb(i)
            else
              uratio = 0.0  !arbitrary
            end if
            write(4,850) x(i),u(i),u1(i),u1deb(i),uratio,t23b(i),
     >          etascal(i),u2(i),t(i),t1(i),t2(i)
  850       format(11(1x,e18.12))
  900     continue
        end if
 1000 continue
      close(unit=4)
      close(unit=3)
*
      stop
      end
* --------------------------------
* this is a program taken from sivells to compute derivatives
*  modified 7-13-90 to deal with errors in endpoint
      SUBROUTINE SCOND (A,B,C,KING)
C     TO OBTAIN PARABOLIC DERIVATIVE OF CURVE (UNEQUALLY SPACED POINTS)
*     IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION A(*), B(*), C(*)
      data eps/0.01/
      N=KING-1
      DO 1 K=2,N
c     write (*,*) 'a(',k,')=',a(k)
      S=A(K)-A(K-1)
      T=A(K+1)-A(K)
1     C(K)=((B(K+1)-B(K))*S*S+(B(K)-B(K-1))*T*T)/(S*S*T+S*T*T)
```

```
      SO=A(2)-A(1)
      if (so .eq. 0.) then
        write (*,*) 'a(1,2=',a(1),a(2)
        stop ' SCOND: so=0'
      end if
      TO=A(3)-A(2)
      if (to .eq. 0.) stop ' SCOND: to=0'
      QO=SO+TO
      C(1)=(-TO*(QO+SO)*B(1)+QO*QO*B(2)-SO*SO*B(3))/QO/SO/TO
* following added when got bad values near wall sps 7-90
      clinear = (b(2)-b(1))/so
      if (clinear .ne. 0.0) then
        error = abs(c(1)-clinear)/clinear
      else
        if (c(1) .ne. 0.0) error = 1.0
      end if
      if (error .gt. eps) then
        write (*,*) 'SCOND: problems with c(1)'
        write (*,*) 'c(1),clinear= ',c(1),clinear
        write (*,*) 'using clinear'
        c(1) = clinear
      end if
      SF=A(KING-1)-A(KING-2)
      TF=A(KING)-A(KING-1)
      QF=SF+TF
      QST=QF*SF*TF
      C(KING)=(SF*(QF+TF)*B(KING)-QF*QF*B(KING-1)+TF*TF*B(KING-2))/QST
      RETURN
      END
* -----------------------
* The subroutines POLINT and LOCATE were obtained from Numerical
*   Recipes by Press et. al., 1st edition.
* ------------
* following routine is used by Frank Chen's differentiation
*  code for getting curvatures from nozzle contour for Gortler work.
*  this code put into the BLTOSTAB.FOR program sps 10-24-94
*
* From jerrypla@eagle.larc.nasa.gov Thu Oct 20 13:37 EST 1994
* Subject: NOS CSDS CODE
      SUBROUTINE CSDS(MAX,IX,X,F,DF,S,IPT,COEF,WK,IERR)
C**********************************************************************
C*                                                                    *
C*    PURPOSE:                                                        *
C*                 SUBROUTINE CSDS FITS A SMOOTH CUBIC SPLINE TO A    *
C*                 UNIVARIATE FUNCTION.  DATA MAY BE UNEQUALLY SPACED. *
C*                                                                    *
C     E3.1
C*                                                                    *
```

```
C*       USE:                                                          *
C*               CALL CSDS(MAX,IX,X,F,DF,S,IPT,COEF,WK,IERR)           *
C*                                                                     *
C*        MAX    INPUT INTEGER SPECIFYING THE MAXIMUM NUMBER OF DATA   *
C*               POINTS FOR THE INDEPENDENT VARIABLE.                  *
C*                                                                     *
C*         IX    INPUT INTEGER SPECIFYING THE ACTUAL NUMBER OF DATA    *
C*               POINTS FOR THE INDEPENDENT VARIABLE.  IX#MAX.         *
C*                                                                     *
C*          X    ONE-DIMENSIONAL INPUT ARRAY DIMENSIONED AT LEAST      *
C*               IX IN THE CALLING PROGRAM.  UPON ENTRY TO CSDS,       *
C*               X(I) MUST CONTAIN THE VALUE OF THE INDEPENDENT        *
C*               VARIABLE AT POINT I.                                  *
C*                                                                     *
C*          F    ONE-DIMENSIONAL INPUT ARRAY DIMENSIONED AT LEAST      *
C*               IX IN THE CALLING PROGRAM.  UPON ENTRY TO CSDS,       *
C*               F(I) MUST CONTAIN THE VALUE OF THE FUNCTION AT        *
C*               POINT X(I).                                           *
C*                                                                     *
C*         DF    ONE-DIMENSIONAL INPUT ARRAY DIMENSIONED AT LEAST      *
C*               IX IN THE CALLING PROGRAM.  UPON ENTRY TO CSDS,       *
C*               DF(I) MUST CONTAIN AN ESTIMATE OF THE STANDARD        *
C*               DEVIATION OF F(I).                                    *
C*                                                                     *
C*          S    A NON-NEGATIVE INPUT PARAMETER WHICH CONTROLS THE     *
C*               EXTENT OF SMOOTHING.  S SHOULD BE IN THE RANGE        *
C*               (IX-(2*IX)**.5)#S#(IX+(2*IX)**.5).                    *
C*                                                                     *
C*        IPT    INPUT INITIALIZATION PARAMETER.  THE USER MUST        *
C*               SPECIFY IPT=-1 WHENEVER A NEW X ARRAY IS              *
C*               INPUT.  THE ROUTINE WILL THEN CHECK TO INSURE THAT    *
C*               THE X ARRAY IS IN STRICTLY INCREASING ORDER.          *
C*                                                                     *
C*       COEF    A TWO-DIMENSIONAL OUTPUT ARRAY DIMENSIONED (MAX,4)    *
C*               IN THE CALLING PROGRAM.  UPON RETURN, COEF(I,J)       *
C*               CONTAINS THE J-TH COEFFICIENT OF THE SPLINE FOR       *
C*               THE INTERVAL BEGINNING AT POINT X(I).  THE            *
C*               FUNCTIONAL VALUE OF THE SPLINE AT ABSCISSA X1,        *
C*               WHERE X(I) .LE. X1 .LE. X(I+1), IS GIVEN BY:          *
C*                  F(X1)=((COEF(I,4)*H+COEF(I,3))*H+COEF(I,2))*H      *
C*                  +COEF(I,1)                                         *
C*               WHERE H=X1-X(I)                                       *
C*                                                                     *
C*          WK   A ONE-DIMENSIONAL WORK AREA ARRAY DIMENSIONED AT      *
C*               LEAST (7*IX+9) IN THE CALLING PROGRAM.                *
C*                                                                     *
C*       IERR    OUTPUT ERROR PARAMETER:                               *
C*               =0   NORMAL RETURN.  NO ERROR DETECTED.               *
```

```
C*                =J   THE J-TH ELEMENT OF THE X ARRAY IS NOT IN      *
C*                     STRICTLY INCREASING ORDER.                     *
C*                =-1  THERE ARE LESS THAN FOUR VALUES IN THE X ARRAY.*
C*                                                                    *
C*                UPON RETURN FROM CSDS, THIS PARAMETER SHOULD BE     *
C*                TESTED IN THE CALLING PROGRAM.                      *
C*                                                                    *
C*                                                                    *
C*                                                                    *
C*    REQUIRED ROUTINES               -NONE                           *
C*                                                                    *
C*    LANGUAGE                        -FORTRAN                        *
C*                                                                    *
C*    DATE RELEASED                   SEPTEMBER 5, 1973               *
C*                                                                    *
C*    LATEST REVISION                 MARCH 1975                      *
C**********************************************************************
C
C
      DIMENSION          X(*),F(*),DF(*),COEF(MAX,*),WK(*)
C
C                                  SET UP WORKING AREAS
C
      IERR=0
      IF (IPT .NE. -1) GO TO 5
    1 IPT=0
      IF( IX .LT. 4 ) GO TO 2
      GO TO 3
    2 IERR=-1
      RETURN
    3 IX1 = IX-1
      DO 4 I = 1,IX1
      IF ( X(I + 1) -X(I) .GT. 0 ) GO TO 4
      IERR = I+1
      RETURN
    4 CONTINUE
      NP1=IX +1
      IB1 = NP1
      IB2 = IB1+NP1
      IB3 = IB2+NP1+1
      IB4 = IB3+NP1
      IB5 = IB4+NP1
      IB6 = IB5+NP1+1
      WK(1) = 0.
      WK(2) = 0.
      WK(IB2) = 0.
      WK(IB3) = 0.
      IJK2 = IB2+NP1
```

```
      WK(IJK2)=0.
      IJK5 = IB5 + 1
      WK(IJK5)=0.
      IJK5 = IB5 + 2
      WK(IJK5)=0.
      WK(IB6) = 0.
      IJK5 = IB5+NP1
      WK(IJK5)=0.
    5 CONTINUE
      P=0.
      H=X(2)-X(1)
      F2 = -S
      FF=(F(2)-F(1))/H
      IF (IX.LT.3) GO TO 25
      DO 6 I=3,IX
         G=H
         H=X(I)-X(I-1)
         E=FF
         FF=(F(I)-F(I-1))/H
         COEF(I-1,1)=FF-E
         IJK3 = IB3+I
         WK(IJK3)=(G+H)*.66666666666667
         IJK4 = IB4+I
         WK(IJK4)=H/3.
         IJK2 = IB2+I
         WK(IJK2)=DF(I-2)/G
         WK(I)=DF(I)/H
         IJK1 = IB1+I
         WK(IJK1)=-DF(I-1)/G-DF(I-1)/H
    6 CONTINUE
      DO 7 I=3,IX
         IJK1=IB1+I
         IJK2=IB2+I
         COEF(I-1,2)=WK(I)*WK(I)+WK(IJK1)*WK(IJK1)+WK(IJK2)*WK(IJK2)
         COEF(I-1,3)=WK(I)*WK(IJK1+1)+WK(IJK1)*WK(IJK2+1)
         COEF(I-1,4)=WK(I)*WK(IJK2+2)
    7 CONTINUE
C
C                                   NEXT ITERATION
C
   10 IF (IX.LT.3) GO TO 25
      DO 15 I=3,IX
         IJK1 = IB1+I-1
         IJK0 = I-1
         WK(IJK1)=FF* WK(IJK0)
         IJK2 = IB2+I-2
         IJK0 = I-2
         WK(IJK2)=G*WK(IJK0)
```

```
         IJK0 = I
         IJK3 = IB3+I
         WK(IJK0)=1./(P*COEF(I-1,2)+WK(IJK3)-FF*WK(IJK1)-G*WK(IJK2))
         IJK5 = IB5+I
         IJKN = IJK5-1
         IJK0 = IJKN-1
         WK(IJK5)=COEF(I-1,1)-WK(IJK1) * WK(IJKN)-WK(IJK2) *WK(IJK0)
         IJK4 = IB4+I
         FF=P* COEF(I-1,3)+WK(IJK4)-H* WK(IJK1)
         G=H
         H=COEF(I-1,4)* P
   15 CONTINUE
      DO 20 I=3,IX
         J=IX-I+3
         IJK5 = IB5+J
         IJK6 = IJK5+1
         IJK7 = IJK6+1
         IJK1 = IB1+J
         IJK2 = IB2+J
         WK(IJK5) = WK(J)*WK(IJK5)-WK(IJK1)*WK(IJK6)-WK(IJK2)*WK(IJK7)
   20 CONTINUE
   25 E=0
      H=0
C
C                                    COMPUTE U AND ACCUMULATE E
C
      DO 30 I=2,IX
         G=H
         IJK5 = IB5+I
         H = (WK(IJK5+1)-WK(IJK5))/(X(I)-X(I-1))
         IJK6 = IB6+I
         WK(IJK6)=(H-G)* DF(I-1) * DF(I-1)
         E=E+WK(IJK6)*(H-G)
   30 CONTINUE
      G=-H* DF(IX)* DF(IX)
      IJK6 = IB6+NP1
      WK(IJK6)=G
      E = E-G*H
      G=F2
      F2=E*P*P
      IF(F2.GE.S .OR. F2.LE.G) GO TO 45
      FF=0.
      IJK6 = IB6+2
      H = (WK(IJK6+1)-WK(IJK6))/(X(2)-X(1))
      IF (IX .LT. 3) GO TO 40
      DO 35 I=3,IX
         G=H
         IJK6 = IB6+I
```

```
         H = (WK(IJK6+1)-WK(IJK6))/(X(I)-X(I-1))
         IJK1 = IB1+I-1
         IJK2 = IB2+I-2
         G = H-G-WK(IJK1)*WK(I-1)-WK(IJK2)*WK(I-2)
         FF=FF +G * WK(I)*G
         WK(I) = G
   35 CONTINUE
   40 H=E-P*FF
      IF(H.LE.0) GO TO 45
C
C                                  UPDATE THE LAGRANGE MULTIPLIER P
C                                     FOR THE NEXT ITERATION
C
      P=P+(S-F2)/((SQRT(S/E)+P)*H)
      GO TO 10
C
C                                  IF E LESS THAN OR EQUAL TO S,
C                                  COMPUTE THE COEFFICIENTS AND RETURN.
C
   45 DO 50  I=2,NP1
         IJK6 = IB6+I
         COEF(I-1,1)=F(I-1)-P*WK(IJK6)
         IJK5 = IB5+I
      COEF(I-1,3)=WK(IJK5)
   50 CONTINUE
      DO 55  I=2,IX
         H=X(I)-X(I-1)
         COEF(I-1,4)=(COEF(I,3)-COEF(I-1,3))/(3. *H)
         COEF(I-1,2)=(COEF(I,1)-COEF(I-1,1))/H -(H*COEF(I-1,4) + COEF
     1   (I-1,3)) * H
   55 CONTINUE
 9005 RETURN
      END
```