Making a Better Wind Tunnel: Updating and Streamlining Wall Correction Methods

by

James L. Buckingham, Jr.

A project presented in partial fulfillment of the requirements
for the degree of

Bachelor's of Science in Physics

The College of William and Mary

2001

INTRODUCTION

Wind tunnel testing is used to examine the forces, pressures, and moments which will act on an object, when exposed to free airflow conditions. However, the very structure of the wind tunnel prevents accurate measurements of such phenomena. The walls of the tunnel, as they channel and focus the flow of the air, distort it, such that data measured does not correctly correspond with free air measurements. Therefore, some sort of 'wall correction' must be made to the data, to make it resemble free air conditions more exactly.

When comparing test results on the same model in different tunnels, the inherent blockage and the presence of walls in each tunnel must be taken into account. If the obstruction is more appreciable in one tunnel than the other, the data will not be comparable.

When using Computational Fluid Dynamics (CFD), the easiest situations to simulate are under free air conditions. Since these results are generally compared to experimental data, which may be biased by wall interference, either the data gained in experimentation must be corrected or the walls must be modeled in the computer simulations. This additional step can be tedious, time−consuming, and very costly.

At low speeds, speeds less than 250mph, solid wall wind tunnels require corrections which generally fit into two categories: downwash corrections and blockage corrections.

Downwash interference is caused by restriction of the streamtube curvature which would ordinarily arise due to momentum introduced into the flow by a lifting body, which causes an apparent increase of wing camber, an increase of angle of attack, and a reduction of downwash at the horizontal stabilizer. In addition, the change in streamtube curvature causes a rotation of the lift vector, which increases the induced drag of the model.

Blockage corrections are created by the reduction in the cross−sectional area of the tunnel, due to the presence of the model and its wake. To maintain continuity in the airflow, the constricted flow must increase in velocity in the region of the model. This blockage results in aerodynamic forces greater than those found in free air conditions because these forces are proportional to the square of the surrounding fluid velocity.

As modern aircraft design procedures require safety guarantees to be made years before test flights can be performed, the accuracy of wind tunnel data mapping to free air conditions has become increasingly important. As such, it is vital to ensure interference is corrected properly. As mentioned before, low speed testing is especially difficult because of these two separate types of interference. Downwash interference and blockage are maximized in low speed situations due to the large downwashes and wakes, respectively, created by the high−lift devices in use.

Consequently, wall correction techniques are necessary, particularly in these low speed tests. These corrections can only be applied to the aerodynamic load and surface pressures, but not to the flow−field velocity measurements.

Several such correction methods have been developed over the years. In 1992, Richard Beckett[1] explored the three more commonly used of these techniques:

1. The technique used at UWAL at the time for production testing, which is a by−hand process.
2. Computer code developed by Hackett[2].
3. Computer code developed by Ashill.

Beckett determined Hackett's method to be the superior of these three methods, as, though it offers nearly identical corrections as those produced by Ashill, it involves far fewer measurements of static pressures and is thus a more efficient program.

The purpose of this project is to alter Hackett's method, modernizing it for easier use in today's aeronautics industry, and for interfacing with LabView, and so that it may be modified for use both as a post−processing method, as it is currently implemented, and as an inline program, such that the data is altered as taken into the system, during the experiment. The former of these uses, i.e. post−processing, is conducive to faster data acquisition during experimentation and the latter to more quickly producing realistic results.

The actual code to be used in the modification comes from two sources. One is the original Fortran language code generated by Hackett. The other is the same program, converted into C language, begun by Luther Jenkins[3].

While it may seemingly here “reinventing the wheel,” this procedure is in no way unnecessary or superfluous. Though the Fortran code works more than adequately, that it is such an old programming language, out of date by today’s standards, makes its use impractical. Further, the ultimate goal of this process is, as mentioned before, to interface this program with LabView, in order that it may be used during data acquisition, in addition to post−processing.

LabView is a visual interface programming environment developed by National Instruments. In place of text for such common programming tools as loops, conditionals, and even the variables, icons and boxes are placed in a diagram and linked through ““wires” in sequence, providing a clear, direct view of the process of a given program. Additionally, many of LabView’s icons and boxes are designed to resemble the dials and buttons found on an instrument panel in the laboratory. For these reasons, the majority of the aeronautics industry, including NASA, Lockheed and others, uses LabView to simplify interfacing with measurement tools in the lab.

However, while LabView is quite powerful and reasonably easy to use, there are some procedures which are better implemented in a text−based code, as they require nested loops and matrix manipulation, which are more easily and efficiently handled in these text−based environments. For this reason, LabView was made to be able to interface with external, text−based code in those instances where it, LabView, is inadequate. It may then seem nonetheless imprudent to convert our Fortran code into C, rather than simply interface the LabView shell with the Fortran code. Unfortunately, though it is remarkably modern in many ways, LabView supports interface with only the C programming language. So, in order to create this interface, first a C language version of this program is needed.

THE PROGRAM

Appendix A contains the codes wsmatr.f and lsqhlt_sub.f, which is Hackett's code as modified for use at NASA Langley Research Center (LaRC) in the Basic Aerodynamics Research Tunnel (BART). This code is the prototype from which the C code is translated. In particular, the subroutines, found in lsqhlt_sub.f are the focus of this project, as the "work" portion of the main( ) function was in place, as described below.

In April of 1996, Luther Jenkins, of NASA's LaRC in Hampton, Virginia, began, essentially, this same project, that of translating the wall correction program from Fortran into C, though his goal was not at the time to interface that code with LabView in any way. To this end, he wrote all the necessary global variables/constants, as well as those local to the main( ) function only, and translated the essential part of the main( ) function. This code is what has been used as the base for the C program here.

The method for translation is simple and straightforward. The Fortran code is stepped through, line by line, and then, with the appropriate alterations, transcribed into the C code. What alterations are necessary are mostly cosmetic differences between the two programming languages. However, some variable names were changed to make their purposes more reticent.

As mentioned above, the main( ) function used is, with minor, inconsequential alterations, including the addition of statements printed to the screen for debugging purposes, that code programmed by Luther Jenkins in 1996. The primary purpose of main( ) is to open the appropriate data file(s), read in the appropriate data, and call the subroutines, which do the majority of the "work," the calculations and matrix modifications. Stepping through main( ), first all of the variables, including some local variables and constants not declared prior to main( ). Next, the user is prompted to enter the name of a file where the data to be modified is located. The file is opened and all the data read in to the appropriate variables, with several printed statements to assure the user things are going smoothly. While reading in the data, several influence matrices, containing measurements from each of the walls, are established for use in determining how much influence a given wall has on data collected at a given point and correcting that data accordingly. Pivot( ) is called during this process to locate the proper endpoints of a line when pitched and yawed. Influence( ), the base function

for the majority of the work done in this program, is called to adjust the measurements recorded appropriately. This process is done for the sources and the vertices. Finally, main( ) calls GJRV( ), which is used to invert the matrices. The rest of main( ) is focused both on running a test procedure of this sort on some test data file and printing/saving the matrices involved in that test. As such, and as the necessary work is accomplished, those portions of main( ) and their related subroutines are not implemented here.

The majority of the work is found in the functions pivot( ) and Influence( ) and subsequent subroutines of these two.

Pivot( ) is used after data acquisition to find the appropriate endpoints for the lines described in the source and vortex matrices, to keep track of each as they are moved within the tunnel. This is accomplished using simple geometry, where (x1,y1,z1) and (x2,y2,z2) are the coordinates, to be determined, of the endpoints of a given line, denoted by (x,y,z) and xpiv, pivoted by angles alfa (alpha) and psi.

Influence( ) determines the influence of each wall on a given point (xp,yp,zp) due to a line−singularity found at (x1,y1,z1) (x2,y2,z2), depending on the type of singularity in question and returns the incremental velocity components as modified by the singularity. The functions called within Influence( ) are LFUNCS( ), TFUNC( ), LNDBEQ( ), LNDBGN( ), LNVXGN( ), LNVXEQ( ), LNSCGN( ), LNSCEQ( ), and CRDTEG( ).

LFUNCS( ) returns values L1F and L2F for calculation of velocities at a given point to do a rectangular hole in an infinite sheet source. Here, the point of calculation is described by (XX,y,z) and the rectangular hole is defined by edges (y1,z1) (y2,z2), with XX = 0. Again, simple algebra and geometry are used to determine these values.

TFUNC( ) returns a value for a velocity at a given point due to a finite rectangular sheet source. The velocity is calculated at point (XX,y,z) with the rectangular sheet source located at XX = 0, with edges (y1,z1) (y2,z2). The value is calculated by adding all the contributing components, determined by simple geometry, in the for(; ;) loop.

LNDBEQ( ) is the Line Doublet equation, used to determine the velocity about a point due to a line doublet singularity. The values calculated, du, dv, dw, are the velocity components

at the point (xc,yc,zc) due to a doublet of length VL1, couple or strength VMU2, and type IT. The calculations done are simple geometry and math.

LNDBGN( ) generates the incremental velocities due to a line doublet. The end coordinates of the line doublet are defined by (xa,ya,za) (xb,yb,zb). VMU is the couple or strength of the doublet. The direction cosines of the couple vector are cx, cy, and cz. VL is the length of the line doublet. The velocities are calculated for an "arbitrary" point in space, defined by (xp,yp,zp), given by the experimental results as the location of the measurement in question. The velocities calculated, using LNDBEQ( ), du, dv, dw, are the velocity components at the point P in the (x,y,z) system. VMUR is the resultant component of the couple of the doublet, normal to the axis of the doublet. In order to eliminate point source and point sink effects in the model due to the line doublet, the component of the couple vector along the axis of the doublet is not included in determining du, dv, and dw. These values are calculated using simple math and by calling CRDTEG( ) and LNDBEQ( ).

LNVXGN( ) generates the incremental velocities due to a line vortex. The line vortex is defined by (xa,ya,za) (xb,yb,zb) and Circ, the couple or strength of the vortex. The velocities are calculated for an "arbitrary" point in space, defined by (xp,yp,zp), given by the experimental results as the location of the measurement in question. The velocities calculated, using LNVXEQ( ), du, dv, dw, are the velocity components at the point P in the (x,y,z) system. The values are calculated using calls to CRDTEG( ) and LNVXEQ( ) and simple math.

LNVXEQ( ) is the Line Vortex equation, called by LNVXGN( ). Circ, xpG, ypG, and zpG define the line vortex. VL is the length of the line vortex. The values calculated, duG, dvG, and dwG are the incremental velocities due to the line vortex. These are calculated using simple geometry and algebra.

LNSCGN( ) generates the incremental velocities due to a line source. The endpoints of the line source are defined by (xa,ya,za) (xb,yb,zb). The couple (strength) is defined by VM. The velocity increments du, dv, and dw are calculated about an "arbitrary" point (xp,yp,zp), given by the experimental results as the location of the measurement in question. These velocities are calculated using simple math and calls to LNSCEQ( ) and CRDTEG( ).

LNSCEQ( ) is the Line Source equation, used in calculating the incremental velocities due to a line source singularity. The couple or strength of the line source is given by VM. The length is given by VL. The incremental velocities, duG, dvG, and dwG, are calculated about the point (xpG,ypG,zpG) and calculated using simple mathematics and geometry.

CRDTEG( ) translates the coordinates from one system to another, i.e. from “Greek to English” or “English to Greek.” “English” coordinates are standard Cartesian coordinates (x,y,z). “Greek” coordinates (Exi,Eta,Zta) are defined with the Exi axis parallel to the center line of the doublet. If IC = 1, the translation to be done is from English to Greek. If IC = 2, the translation is to be from Greek to English. The ends of the line double are A and B with the coordinates for each point defined by (xa0,ya0,za0) and (xb0,yb0,zb0) respectively. The point (xp0,yp0,zp0) is a general point in space given by the experimental results as the location of the measurement in question. It should be noted that the system is set up for positive winding numbers for rotation of the axes, where EPSI and ALFA (alpha) are the angles of rotation of the Eta (or Y) and Zta (or Z) axes, respectively.

GJRV( ) is used actually to invert the matrices. A is the input array which will be altered. N is the rank of A. NL is the row dimension of A. Epsil is the test value for the pivot point singularity check. And, Ierr is nonzero if the matrix is singular. If the matrix is nonsingular, A contains A−inverse. In order to invert the matrix, first the largest element is found and placed in the pivot. Then, the columns are swapped around the pivot.

To make sure the C code works, first a line−by−line comparison to the Fortran version is performed. As expected, in slightly less space than the Fortran code, the C code accomplishes the same tasks. Next, the C program is compiled and run on the file “testinput.dat,” provided by Luther Jenkins. This test data is also used in debugging the C code, with print statements placed to display the data in its current incarnation after each of the important steps in the process, i.e. after each call to pivot( ) or Influence( ).

Running this test shows that the data is inputted properly from the test file and that pivot( ) and Influence( ) are working properly. Also, the matrices are all properly established and properly inverted as needed. The rest of the program, superfluous to the data modification process described above, is neither implemented nor tested, but rather “commented out” in order that it doesn’t disturb the processing of the rest of the code.

WHAT IS LEFT TO DO?

There are several things remaining to be accomplished in this project. The remainder of the Fortran code, that including the test run on sample data and the printing of the matrices in proper format, while unnecessary, is a helpful procedure which will need implementation at a later date. The program needs to be linked to a DLL (Dynamic Link Library) so that it can be accessed by other programs within the Windows operating system, including LabView, and a shell needs to be created with LabView for accessing that DLL and, therefore, this program. These last two are the only remaining steps in accomplishing the ultimate goal of this project, to create access to this program through LabView that it may be used either as a post−processing operation or an “online” operation, processed as the data is acquired.

References

1. R. Beckett, An Evaluation of Three Wind Tunnel Wall Interference Correction Techniques, University of Washington Department of Aeronautics and Astronautics, 1992

2. J. E. Hackett, S. Sampath, and C.G. Phillips, Determination of Wind Tunnel Constraint Effects By a Unified Wall Pressure Signature Method: Part I. Applications to Winged Configurations, NASA CR 166, 1981

3. L. N. Jenkins, Wall Correction Software (in C and in Fortran), NASA Langley Research Center (LaRC), Personal contact

Appendix A

The following contains wsmatr.f and lsqhlt_sub.f, Hackett's code as modified for use at NASA Langley Research Center (LaRC) Basic Aerodynamics Research Tunnel (BART).

```
      PROGRAM LSQITER                                    !VAX
C−  Iterative Solution Accounting For Cross_effects Of Vortices
C−  and Sources. Least_Square Approach.
C−
C−  Version : HLT_DATA_REDUCTION.
C−  TAPE7 : Load_Coefficients. Experimental And Corrected.
C_  TAPE8 : Matrix File
C_  TAPE9 : Results.  Input and Calculated Signatures, and Center_line
C−                Interference Velocities.
C−  TAPE10: HLT Experimental Data
C−  TAPE11: Interference Velocities Due to Jets.
C−  TAPE16: INFLUENCE COEFFT MATRICES (INPUT)
C−
c **********Plotting stuff*******************************
      INCLUDE 'plotcommon.blk'
              INCLUDE 'plotinit.blk'
C       **** Window stuff ******
                        integer*4  InitGraphWindow      ! function call in uis_lib.f
                        integer*4  myWindow(12)         ! storage for the window pointer
                        common /oneplace/myWindow
              DATA wtitle(1),wtitle(2),wtitle(3)/'BLOCKAGE','LIFT INTERFERENCE',
     +     'LIFT CURVE'/
              DATA alphamin,alphamax,clmin,clmax/−10.,40.,−.5,1.5/
C ********************************************************************
              character*1 ans
              REAL xarray(30),yarray(30),yarray2(30),yarray3(30)
              REAL cpemwl(25),cpemrf(25)
      COMMON/IMAT/ UGRF(25,25),UQWL(25,25),UGCL(25,25),UQCL(25,25),
     *        WGCL(25,25),WQCL(25,25),UGWL(25,25),WGWL(25,25),
     *        UQRF(25,25)
      COMMON/LMAT/ AQ(25,25),AG(25,25)
      COMMON/ARR1/ XR(25),YR(25),ZR(25),XL(25),YL(25),ZL(25),
     *        XV(25),YV(25),ZV(25),XS(25),YS(25),ZS(25),
     *        SBV(25),SBS(25),PSIV(25),ALFV(25),PSIS(25),ALFS(25),
     *        CPOWL(25),UCPWL(25),UCPRF(25),UXQRF(25),UXGWL(25),
     *        WXGWL(25),ULIFT(25),UBLKG(25),GAMA(25),SIGMA(25)
      COMMON/IMAG/ B,H,LAYERS,IRF
      COMMON/SIZE/ NR,NW,NV,NS,XPVOR,XPSRC
      COMMON /BLKB/ ALFU,POU,QOU,CMUU,CLU,CDU,CMU
      DIMENSION VWALL(25,3),VROOF(25,3),VCNTR(25,3) !,TITLE(80)
      CHARACTER TAPE7*13,TAPE9*13, PFIX*3,TITLE*80
      PI = 3.142592654
      RAD = PI/180.0
      DEG = 180.0/PI
              IPOINT = 0
C−
C−  READ INPUT
C−          ***  MAIN INPUT ***
              CALL MoveOutWindow(500,50,1000,250)
              WRITE(6,'(a,$)') Input the upwash angle −−> '
              READ(5,*)alfupw
              WRITE(6,*)'Open the main input file'
              OPEN(2,file=*,status='old')
C−
      READ (2,'(a)') TITLE
      READ (2,*) ITERMAX,MATPRT,MATSAV,IPRT,KROSG,KROSQ,ICORR,JETEFCT
      READ (2,*) SAREA,AWB,BWB, EC1,EC2,EC3, XBCOR
              WRITE(6,*)title
              WRITE(6,*)ITERMAX,MATPRT,MATSAV,IPRT,KROSG,KROSQ,ICORR,JETEFCT
              WRITE(6,*)SAREA,AWB,BWB, EC1,EC2,EC3, XBCOR
              WRITE(6,*)'Read in main input'
      ICOR = ABS(ICORR)
      IF(KROSG+KROSQ .EQ. 0) ITERMAX = 1
      IF(MATSAV .EQ. 2) THEN
            READ (2,*) LAYERS,IRF,NR,NW,NV,NS
            READ (2,*) B,H,XPVOR,XPSRC
                        DO I=1,nr
                                    READ(2,*)(dum,j=1,3)
```

```
                        END DO
                        DO I=1,nw
                                READ(2,*)(dum,j=1,3)
                        END DO
                        DO I=1,nv
                                READ(2,*)(dum,j=1,6)
                        END DO
                        DO I=1,ns
                                READ(2,*)(dum,j=1,6)
                        END DO
                        READ(2,*)(cpemrf(j),j=1,nr),(cpemwl(j),j=1,nw)
                        GO TO 300
                END IF
      IF(MATSAV .LE.1) GO TO 10
      CALL TAPEIO(3,2)
      MATSAV = 1
      GO TO 95
C−
C−          *** GEOMETRY INPUT ***
C−
  10  READ (2,*) LAYERS,IRF,NR,NW,NV,NS
      READ (2,*) B,H,XPVOR,XPSRC
      DO 11 I = 1,NR
      READ (2,*) XR(I),YR(I),ZR(I)
          XR(I) = XR(I)*B
          YR(I) = YR(I)*B
          ZR(I) = ZR(I)*H
  11  CONTINUE
      DO 15 I = 1,NW
      READ (2,*) XL(I),YL(I),ZL(I)
          XL(I) = XL(I)*B
          YL(I) = YL(I)*B
          ZL(I) = ZL(I)*H
  15  CONTINUE
      DO 20 I = 1,NV
      READ (2,*) XV(I),YV(I),ZV(I),SBV(I),PSIV(I),ALFV(I)
          XV(I) = XV(I)*B
          YV(I) = YV(I)*B
          ZV(I) = ZV(I)*H
          SBV(I) = SBV(I)*B
          PSIV(I) = PSIV(I)*RAD
          ALFV(I) = ALFV(I)*RAD
  20  CONTINUE
      DO 25 I = 1,NS
      READ (2,*) XS(I),YS(I),ZS(I),SBS(I),PSIS(I),ALFS(I)
          XS(I) = XS(I)*B
          YS(I) = YS(I)*B
          ZS(I) = ZS(I)*H
          SBS(I) = SBS(I)*B
          PSIS(I) = PSIS(I)*RAD
          ALFS(I) = ALFS(I)*RAD
  25  CONTINUE
                READ(2,*)(cpemrf(j),j=1,nr),(cpemwl(j),j=1,nw)
      WRITE(6,*)'Read in the geometry input'
C−
C−  SET UP SOURCE_INFLUENCE MATRICES
C−
      MINIT = 0
      DO 60 J = 1,NS
      XPIV = XPSRC*SBS(J)
      CALL PIVOT(XS(J),YS(J),ZS(J), XPIV, SBS(J)/2.0,PSIS(J),ALFS(J),
     *          X1,Y1,Z1, X2,Y2,Z2)
C...CROSS EFFECT SOURCE/(ROOF − FLOOR)
      U3 = 0.
      U4 = 0.
      DO 40 I = 1,NR
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 0.5, 1, MINIT, 1,
     *          XR(I),YR(I),ZR(I), U1,V1,W1)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 0.5, 1, MINIT, 1,
     *          XR(I),YR(I),ZR(I), U2,V2,W2)
      IF(IRF .EQ. 0) GO TO 38
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 0.5, 1, MINIT, 1,
     *          XR(I),YR(I),−ZR(I), U3,V3,W3)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 0.5, 1, MINIT, 1,
     *          XR(I),YR(I),−ZR(I), U4,V4,W4)
  38  UQRF(I,J) = U1+U2−(U3+U4)
  40  CONTINUE
```

```
C...DIRECT EFFECT. SOURCE/WALL
      DO 50 I = 1,NW
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 0.5, 1, MINIT, 1,
     *          XL(I),YL(I),ZL(I), U1,V1,W1)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 0.5, 1, MINIT, 1,
     *          XL(I),YL(I),ZL(I), U2,V2,W2)
      UQWL(I,J) = U1+U2
   50 CONTINUE
C...  CENTER_LINE INTERFRERNCE MATRIX.(X CORRESPONDDS TO WALL POINTS)
      DO  55 I = 1,NW
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 0.5, 1, 1, 1,
     *          XL(I),0.0,0.0, U1,V1,W1)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 0.5, 1, 1, 1,
     *          XL(I),0.0,0.0, U2,V2,W2)
      UQCL(I,J) = U1+U2
      WQCL(I,J) = W1+W2
   55 CONTINUE
   60 CONTINUE
C−
C−  SET UP INFLUENCE MATRICES FOR VORTICES
C−
      DO 90 J = 1,NV
      XPIV = XPVOR*SBV(J)
      CALL PIVOT(XV(J),YV(J),ZV(J), XPIV, SBV(J)/2.0,PSIV(J),ALFV(J),
     *          X1,Y1,Z1, X2,Y2,Z2)
C...DIRECT EFFECT. VORTEX/(ROOF − FLOOR)
      U3 = 0.
      U4 = 0.
      DO 70 I = 1,NR
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 1.0, 2, MINIT, 1,
     *          XR(I),YR(I),ZR(I), U1,V1,W1)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 1.0, 2, MINIT, 1,
     *          XR(I),YR(I),ZR(I), U2,V2,W2)
      IF(IRF .EQ. 0) GO TO 68
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 1.0, 2, MINIT, 1,
     *          XR(I),YR(I),−ZR(I), U3,V3,W3)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 1.0, 2, MINIT, 1,
     *          XR(I),YR(I),−ZR(I), U4,V4,W4)
   68 UGRF(I,J) = U1+U2−(U3+U4)
   70 CONTINUE
C...CROSS EFFECT. VORTEX/WALL
      DO 80 I = 1,NW
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 1.0, 2, MINIT, 1,
     *          XL(I),YL(I),ZL(I), U1,V1,W1)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 1.0, 2, MINIT, 1,
     *          XL(I),YL(I),ZL(I), U2,V2,W2)
      UGWL(I,J) = U1+U2
      WGWL(I,J) = W1+W2
   80 CONTINUE
C...  CENTER_LINE INTERFRERNCE MATRIX.(X CORRESPONDDS TO WALL POINTS)
      DO  85 I = 1,NW
      CALL INFLU(X1,Y1,Z1, X2,Y2,Z2, 1.0, 2, 1, 1,
     *          XL(I),0.0,0.0, U1,V1,W1)
      CALL INFLU(X2,−Y2,Z2, X1,−Y1,Z1, 1.0, 2, 1, 1,
     *          XL(I),0.0,0.0, U2,V2,W2)
      UGCL(I,J) = U1+U2
      WGCL(I,J) = W1+W2
   85 CONTINUE
   90 CONTINUE
C−
C−  FORM [A]_INVERSE FOR SOURCE/WALL
C−
      WRITE(6,*)'Forming the inverse matrices'
   95 EPS = 0.0
      DO 110 I = 1,NS
      DO 110 J = 1,NS
      AQ(I,J) = 0.0
      DO 100 K = 1,NW
      AQ(I,J) = AQ(I,J) + UQWL(K,I)*UQWL(K,J)
  100 CONTINUE
  110 CONTINUE
      IERR = 0
      CALL GJRV(AQ,NS,25,EPS,IERR)
      IF(IERR .NE. 0) GO TO 310
C−
C−  FORM [A]_INVERSE FOR VORTICES/ROOF
C−
```

```
      EPS = 0.0
      DO 210 I = 1,NV
      DO 210 J = 1,NV
      AG(I,J) = 0.0
      DO 200 K = 1,NR
      AG(I,J) = AG(I,J) + UGRF(K,I)*UGRF(K,J)
  200 CONTINUE
  210 CONTINUE
      IERR = 0
      CALL GJRV(AG,NV,25,EPS,IERR)
      IF(IERR .NE. 0) GO TO 310
C−
C−  OPTIONALLY PRINT/SAVE MATRICES
C−
  300 IF(MATSAV .NE. 0) CALL TAPEIO(MATSAV,8)
      IF(IPRT .EQ. 0) GO TO 305
      WRITE (6,3100) TITLE
      WRITE (6,3200) B,H,LAYERS
  305 IF(MATPRT .EQ. 0) GO TO 320
  310 CALL OUT(UGRF, 25,25, NR,NV, 'U_GAMA_ROOF',11)
      CALL OUT(UQWL, 25,25, NW,NS, 'U_SORC_WALL',11)
      CALL OUT(UQRF, 25,25, NR,NS, 'U_SORC_ROOF',11)
      CALL OUT(UGWL, 25,25, NW,NV, 'U_GAMA_WALL',11)
      CALL OUT(WGWL, 25,25, NW,NV, 'W_GAMA_WALL',11)
      IF(IERR .NE. 0) STOP
C−
C−          *** TEST SIGNATURE INPUT ***
C−
C−  READ_IN RUN NUMBER AND WALL SIGNATURES
C−  SET_UP BOUNDARY CONDITIONS
C−
C−  ********************************************************************!
C−      The structure of the program for INPUTing run identification    !
C−      and wall signatures in the next 15 statements are for the       !
C−      Knee_Blown_Flap experiments of LOCKHEED_GEORGIA.  The user      !
C−      should make appropriate changes in this section and re_write    !
C−      subroutine "READCP" to define the arrays "UCPRF" and "UCPWL"    !
C−      as indicated below:                                             !
C−          UCPRF(I) = (U,ROOF(I)−U,FLOOR(I)), if IRF = 1               !
C−                   =  U,ROOF(I)            if IRF = 0                 !
C−          UCPWL(I) = (U,WALL1(I)+U,WALL2(I))/2                        !
c 320    READ (2,*,END=999) ITEST,IRUN,IPMIN,IPMAX,IFLOOR,IROOF,        !
c     *                     IWAL1,IWAL2                                 !
  320    OPEN(7,file=*'Create corrected data file',status='unknown')
       WRITE(7,'(12a8)')"aoau","qu","CLu","CDu","Cmu","aoac","qc",
     +          "CLc","CDc","Cmc","qc/qu","Æ aoa"
                            OPEN(9,file=*'Create pressure sign. file',status='unknown')
C − Calculate the empty tunnel q calibration at the specifiec model location
        qoe = 1.−(cpemwl(icorr) + cpemrf(icorr))/2.
                            write(6,*)qoe
                            WRITE(6,*)'Where is the run time force data?'
                            OPEN(10,file=*,status='old')
                            WRITE(6,*)'Where is the run time pressure data?'
                            OPEN(11,file=*,status='old')
c                           DO i=1,4
c                             READ(10,*)
c                           END DO
                            DO WHILE(ipoint .LT. 99)
C − Read in the uncorrected coefficients
                              READ(10,end = 600,*)alfu,qou,clu,cdu,cmu
                              IF(alfu .NE. 999.)THEN
          ipoint = ipoint + 1
                               WRITE(6,*)'ipoint',ipoint,nw,alfu
                               READ(11,*)dum
C − Read in the tunnel wall pressure coefficients
                               READ(11,*)(ucprf(i),i=1,nr),(ucpwl(i),i=1,nw)
c       CALL READCP(UCPRF, NR, IRUN, IP, IFLOOR, IROOF, 1, IRF)         !
c       CALL READCP(UCPWL, NW, IRUN, IP, IWAL1,  IWAL2, 0, IRF)         !
C−                                                                      !
C−  ********************************************************************!
C − Remove the empty tunnel pressure signature from the data
C −     (Done with D. Wilsden's equation
        DO i=1,nr
                              ucprf(i) = ucprf(i) − cpemrf(i)*(1.−ucprf(i)+cpemrf(i))
                             END DO
        DO i=1,nw
                              ucpwl(i) = ucpwl(i) − cpemwl(i)*(1.−ucpwl(i)+cpemwl(i))
```

```
                        END DO
C − Convert the pressure coefficients to velocity increments
      DO i=1,nr
                        q = 1.−ucprf(i)
                        ucprf(i) = SQRT(q)−1.
                       END DO
      DO i=1,nw
                        q = 1.−ucpwl(i)
                        ucpwl(i) = SQRT(q)−1.
                       END DO

      IF(IPRT.NE.0)WRITE(6,'(a,f5.2)')'angle of attack of ',alfu
      IF(IPRT .NE. 0) WRITE (6,3300)
 350   DO 370 I = 1,NR
       UXQRF(I) = 0.0
       IF(IPRT .NE. 0) WRITE (6,1200) I,XR(I)/B,YR(I)/B,ZR(I)/H,
     *                  UCPRF(I)
 370   CONTINUE
C
      IF(IPRT .NE. 0) WRITE (6,3400)
       DO 380 I = 1,NW
        CPOWL(I) = 1.0−(UCPWL(I)+1.0)**2
     IF(IPRT .NE. 0) WRITE (6,1200) I,XL(I)/B,YL(I)/B,ZL(I)/H, UCPWL(I)
 380    CONTINUE
C−
C−  BEGIN ITERATIVE CYCLE
C−
      ITER = 0
 400   ITER = ITER + 1
      IF(IPRT .NE. 0) WRITE (6,3500) ITER
C−
C−  SUBTRACT CROSS_INFLUENCE OF SOURCES FROM ROOF SIGNATURE AND
C−  FORM LEAST_SQUARE COLUMN VECTOR FOR VORTICES
C−
      DO 410 I = 1,NV
       ULIFT(I) = 0.0
      DO 410 K = 1,NR
 410   ULIFT(I) = ULIFT(I) + UGRF(K,I)*(UCPRF(K)−UXQRF(K) )
C−
C−  COMPUTE GAMA'S
C−
      GTOT = 0.0
      DO 425 I = 1,NV
       GAMA(I) = 0.0
       DO 420 J = 1,NV
 420     GAMA(I) = GAMA(I) + AG(I,J)*ULIFT(J)
      GTOT = GTOT + GAMA(I)/SQRT(B*H)
 425   CONTINUE
C−
C−  COMPUTE CROSS_EFFECT OF GAMA'S ON SIDE_WALL
C−
      DO 430 I = 1,NW
       UXGWL(I) = 0.0
       WXGWL(I) = 0.0
       IF(KROSG .EQ. 0) GO TO 430
       DO 430 J = 1,NV
        UXGWL(I) = UXGWL(I) + UGWL(I,J)*GAMA(J)
        WXGWL(I) = WXGWL(I) + WGWL(I,J)*GAMA(J)
 430    CONTINUE
C−
C−  SUBTRACT CROSS_INFLUENCE OF GAMA'S FROM WALL SIGNATURE AND
C−  FORM LEAST_SQUARE COLUMN VECTOR FOR SOURCES
C−
       FACTOR = 1.0
 436   CONTINUE
      DO 440 I = 1,NS
       UBLKG(I) = 0.0
       DO 440 K = 1,NW
        UTEMP = 1.0 − CPOWL(K) − FACTOR*WXGWL(K)**2
        IF(UTEMP .GT. 0.0) GO TO 438
        IF(FACTOR .LE. 0.1E−06) GO TO 610
        FACTOR = 0.0
        GO TO 436
 438     UTEMP = SQRT(UTEMP) − UXGWL(K) − 1.0
       UBLKG(I) = UBLKG(I) + UQWL(K,I)*UTEMP
 440   CONTINUE
C−
```

```
C−  COMPUTE SIGMA'S
C−
      QTOT = 0.0
      DO 455 I = 1,NS
       SIGMA(I) = 0.0
       DO 450 J = 1,NS
        SIGMA(I) = SIGMA(I) + AQ(I,J)*UBLKG(J)
 450    CONTINUE
       QTOT = QTOT + SIGMA(I)/(B*H)
 455   CONTINUE
C−
C−  PRINT OUT GAMA'S, SIGMA'S AND CROSS_EFFECT TERMS
C−
      IF(IPRT .EQ. 0) GO TO 490
      WRITE (6,3600)
      WRITE (6,3700)
      DO 460 I = 1,NV
       GP = GAMA(I)/SQRT(B*H)
       WRITE (6,1300) I,XV(I)/B,YV(I)/B,ZV(I)/H,
     *          SBV(I)/B,PSIV(I)/RAD,ALFV(I)/RAD,GP
 460   CONTINUE
      WRITE (6,3900) GTOT
      WRITE (6,3800)
      WRITE (6,3700)
      DO 465 I = 1,NS
       QP = SIGMA(I)/(B*H)
       WRITE (6,1300) I,XS(I)/B,YS(I)/B,ZS(I)/H,
     *          SBS(I)/B,PSIS(I)/RAD,ALFS(I)/RAD, QP
 465   CONTINUE
      WRITE (6,3900) QTOT
      IMAX = NR
      IF(NW .GT. IMAX) IMAX = NW
      WRITE (6,4300)
      DO 470 I = 1,IMAX
       WRITE (6,1400) I,UXQRF(I),UXGWL(I),WXGWL(I)
    IF((1.0 − CPOWL(I) − WXGWL(I)**2) .LT. 0.0) WRITE (6,1600)
 470   CONTINUE
      IF(FACTOR .LE. 0.1E−06) WRITE (6,6460)
C−
C−  RE_CALCULATE WALL BOUNDARY CONDITIONS, U , ON ROOF
C−
      WRITE (6,4000)
 490   DO 510 I = 1,NR
       VROOF(I,1) = 0.0
       DO 500 J = 1,NV
 500    VROOF(I,1) = VROOF(I,1) + UGRF(I,J)*GAMA(J)
       IF(KROSQ .EQ. 0) GO TO 506
       DO 505 J = 1,NS
 505    VROOF(I,1) = VROOF(I,1) + UQRF(I,J)*SIGMA(J)
 506    CPC = −VROOF(I,1)**2 − 2.0*VROOF(I,1)
    IF(IPRT .NE. 0) WRITE (6,1200) I,XR(I)/B,YR(I)/B,ZR(I)/H,
     *                    VROOF(I,1),CPC
 510   CONTINUE
C−
C−  RE_CALCULATE BOUNDARY CONDITIONS U & W ON SIDE WALL
C−
      IF(IPRT .NE. 0) WRITE (6,4100)
      DO 530 I = 1,NW
    VWALL(I,1) = 0.0
    VWALL(I,3) = 0.0
    DO 520 J = 1,NS
 520  VWALL(I,1) = VWALL(I,1) + UQWL(I,J)*SIGMA(J)
    IF(KROSG .EQ. 0) GO TO 528
    DO 525 J = 1,NV
    VWALL(I,1) = VWALL(I,1) + UGWL(I,J)*GAMA(J)
 525  VWALL(I,3) = VWALL(I,3) + WGWL(I,J)*GAMA(J)
 528  ULIFT(I) = −(FACTOR*VWALL(I,3)**2+VWALL(I,1)**2+2.0*VWALL(I,1) )
    IF(IPRT .NE. 0) WRITE (6,1200) I,XL(I)/B,YL(I)/B,ZL(I)/B,
     *          VWALL(I,1),VWALL(I,3),  ULIFT(I)
 530  CONTINUE
C−
C−  OBTAIN CENTER_LINE INTERFERENCE VELOCITIES
C−
    IF(ITER .EQ. ITERMAX) WRITE (9,'(f6.2)') alfu
    IF(IPRT .NE. 0) WRITE (6,4200)
    YCENTER = 0.
    ZCENTER = 0.
```

```
      DO 570 I = 1,NW
      UQCTL = 0.0
      WQCTL = 0.0
      DO 550 J = 1,NS
      UQCTL = UQCTL + UQCL(I,J)*SIGMA(J)
      WQCTL = WQCTL + WQCL(I,J)*SIGMA(J)
  550 CONTINUE
      UGCTL = 0.0
      WGCTL = 0.0
      DO 560 J = 1,NV
      UGCTL = UGCTL + UGCL(I,J)*GAMA(J)
      WGCTL = WGCTL + WGCL(I,J)*GAMA(J)
  560 CONTINUE
      VCNTR(I,1) = UQCTL + UGCTL
      VCNTR(I,3) = WQCTL + WGCTL
      IF(IPRT .NE. 0) WRITE (6,1200) I,XL(I)/B,YCENTER,ZCENTER,
     *              VCNTR(I,1),VCNTR(I,3), UQCTL,UGCTL,WQCTL,WGCTL
      IF(ITER .EQ. ITERMAX) WRITE (9,1500) I,XL(I)/B, CPOWL(I),ULIFT(I),
     *  VCNTR(I,1), UCPRF(I), VROOF(I,1),VCNTR(I,3)
  570 CONTINUE
C−
C−  TERMINATE IF REQUIRED NO OF ITERARIONS HAVE BEEN PERFORMED.
C−  OTHERWISE, DETERMINE CROSS_INFLUENCE U DUE TO SOURCES ON ROOF
C−  AND CONTINUE ITERATIONS.
C−
      IF(ITER .GE. ITERMAX) GO TO 590
      DO 580 I = 1,NR
      UXQRF(I) = 0.0
      IF(KROSQ .EQ. 0) GO TO 580
      DO 580 J = 1,NS
      UXQRF(I) = UXQRF(I) + UQRF(I,J)*SIGMA(J)
  580 CONTINUE
      GO TO 400
  590 CONTINUE
      IPASS = 1
      IF(JETEFCT .EQ. 0) WRITE (6,6100)TITLE,ITEST,IRUN,IP
      IF(JETEFCT .NE. 0) WRITE (6,6110)TITLE,ITEST,IRUN,IP
                ymin1 = −.05
                ymin2 = −.05
                ymax1 = .05
                ymax2 = .05
                        do i=1,nw
                          xarray(i) = xl(i)/b
                          yarray(i) = vcntr(i,1)
                          yarray3(i) = vcntr(i,3)
                          yarray2(i) = vroof(i,1)
                          ymin1 = MIN(ymin1,cpowl(i),ulift(i),yarray(i))
                          ymin2 = MIN(ymin2,ucprf(i),yarray2(i),yarray3(i))
                          ymax1 = MAX(ymax1,cpowl(i),ulift(i),yarray(i))
                          ymax2 = MAX(ymax2,ucprf(i),yarray2(i),yarray3(i))
                        end do
                 IF(ipoint .NE. 1)THEN
                        DO i=1,2
                          CALL place(i)
                  CALL finitt
                        END DO
                        END IF
                 CALL initt(1)
                        CALL binitt
                        CALL window_pos(1.,7.)
                        CALL slim(5.,4.)
                        CALL dlimx(0.,3.)
                        CALL dlimy(ymin1,ymax1)
                 CALL iline(1)
                 CALL pen_color(2)
                 CALL npts(nw)
                 CALL symbl(1)
                 CALL Dsplay(xarray,cpowl)
                        CALL symbl(2)
                 CALL Cplot(xarray,ulift)
                        CALL symbl(3)
                        CALL Cplot(xarray,yarray)
                        CALL initt(2)
                        CALL binitt
                        CALL window_pos(7.,7.)
                        CALL slim(5.,4.)
                        CALL dlimx(0.,3.)
```

```
                    CALL dlimy(ymin2,ymax2)
                    CALL pen_color(6)
                    CALL npts(nw)
                    CALL symbl(1)
                CALL Dsplay(xarray,ucprf)
                    CALL symbl(2)
                CALL Cplot(xarray,yarray2)
                    CALL symbl(3)
                    CALL Cplot(xarray,yarray3)
591   IF(IPRT .NE. 0)THEN
      CALL OUTPUT(NW,NS,XS,SBS(1),0,
     *            UCPWL,XL,SIGMA,VCNTR,VWALL,ICOR, IPASS−1)
      CALL OUTPUT(NR,NV,XV,SBV(1),1,
     *            UCPRF,XR,GAMA, VCNTR,VROOF,ICOR, IPASS−1)
            END IF
C−
C−  CORRECT ALL MEASURED QUANTITIES
C−
 592  IF (ICORR .GT. 0) THEN
        DU = VCNTR(ICOR,1)
        DW = VCNTR(ICOR,3)
      ELSE
        CALL INTER (XL, VCNTR(1,1), XBCOR*B, DU, NW, 1)
        CALL INTER (XR, VCNTR(1,3), XBCOR*B, DW, NR, 1)
      END IF
C−−−−−
C     Correction to Load Coeffts. using Delta_U and Delta_W of "LSQ"
C−−−−−
      FACT = (1.0+DU)**2
      QOC = QOU*FACT*qoe
      CMUC = CMUU/FACT
      CLC = CLU/FACT/qoe
      CMC = CMU/FACT/qoe
      DELA = DW/(1.0+DU)
      ALFC = ALFU + DELA*DEG
      BHBYS = B*H/SAREA
      DELCD = −BHBYS*QTOT**2
      CDC = (CDU−DELCD)/FACT/qoe
C−
C − Add in the user specified upwash angle
      dela = dela + alfupw*RAD
                alfc = alfc + alfupw
      SINA = SIN(DELA)
      COSA = COS(DELA)
      CD = CDC*COSA + CLC*SINA
      CL = CLC*COSA − CDC*SINA
C−
      IF(IPRT.NE.0)THEN
      WRITE (6,6440) YL(ICOR)/B,ZL(ICOR)/H, YR(ICOR)/B,ZR(ICOR)/H,
     *            DU,DW,DELCD
      WRITE (6,6400)
      WRITE (6,6410) ALFU,POU,QOU,CMUU,CLU,CDU,CMU
      WRITE (6,6420) ALFC,POU,QOC,CMUC,CLC,CDC,CMC
      WRITE (6,6430) ALFC,POU,QOC,CMUC,CL, CD, CMC, DELA*DEG
                END IF
C−−−−−
C     Correction to Load Coeffts. by Conventional Methods
C−−−−−
      DELAP = AWB*CLC/(1.+BWB*CMUC)          ! Williams & Butler D_Alfa
      ALFP = ALFU + DELAP*DEG
      SINA = SIN(DELAP)
      COSA = COS(DELAP)
      IF (EC1*EC2*EC3 .GT. 0.0) THEN
        EPN = EC1 + EC2*(CDU − EC3*CLU**2)
        FACT = 1.0     !     /(1.0+EPN)**2   !<== NOTE SUPRESSION OF DIVISION
        CL1 = CLU*FACT
        CD1 = CDU − EC1*(CDU − EC3*CLU**2)
        QO1 = QOU*FACT
        CMU1= CMUU*FACT
        CM1 = CMU*FACT
      ELSE
        CL1 = CLC
        CD1 = CDC
        QO1 = QOC
        CMU1= CMUC
        CM1 = CMC
      END IF
```

```
      CDP = CD1*COSA + CL1*SINA
      CLP = CL1*COSA − CD1*SINA
      IF(IPRT.NE.0)THEN
      WRITE (6,6431) ALFP, POU, QO1, CMU1, CLP, CDP, CM1, DELAP*DEG
                END IF
C−
C−  ADD EFFECT OF JET/S, IF PRESENT.
C−
      IF(JETEFCT .EQ. 0) GO TO 595
      IF(IPASS .EQ. 2) GO TO 595
      IPASS = 2
      CALL JETADD(NW, NR, VWALL,VROOF, VCNTR, ITEST,IRUN,IP)
      IF(IPRT.NE.0)WRITE (6,6120) TITLE,ITEST,IRUN,IP
      GO TO 591
C−  WRITE FINAL RESULTS TO MASS STORAGE
C−
 595   WRITE(7,'(2(2f6.2,3f9.4),3f9.4)')alfu,qou,clu,cdu,cmu,
     +      alfc,qoc,clc,cdc,cmc,qoc/qou,(alfc−alfu)
       WRITE(6,'(2(2f6.2,3f9.4),3f9.4)')alfu,qou,clu,cdu,cmu,
     +      alfc,qoc,clc,cdc,cmc,qoc/qou,(alfc−alfu)
c 595  WRITE (7,1100) ITEST, IRUN, IP
c      WRITE (7,1000) ALFU, POU, QOU, CMUU, CLU, CDU, CMU
c      WRITE (7,1000) ALFC, POU, QOC, CMUC, CLC, CDC, CMC, DU
c      WRITE (7,1000) ALFC, POU, QOC, CMUC, CL,  CD,  CMC, DELA*DEG
c      WRITE (7,1000) ALFP, POU, QO1, CMU1, CL1, CD1, CM1, EPN
c      WRITE (7,1000) ALFP, POU, QO1, CMU1, CLP, CDP, CM1, DELAP*DEG
C  ******PLOT THE DATA
      IF (ipoint .EQ. 1)THEN
                 CALL initt(3)
                          CALL binitt
                          CALL window_pos(.75,1.)
                          CALL slim(5.,5.5)
                          CALL dlimx(alphamin,alphamax)
                          CALL dlimy(clmin,clmax)
                          CALL pen_color(4)
                          CALL symbl(1)
                          CALL npts(1)
                          CALL Dsplay(alfu,clu)
                ELSE
                  CALL place(3)
                          CALL npts(1)
                  CALL pen_color(4)
                  CALL symbl(1)
                          CALL Cplot(alfu,clu)
                END IF
                          CALL symbl(2)
                          CALL Cplot(alfc,clc)
c                         CALL symbl(3)
c                         CALL Cplot(alfp,clp)
      ELSE
                 ipoint = 99
                END IF
      END DO   ! used to be GO TO 330
 600  IP = 0
                WRITE(6,*)'Got to end of data (hit any key)'
                READ(5,'(a)')ans

c      WRITE (7,1100) IP,IP,IP
      CLOSE(7)
      CLOSE(9)
c      GO TO 320
 610  WRITE (6,6450)
      DO i=1,3
                 CALL place(i)
       CALL finitt
                END DO
 999  STOP
C
C
 1000 FORMAT (8F10.4)
 1100 FORMAT (16I5)
 1200 FORMAT (I4,1X,3F10.4,1X,3E12.4,2(1X,2E12.4))
 1300 FORMAT (I4,1X,6F10.4,1X, E12.4)
 1400 FORMAT (I4,1X,3E12.4)
 1500 FORMAT (I5,9E12.4)
 1600 FORMAT (1H+,42X,'****')
C
```

```
 3000 FORMAT (80A1)
 3100 FORMAT (1H1,80A1/)
 3200 FORMAT (/' TUNNEL GEOMETRY :'/18X,'BREADTH:',3X,F8.3/18X,
    *       'HEIGHT:',4X,F8.3/18X,'IMAGE LAYERS:',I2/)
 3300 FORMAT (/' INPUT BOUNDARY CONDITIONS.  ROOF−FLOOR'//
    *       3X,'I',7X,'X/B',7X,'Y/B',7X,'Z/H',8X,'U,R−F',/)
 3400 FORMAT (/' INPUT BOUNDARY CONDITIONS.  SIDE_WALL'/
    *       3X,'I',7X,'X/B',7X,'Y/B',7X,'Z/H',7X,'U,WALL'/)
 3500 FORMAT (///,13(1H−)/' ITERATION :',I2/1H ,13(1H−)/)
 3600 FORMAT (/' VORTEX STRENGTHS.  (STR = G/(U*SQRT(B*H))'/)
 3700 FORMAT (/3X,'I',7X,'X/B',7X,'Y/B',7X,'Z/H',4X,'SPAN/B',
    *       5X,'SWEEP',5X,'PITCH',10X,'STR'/)
 3800 FORMAT (/' SOURCE STRENGTHS.  (STR = Q/(U*B*H)'/)
 3900 FORMAT (/60X,'TOTAL:',E12.4/)
 4000 FORMAT (/' RE_CALCULATED BOUNDARY CONDITIONS ON ROOF'//
    *       3X,'I',7X,'X/B',7X,'Y/B',7X,'Z/H',
    *       8X,'U_ROOF',5X,'CP_ROOF'/)
 4100 FORMAT (/' RE_CALCULATED BOUNDARY CONDITIONS ON WALL'//
    *       3X,'I',7X,'X/B',7X,'Y/B',7X,'Z/H',
    *       8X,'U_WALL',6X,'W_WALL',5X,'CP_WALL'/)
 4200 FORMAT (/' CENTER_LINE INTERFERENCE VELOCITIES'//
    *       3X,'I',7X,'X/B',7X,'Y/B',7X,'Z/H',
    *       8X,'U_CNTR',6X,'W_CNTR',7X,'U_Q_CL',6X,'U_G_CL',
    *       7X,'W_Q_CL',6X,'W_G_CL'/)
 4300 FORMAT (/' CROSS_EFFECT TERMS:'// 9X,'U_Q_ROOF',3X,
    *                 'U_G_WALL',3X,'W_G_WALL'/)
 4400 FORMAT ('1RUN NUMBER:',I3,'  POINT NUMBER:',I3/1X,38(1H−)//)
C
 6100 FORMAT (1H1, 80A1,'TEST:',I3,',  RUN:',I3,',  POINT:',I2)
 6110 FORMAT ('1MODEL ONLY', 2X,80A1,'TEST:',I3,',  RUN:',I3,
    *       ',  POINT:',I2)
 6120 FORMAT ('1MODEL WITH JET', 2X, 80A1,'TEST:',I3,',  RUN:',I3,
    *       ',  POINT:',I3)
 6400 FORMAT(19X,'ALFA',6X,'PNOT',6X,'QNOT',7X,'CMU',8X,
    1       'CL',8X,'CD',8X,'CM' )
 6410 FORMAT (' MEASURED    '   ,8F10.4)
 6420 FORMAT (' CORRECTED   '   ,8F10.4)
 6430 FORMAT (' RESOLVED    '   ,8F10.4)
 6431 FORMAT (' CLASSICAL   '   ,8F10.4)
 6440 FORMAT (/' WALL POINTS −−− BLOCKAGE : Y/B=',F7.4,', Z/H=',F7.4,
    1        '   LIFT : Y/B=',F7.4,', Z/H=',F7.4/
    1        ' VALUES USED IN CORRECTIONS :  DU/U=',F7.4,', DW/U=',
    1        F7.4,', DELTA−CD=',F7.4)
 6450 FORMAT (///' **** SQRT OF NEGATIVE NUMBER WHILE COMPUTING'
    *              ' U_WALL_EFFECTIVE ****'///)
 6460 FORMAT (43X,'**** : SQRT OF NEGATIVE NUMBER WHILE COMPUTING'
    *       ' U_WALL_EFFECTIVE'/50X,'CROSS_TERM W_G_WALL IGNORED'
    *       ' AT ALL POINTS'/)
C
C
      END


      SUBROUTINE TAPEIO(MATSAV,IU)
C
C−    MATSAV=1 : WRITE ALL REQUIRED MATRICES
C−    MATSAV=2 : READ  ALL REQUIRED MATRICES
C−    MATSAV=3 : READ ONLY THE INFLUENCE COEFFICINT MATRICES
C
      COMMON/IMAT/ UGRF(25,25),UQWL(25,25),UGCL(25,25),UQCL(25,25),
     *           WGCL(25,25),WQCL(25,25),UGWL(25,25),WGWL(25,25),
     *           UQRF(25,25)
      COMMON/LMAT/ AQ(25,25),AG(25,25)
      COMMON/ARR1/ XR(25),YR(25),ZR(25),XL(25),YL(25),ZL(25),
     *           XV(25),YV(25),ZV(25),XS(25),YS(25),ZS(25),
     *           SBV(25),SBS(25),PSIV(25),ALFV(25),PSIS(25),ALFS(25),
     *           CPOWL(25),UCPRF(25),UXQRF(25),UXGWL(25),WXGWL(25),
     *           ULIFT(25),UBLKG(25),GAMA(25),SIGMA(25)
      COMMON/IMAG/ B,H,LAYERS,IRF
      COMMON/SIZE/ NR,NW,NV,NS,XPVOR,XPSRC
C
      PI = 3.141592654
      RAD = PI/180.0
C
      IF(MATSAV .LE. 0) RETURN
      IF(MATSAV .EQ. 1) GO TO 30
C
```

```
      WRITE(6,*)'Open the matrix influence file'
                OPEN(IU,file=*,status = 'old')
      READ (IU,*) LAYERS,IRF,NR,NW,NV,NS
      READ (IU,*) B,H,XPVOR,XPSRC
      DO 10 I = 1,NR
      READ (IU,*) XR(I),YR(I),ZR(I)
          XR(I) = XR(I)*B
          YR(I) = YR(I)*B
          ZR(I) = ZR(I)*H
   10 CONTINUE
      DO 15 I = 1,NW
      READ (IU,*) XL(I),YL(I),ZL(I)
          XL(I) = XL(I)*B
          YL(I) = YL(I)*B
          ZL(I) = ZL(I)*H
   15 CONTINUE
      DO 20 I = 1,NV
      READ (IU,*) XV(I),YV(I),ZV(I),SBV(I),PSIV(I),ALFV(I)
          XV(I) = XV(I)*B
          YV(I) = YV(I)*B
          ZV(I) = ZV(I)*H
          SBV(I) = SBV(I)*B
          PSIV(I) = PSIV(I)*RAD
          ALFV(I) = ALFV(I)*RAD
   20 CONTINUE
      DO 25 I = 1,NS
      READ (IU,*) XS(I),YS(I),ZS(I),SBS(I),PSIS(I),ALFS(I)
          XS(I) = XS(I)*B
          YS(I) = YS(I)*B
          ZS(I) = ZS(I)*H
          SBS(I) = SBS(I)*B
          PSIS(I) = PSIS(I)*RAD
          ALFS(I) = ALFS(I)*RAD
   25 CONTINUE
C
      READ (IU,*) ((UGRF(I,J),I=1,NR),J=1,NV)
      READ (IU,*) ((UGWL(I,J),I=1,NW),J=1,NV)
      READ (IU,*) ((WGWL(I,J),I=1,NW),J=1,NV)
      READ (IU,*) ((UQWL(I,J),I=1,NW),J=1,NS)
      READ (IU,*) ((UQRF(I,J),I=1,NR),J=1,NS)
      READ (IU,*) ((UQCL(I,J),I=1,NW),J=1,NS)
      READ (IU,*) ((WQCL(I,J),I=1,NW),J=1,NS)
      READ (IU,*) ((UGCL(I,J),I=1,NW),J=1,NV)
      READ (IU,*) ((WGCL(I,J),I=1,NW),J=1,NV)
       IF(MATSAV .EQ. 3) RETURN
      READ (IU,*) ((  AG(I,J),I=1,NV),J=1,NV)
      READ (IU,*) ((  AQ(I,J),I=1,NS),J=1,NS)
      RETURN
C
   30 OPEN(IU,file=*'Matrix storage file',status='unknown')
      WRITE (IU,1100) LAYERS,IRF,NR,NW,NV,NS
      WRITE (IU,1000) B,H,XPVOR,XPSRC
      DO 40 I = 1,NR
      WRITE (IU,1000) XR(I)/B,YR(I)/B,ZR(I)/H
   40 CONTINUE
      DO 45 I = 1,NW
      WRITE (IU,1000) XL(I)/B,YL(I)/B,ZL(I)/H
   45 CONTINUE
      DO 50 I = 1,NV
      WRITE (IU,1000) XV(I)/B,YV(I)/B,ZV(I)/H,SBV(I)/B,
     *           PSIV(I)/RAD,ALFV(I)/RAD
   50 CONTINUE
      DO 55 I = 1,NS
      WRITE (IU,1000) XS(I)/B,YS(I)/B,ZS(I)/H,SBS(I)/B,
     *           PSIS(I)/RAD,ALFS(I)/RAD
   55 CONTINUE
C
      WRITE (IU,1000) ((UGRF(I,J),I=1,NR),J=1,NV)
      WRITE (IU,1000) ((UGWL(I,J),I=1,NW),J=1,NV)
      WRITE (IU,1000) ((WGWL(I,J),I=1,NW),J=1,NV)
      WRITE (IU,1000) ((UQWL(I,J),I=1,NW),J=1,NS)
      WRITE (IU,1000) ((UQRF(I,J),I=1,NR),J=1,NS)
      WRITE (IU,1000) ((UQCL(I,J),I=1,NW),J=1,NS)
      WRITE (IU,1000) ((WQCL(I,J),I=1,NW),J=1,NS)
      WRITE (IU,1000) ((UGCL(I,J),I=1,NW),J=1,NV)
      WRITE (IU,1000) ((WGCL(I,J),I=1,NW),J=1,NV)
      WRITE (IU,1000) ((  AG(I,J),I=1,NV),J=1,NV)
```

```
      WRITE (IU,1000) ((  AQ(I,J),I=1,NS),J=1,NS)
      RETURN
C
 1000 FORMAT (5E16.8)
 1100 FORMAT (16I5)
      END

      SUBROUTINE READCP(VCP,NO,IRUN,IP,IRAIL,JRAIL,LIFT, IRF)
      COMMON /BLKB/ ALFU,POU,QOU,CMUU,CLU,CDU,CMU
      DIMENSION VCP(NO), VCPA(30,8), XCP(25), XCP7(30)
C     DATA XCP7 / −35.0, −22.1, −18.3, −14.3, −10.2, −6.2, −2.2, 0.8,
C    1          1.8, 2.8, 3.8, 4.8, 6.8, 9.0, 10.8, 12.8, 15.0, 18.0,
C    1         21.0, 24.0, 30.0, 36.0, 42.0, 48.0, 54.0, 60.0, 4*0.0 /
C     DATA XCP/ −35.0, −20.0, −16.0, −12.0, −8.0, −4.0, 0.0, 3.0, 6.0,
C    *          9.0, 12.0, 15.0, 18.0, 21.0, 25.26, 30.0, 36.0, 42.0,
C    *         48.0, 54.0, 60.0, 4*0.0/
C
C−−−Returns U−Velocity values using CP−values from file 'TAPE10'
C−−−PARAMETERS _
C      VCP    − ARRAY OF VELOCITY VALUES FROM MEASURED CP_S
C      NO    − NUMBER OF CP−VALUES
C      IRUN  − TUNNEL RUN ID NUMBER
C      IP    − POINT NUMBER
C      IRAIL − FIRST  RAIL NUMBER FOR CP−VALUES
C      JRAIL − SECOND RAIL NUMBER FOR CP−VALUES
C      LIFT  − 0 OR 1
C            0 −− VCP =(VCP(JRAIL) + VCP(IRAIL))/2, FOR SIDEWALLS
C            1 −− VCP =(VCP(JRAIL) − VCP(IRAIL)), FOR (ROOF−FLOOR)
C      IRF   − OPTION FLAG. 0=ROOF ONLY, 1=(ROOF−FLOOR)
C
  10  READ (10,500,END=98) NTEST,NRUN,NPNT, NRAILS, NCP
      IF (NO .NE. NCP) THEN
        WRITE (6,610) NO, NCP
        GO TO 99
      END IF
      IF ((1−IRAIL)*(NRAILS−IRAIL) .GT. 0 .OR.
     *    (1−JRAIL)*(NRAILS−JRAIL) .GT. 0     ) THEN
        WRITE (6,620) IRAIL, JRAIL, 1, NRAILS
        GO TO 99
      END IF
      DO 20 J = 1,NRAILS
  20  READ (10,510) (VCPA(I,J), I=1,NCP)
      READ (10,520) ALFU,POU,QOU,CMUU,CLU,CDU,CMU
      IF(NRUN .NE. IRUN) GO TO 10
      IF(NPNT .EQ. IP) GO TO 25
      IF(NPNT .LT. IP) GO TO 10
      IP = 0
      GO TO 45
C
  25  SIGN = 1.0
      DENOM = 2.0
      IF(LIFT .EQ. 0) GO TO 26
      SIGN = −1.0*FLOAT(IRF)
      DENOM = 1.0
  26  IRL = IRAIL
C
  30  DO 40 I = 1,NO
      VCP(I) = (VCPA(I,JRAIL) + SIGN*VCPA(I,IRL  ))/DENOM
      Q = 1.0−VCP(I)
      IF (Q .LT. 0) THEN
        WRITE (6,630) I, VCP(I)
        GO TO 99
      END IF
      VCP(I) = SQRT(Q) − 1.0
  40  CONTINUE
  45  NN = (NO+8)/9
      NN = NN*NRAILS + 2
      DO 50 I = 1,NN
  50  BACKSPACE 10
      RETURN
C
  98  WRITE (6,600) IRUN
  99  IRUN = −IRUN
      RETURN
C
 500  FORMAT (16I5)
 510  FORMAT (2X,9F8.4)
```

```
 520  FORMAT (7F10.2)
 600  FORMAT (///' −−−_ EOF ON TAPE10 _−−−'//' ** NO DATA WITH IDRUN −',
     1  I3,' **'//)
 610  FORMAT (' −−: ERROR IN ROUTINE "READCP" :−−'/
     *       '     MISMATCH BETWEEN AVAILABLE AND REQUESTED NO OF',
     *       ' CP−MEASUREMENT POINTS'/'     N−REQ =',I4, '  N−AVL =',
     *       I4/)
 620  FORMAT (' −−: ERROR IN ROUTINE "READCP" :−−'/
     *       '     RAIL NUMBERS OUT OF RANGE'/'     RAILS REQUESTED =',
     *       2I3/'     RAILS AVAILABLE = ', 2I3/)
 630  FORMAT (' −−: ERROR IN ROUTINE "READCP" :−−'/
     *       '     SQRT(1−CP) IS NEGATIVE'/'     I =', I3, '(1−CP) =',
     *       E13.5)
      END

      SUBROUTINE INTER(XD,FD,XI,FI,MAX,IDO)
C
C     GIVEN A SET OF VALUES IN THE ARRAY "FD" AT DATA POINTS "XD",
C     THIS SUBROUTINE INTERPOLATES THEM AT POINTS "XI"
C     THE INTERPOLATED VALUES ARE PLACED IN ARRAY "FI"
C     MAX  : NO. OF POINTS IN "XD" FOR WHICH "FD" IS DEFINED
C     IDO  : NO. OF POINTS IN "XI" FOR WHICH "FI" IS DESIRED.
C
      DIMENSION XD(1),FD(1),XI(1),FI(1)
C
      DO 40 I = 1,IDO
      IF(XI(I) .GT. XD(1)) GO TO 10
      J = 2
      GO TO 30
  10  DO 20 J = 1,MAX
      IF(XD(J) .GE. XI(I)) GO TO 30
  20  CONTINUE
      J = MAX
  30  JP = J
      JM = J−1
      FI(I) = FD(JM) + (FD(JP) − FD(JM))*(XI(I)−XD(JM))/
     1       (XD(JP)−XD(JM))
  40  CONTINUE
      RETURN
      END

      SUBROUTINE OUTPUT(NO,NS, XM,SPAN, LIFT, VCP,XCP,SIGMA,
     *               DC,DW, ICORR, JET)
C
C −−− OUTPUT OF LIFT AND BLOCKAGE CORRECTIONS
C −−− GENERATION OF LINE PRINTER PLOTS

      COMMON/IMAG/ B,H,LAYERS,IRF
      DIMENSION VCP(NO),XCP(NO),SIGMA(NS),DC(25,3),DW(25,3)
      DIMENSION XM(NS),NP(61)
C
      SPB = SPAN/B*2.0
C −−− SET UP HEADERS AND PLOT LIMITS
      L=2*LIFT+1
      IF(LIFT.EQ.1) GO TO 1
      VALU = B*H
      WRITE(6,10)
      GO TO 2
   1  WRITE(6,11)
      VALU = SQRT(B*H)
   2  DO n=1,no
                 x=xcp(n)/b
                         WRITE(6,'(2X,6F8.4)')x,vcp(n),(dw(n,j),j=1,3),dc(n,l)
               END DO
      RETURN

  10  FORMAT(/6X,'POSN',3X,'INPUT',3(4X,'WALL'),5X,'C/L',4X,
     *       'BLOCKAGE CORRECTION'/
     *       7X,'X/B',2(4X,'DU/U'),4X,'DV/U',4X,'DW/U',4X,'DU/U'  I,
     *       5X,'X/B',3X,' S' )
  11  FORMAT(/6X,'POSN',3X,'INPUT',3(4X,'WALL'),5X,'C/L',6X,
     *       'LIFT CORRECTION'/
     *       7X,'X/B',2(4X,'DU/U'),4X,'DV/U',4X,'DW/U',4X,'DU/U'  I,
     *       5X,'X/B',3X,' S' )
      END
```

```
      SUBROUTINE OUT(A,ID,JD,MM,NN,TITL,NCAR)
C
C-  PRINTS OUT A TWO DIMENSIONAL ARRAY A(I,J), IN THE RANGE I=1 TO MM
C-  AND J=1 TO NN.  ID AND JD ARE THE ARRAY'S DIMENSION LIMITS.
C-  "TITL" is a title with "NCAR" number of characters.
C
      DIMENSION A(ID,JD)
      CHARACTER TITL(79)
      MLO = 1
      MHI = (NCAR+3)/4
      WRITE (6,600) (TITL(M),M=1,NCAR)
C
   10 MHI = MLO + 8
      IF(MHI .GT. MM) MHI = MM
      WRITE (6,610) (M,M=MLO,MHI)
      WRITE (6,612)
C
      DO 20 N = 1,NN
   20 WRITE (6,620) N,(A(M,N),M=MLO,MHI)
      MLO = MLO + 9
      IF(MLO .LE. MM) GO TO 10
C
      MLO = 1
      NLO = 1
      MHI = 1
      NHI = 1
      AHI = A(1,1)
      ALO = A(1,1)
C
      DO 50 M = 1,MM
      DO 50 N = 1,NN
      IF(A(M,N) .LE. AHI) GO TO 40
      AHI = A(M,N)
      MHI = M
      NHI = N
   40 IF(A(M,N) .GE. ALO) GO TO 50
      ALO = A(M,N)
      MLO = M
      NLO = N
   50 CONTINUE
C
      WRITE (6,630) MLO,NLO,ALO,MHI,NHI,AHI
      RETURN
C
  600 FORMAT (1H1,79A1)
  610 FORMAT (/8X,'M = ',I2,9I13)
  612 FORMAT (3X,1HN/)
  620 FORMAT (I4,1X,9E13.5)
  630 FORMAT (//' MINIMUM AND MAXIMUM :'/26X,1HM,3X,1HN,5X,5HVALUE,
     1          /(23X,2I4,E13.5))
      END

      SUBROUTINE JETADD(NW,NR, VWALL,VROOF,VCNTR, ITEST,IRUN,IP)
C-
C-  Subroutine to add the effects of jet/s.
C-
      DIMENSION VCL(3), VWALL(25,3),VROOF(25,3),VCNTR(25,3)
      NN = MAX(NW,NR)
      IREW = 0
C
   10 READ (11,1000,END=50) IT,IR,IPT
      IF(IT.NE.ITEST .OR. IR.NE.IRUN .OR. IPT.NE.IP) GO TO 40
      DO 30 I = 1,NN
      READ (11,1100) J,XOB, UWL, URF, (VCL(K),K=1,3)
      VWALL(I,1) = VWALL(I,1) + UWL
      VROOF(I,1) = VROOF(I,1) + URF
      DO 20 K = 1,3
   20 VCNTR(I,K) = VCNTR(I,K) + VCL(K)
   30 CONTINUE
C
      RETURN
C
   40 DO 45 I = 1,NN
   45 READ (11,1100)
      GO TO 10
C
   50 IF(IREW .NE. 0) GO TO 60
```

```
      IREW = 1
      REWIND 11
      GO TO 10
C
   60  WRITE (6,600) ITEST,IRUN,IP
      RETURN
C
  600  FORMAT (//' *** NO JET_EFFECT DATA FOUND IN TAPE11 FOR '
     *         'ITEST/IRUN/IPT = ',3(I3,1H/), '  ***'//)
 1000 FORMAT (16I5)
 1100 FORMAT (I5,7E15.8)
C
      END
```

```
      SUBROUTINE PIVOT(X,Y,Z, XPIV,SP,PSI,ALFA, X1,Y1,Z1,
     *                 X2,Y2,Z2)
C
C  DETERMINES THE END POINTS OF A LINE OF LENGTH "SP",
C  WHEN IT IS PITCHED AND YAWED.
C
        TANP = TAN(PSI)
        SINA = SIN(ALFA)
        COSA = COS(ALFA)
        X1 = X + XPIV*(1.0−COSA)
        Y1 = Y
        Z1 = Z + XPIV*SINA
        X2 = X + XPIV*(1.0−COSA) + SP*TANP*COSA
        Y2 = Y + SP
        Z2 = Z − (SP*TANP−XPIV)*SINA
        RETURN
        END

      SUBROUTINE INFLU(X1,Y1,Z1, X2,Y2,Z2, STRENT,ITYP,MINIT,ITRUNC,
     1                 XP,YP,ZP, DU,DV,DW)
C
C PARAMETERS:
C       X1,Y1,Z1 : Line_singularity End Point Co_ordinates, First Point
C       X2,Y2,Z2 : Line_singularity End Point Co_ordinates, Second Point
C       STRENT   : Singularity Strength (Total for ITYP=2,3
C                                  per unit length for ITYP=2)
C       ITYP     : Type Of Singularity... 1=Source/Sink,
C                                   2=Horse_Shoe Vortex,
C                                     (GAMA +ve Y1 TO Y2)
C                                   3=Doublet.
C       MINIT    : Initial Image Layer (=0 To Include Model In Tunnel)
C       XP,YP,ZP : Calculation Point
C       DU,DV,DW : Incremental Velocity Components, RETURNed.
C
      COMMON /IMAG/ B,H,LAYERS,IRF
      COMMON /DIRC/ DCXO,DCYO,DCZO
      IBUG = 0
      IS = −1
      DU = 0.0
      DV = 0.0
      DW = 0.0
      DCX = DCXO
      XINF = 10000.0*B
      STPERL = STRENT
      IF(ITYP .EQ. 2) GO TO 5
      ELS = SQRT( (X2−X1)**2 + (Y2−Y1)**2 + (Z2−Z1)**2 )
      STPERL = STRENT/ELS
C
    5 LMAX = LAYERS + 1
      DO 30 LDO = 1,LMAX
      FAC = 1.0
      L = LDO−1
      IF(L .LT. MINIT) GO TO 30
      IF(LDO.EQ.LMAX .AND. L.GT.0) FAC = 0.5 + 1.0/FLOAT(4*L)
      IF(ITYP .NE. 2) FAC = 1.0
C
      MNDO = L*2 + 1
      NOT = L + 1
      MN1 = MNDO − 1
C
      DO 20 MDO = 1,MNDO
      M = MDO − NOT
      REVM = (IS)**(IABS(M))
      YONE = FLOAT(M)*B + REVM*Y1
      YTWO = FLOAT(M)*B + REVM*Y2
C
      NINC = MN1
      IF(MDO.EQ.1 .OR. MDO.EQ.MNDO) NINC = 1
      DO 20 NDO = 1,MNDO,NINC
      N = NDO − NOT
      REVN = (IS)**(IABS(N))
      X2IM = X2
      X1IM = X1
      Y1IM = YONE
      Y2IM = YTWO
      Z1IM = FLOAT(N)*H + REVN*Z1
      Z2IM = FLOAT(N)*H + REVN*Z2
```

```
      IF(REVM*REVN .GT. 0) GO TO 10
C
      TEMP = Y2IM
      Y2IM = Y1IM
      Y1IM = TEMP
      TEMP = Z2IM
      Z2IM = Z1IM
      Z1IM = TEMP
      TEMP = X2IM
      X2IM = X1IM
      X1IM = TEMP
C
   10 DCY = DCYO*REVN
      DCZ = DCZO*REVM
C
      IF(ITYP−2) 11,12,13
   11 CALL LNSCGN(X1IM,Y1IM,Z1IM, X2IM,Y2IM,Z2IM, STPERL,
     1          XP,YP,ZP, DELU,DELV,DELW)
      GO TO 16
   12 CALL LNVXGN(X1IM,Y1IM,Z1IM, XINF,Y1IM,Z1IM, STPERL,
     *          XP,YP,ZP, DUT1,DVT1,DWT1)
      CALL LNVXGN(X2IM,Y2IM,Z2IM, X1IM,Y1IM,Z1IM, STPERL,
     1          XP,YP,ZP, DUBN,DVBN,DWBN )
      CALL LNVXGN(XINF,Y2IM,Z2IM, X2IM,Y2IM,Z2IM, STPERL,
     *          XP,YP,ZP, DUT2,DVT2,DWT2)
      DELU = DUT1 + DUBN + DUT2
      DELV = DVT1 + DVBN + DVT2
      DELW = DWT1 + DWBN + DWT2
      GO TO 16
   13 CALL LNDBGN(X1IM,Y1IM,Z1IM, X2IM,Y2IM,Z2IM, STPERL,
     1          DCX,DCY,DCZ, XP,YP,ZP, DELU,DELV,DELW,
     2          VMUR, CXR,CYR,CZR)
   16 DU = DU + FAC*DELU
      DV = DV + FAC*DELV
      DW = DW + FAC*DELW
      IF(IBUG .NE. 0) WRITE (6,600) M,N,XP,YP,ZP,DELU,DELV,DELW,DU,DV,DW
  600 FORMAT (2I4,3X,3F8.3,2(2X,3E14.6))
   20 CONTINUE
   30 CONTINUE
C
C−  ADD TRUNCATION SHEET FOR SOURCE/SINK
C
      IF(ITRUNC .EQ. 0) RETURN
      IF(LAYERS.EQ.0 .OR. ITYP.NE.1) RETURN
      QBH = STRENT/(2.0*B*H)
      PI = 3.141592654
      XM = (X1+X2)/2.0
      XX = XP − XM
      SIGNX = SIGN(1.0, XX)
      IF(ABS(XX) .LT. 0.1E−06) SIGNX = 0.0
      YR = LAYERS*B + B/2.
      ZU = LAYERS*H + H/2.
      YL = −YR
      ZL = −ZU
      TF =  TFUNC(XX,YP,ZP, YL,ZL, YR,ZU)
      CALL LFUNCS(XX,YP,ZP, YL,ZL, YR,ZU, FL1,FL2)
      DU = DU + QBH + QBH*(SIGNX−TF/(2.0*PI))
      DV = DV + QBH*FL1/(2.0*PI)
      DW = DW + QBH*FL2/(2.0*PI)
      RETURN
      END

      SUBROUTINE LFUNCS (XX,Y,Z,Y1,Z1,Y2,Z2,L1F,L2F)
C −−−
C −−− RETURNS L1 AND L2 FUNCTIONS FOR CALCULATION OF V AND W VELOCITIES
C −−− RESPECTIVELY AT POINT XX,Y,Z DUE TO THE EDGES Y1 Z1 Y2 Z2 OF A
C −−− RECTANGULAR HOLE IN AN INFINITE SHEET SOURCE AT XX=0
C −−−
      REAL N1,N2,N3,N4,L1F,L2F
      DY=Y2−Y1
      DZ=Z2−Z1
      XQ=XX*XX
      Y1Q=(Y−Y1)**2
      Y2Q=(Y−Y2)**2
      Z1Q=(Z−Z1)**2
      Z2Q=(Z−Z2)**2
      R1=SQRT(XQ+Y1Q+Z2Q)
```

```
      R2=SQRT(XQ+Y2Q+Z2Q)
      R3=SQRT(XQ+Y2Q+Z1Q)
      R4=SQRT(XQ+Y1Q+Z1Q)
      N1=R4+R1−DZ
      N2=R2+R3+DZ
      N3=R3+R4−DY
      N4=R1+R2+DY
      D1=R4+R1+DZ
      D2=R2+R3−DZ
      D3=R3+R4+DY
      D4=R1+R2−DY
      L1F=ALOG(N1*N2/(D1*D2))
      L2F=ALOG(N3*N4/(D3*D4))
      RETURN
      END
      FUNCTION TFUNC (XX,Y,Z,Y1,Z1,Y2,Z2)
C −−−
C −−− RETURNS FUNCTION−T FOR COMPUTATION OF U−VELOCITY AT POINT XX,Y,Z
C −−− DUE TO A FINITE RECTANGALAR SHEET S6URCE Y1 Z1 Y2 Z2 AT XX=0
C −−−
      DIMENSION SIGN(4),YY(4),ZZ(4)
      SUM=0.
      IF(ABS(XX).LT..00000001) GO TO 2
      SIGN(1)=1.
      SIGN(2)=−1.
      SIGN(3)=−1.
      SIGN(4)=1.
      YY(1)=Y−Y1
      YY(2)=YY(1)
      YY(3)=Y−Y2
      YY(4)=YY(3)
      ZZ(1)=Z−Z1
      ZZ(2)=Z−Z2
      ZZ(3)=ZZ(1)
      ZZ(4)=ZZ(2)
      DO 1 J=1,4
      XJ=YY(J)*ZZ(J)/(XX*SQRT(XX*XX+YY(J)*YY(J)+ZZ(J)*ZZ(J)))
      PART=SIGN(J)*ATAN(XJ)
    1 SUM=SUM+PART
    2 TFUNC=SUM
      RETURN
      END
      SUBROUTINE LNDBEQ(VMU2,XC,YC,ZC,VL1,DU,DV,DW,IT)
C−  LINE DOUBLET EQUATION
      PI=3.141592654
C THE VALUE OF 'IT' TAKES THE VALUE OF '2' FORTYPE (1,2) AND '3' FOR
C TYPE (1,3).
      CONST=VMU2/(4.*PI)
      IF(IT−2)1,1,2
    2 TEMP=ZC
      ZC=YC
      YC=TEMP
    1 VL12=VL1/2.
      XCPL=XC+VL12
      XCML=XC−VL12
      TMPP=YC**2+ZC**2
      TMPN=YC**2−ZC**2
      DN1=XCML**2+TMPP
      DP1=XCPL**2+TMPP
      TEMP=(.01*VL1)**2
      IF(TMPP−TEMP)11,11,13
   11 TEMP=ABS(XC)−VL12
      IF(TEMP)18,18,5
   18 DU=0.
      DV=0.
      DW=0.
      GO TO 3
    5 DNOMP=SQRT(DP1**3)
      DNOMN=SQRT(DN1**3)
      DU=CONST*YC*(1./DNOMN−1./DNOMP)
      DV=.5*CONST*(1./XCPL**2−1./XCML**2)
      DW=0.
      GO TO 3
   13 DNOMP=SQRT(DP1**3)
      DNOMN=SQRT(DN1**3)
      DU=CONST*YC*(1./DNOMN−1./DNOMP)
      DV=CONST*((XCPL*(TMPN*DP1+TMPP*YC**2)/DNOMP−XCML*(TMPN*DN1+TMPP*YC
```

```
     1**2)/DNOMN)/TMPP**2)
      DW=CONST*YC*ZC*(XCPL*(2.*DP1+TMPP)/DNOMP−XCML*(2.*DN1+TMPP)/DNOMN)
     1/TMPP**2
      IF(IT−2)3,3,4
    4 TEMP=DW
      DW=DV
      DV=TEMP
    3 CONTINUE
      RETURN
      END

      SUBROUTINE LNDBGN(XA,YA,ZA,XB,YB,ZB,VMU,CX,CY,CZ,XP,YP,ZP,DU,DV,DW
     1,VMUR,CXR,CYR,CZR)
C−
C− GENERATES DU,DV,DW VELOCITIES DUE TO A LINE DOUBLET
C−
C DATA TO BE SUPPLIED.**************************************************
C −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
C XA,YA,ZA,XB,YB,ZB, ARE THE END COORDINATES OF THE LINE DOUBLET.
C VMU IS THE COUPLE OF THE DOUBLET.
C CX,CY,CZ ARE THE DIRECTION COSINES OF THE COUPLE VECTOR.
C CX=X/R     CY=Y/R     AND     CZ=Z/R. R=SQRT(X**2+Y**2+Z**2)
C VL IS THE LENGTH OF THE LINE DOUBLET.
C XP,YP,ZP ARE THE COORDINATES OF ANY POINT IN SPACE.
C VALUES CALCULATED BY THE PROGRAM**************************************
C−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
C DU,DV,DW ARE THE VELOCITY COMPONENTS AT THE POINT 'P' IN THE (X,Y,Z)
C SYSTEM
C VMUR IS THE RESULTANT COMPONENT OF THE COUPLE OF THE DOUBLET.
C THIS COMPONENT IS NORMAL TO THE AXIS OF THE DOUBLET.
C CXR,CYR,CZR ARE THE DIRECTION COSINES OF THE COUPLE VECTOR USED IN
C CALCULATING DU,DV,DW.
C NOTE.  THE COMPONENT OF THE COUPLE VECTOR ALONG THE AXIS OF THE
C DOUBLET WAS NOT INCLUDED IN DETERMINING DU,DV,DW. THIS ELIMINATED
C POINT SOURCE AND POINT SINK EFFECTS IN THE MODELDUE TO THE LINE
C DOUBLET. **************************************************
C*****************************************************************
      TEMP=(XA+XB)/2.
      XAO=XA−TEMP
      XBO=XB−TEMP
      XPO=XP−TEMP
      TEMP=(YA+YB)/2.
      YAO=YA−TEMP
      YBO=YB−TEMP
      YPO=YP−TEMP
      TEMP=(ZA+ZB)/2.
      ZAO=ZA−TEMP
      ZBO=ZB−TEMP
      ZPO=ZP−TEMP
      IC=1
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,XPO,YPO,ZPO,XPCG,YPCG,ZPCG,IC)
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,CX,CY,CZ,CXG,CYG,CZG,IC)
      VMU2=VMU*CYG
      VMU3=VMU*CZG
      VMUR=SQRT(VMU2**2+VMU3**2)
      VL1=SQRT((XA−XB)**2+(YA−YB)**2+(ZA−ZB)**2)
      IT=2
      CALL LNDBEQ(VMU2,XPCG,YPCG,ZPCG,VL1,DU2,DV2,DW2,IT)
      IT=3
      CALL LNDBEQ(VMU3,XPCG,YPCG,ZPCG,VL1,DU3,DV3,DW3,IT)
      DUG=DU2+DU3
      DVG=DV2+DV3
      DWG=DW2+DW3
      CXG=0.
      TEMP=SQRT(CYG*CYG+CZG*CZG)
      CYG=CYG/TEMP
      CZG=CZG/TEMP
      IC=2
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,DU,DV,DW,DUG,DVG,DWG,IC)
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,CXR,CYR,CZR,CXG,CYG,CZG,IC)
      CONTINUE
      RETURN
      END

      SUBROUTINE LNVXGN(XA,YA,ZA,XB,YB,ZB,CIRC,XP,YP,ZP,DU,DV,DW)
C GENERATES INCREMENTAL VELOCITY DUE TO A LINE VORTEX.
      TEMP=(XA+XB)/2.
```

```
      XAO=XA−TEMP
      XBO=XB−TEMP
      XPO=XP−TEMP
      TEMP=(YA+YB)/2.
      YAO=YA−TEMP
      YBO=YB−TEMP
      YPO=YP−TEMP
      TEMP=(ZA+ZB)/2.
      ZAO=ZA−TEMP
      ZBO=ZB−TEMP
      ZPO=ZP−TEMP
      IC=1
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,XPO,YPO,ZPO,XPCG,YPCG,ZPCG,IC)
      VL1=SQRT((XA−XB)**2+(YA−YB)**2+(ZA−ZB)**2)
      CALL LNVXEQ(CIRC,XPCG,YPCG,ZPCG,VL1,DUG,DVG,DWG)
      IC=2
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,DU,DV,DW,DUG,DVG,DWG,IC)
      CONTINUE
      RETURN
      END

      SUBROUTINE LNVXEQ(CIRC,XPG,YPG,ZPG,VL,DUG,DVG,DWG)
C− LINE VORTEX EQUATION
      PI=3.141592654
      DUG=0.
      YZ2=YPG*YPG+ZPG*ZPG
      VL2=VL/2.
      XPL2=XPG+VL2
      XML2=XPG−VL2
      DN=XML2**2+YZ2
      DP=XPL2**2+YZ2
      RDN=SQRT(DN)
      RDP=SQRT(DP)
      CONST=.25*CIRC/PI
C     TEMP=.01*VL
      TEMP=0.000001
      IF(YZ2−TEMP)2,2,4
    2 TEMP=ABS(XPG)−VL2
      IF(TEMP)8,8,5
    8 DVG=0.
      DWG=0.
      GO TO 7
    5 TEMP=.5*CONST*(1./XML2**2−1./XPL2**2)
      DVG=−ZPG*TEMP
      DWG=YPG*TEMP
      GO TO 7
    4 TEMP=CONST*(XML2/RDN−XPL2/RDP)/YZ2
      DVG=ZPG*TEMP
      DWG=−YPG*TEMP
    7 CONTINUE
      RETURN
      END

      SUBROUTINE LNSCGN(XA,YA,ZA,XB,YB,ZB,VM,XP,YP,ZP,DU,DV,DW)
C− GENERATES INCREMENTAL VELOCITIES DUE TO LINE SOURCE
      TEMP=(XA+XB)/2.
      XAO=XA−TEMP
      XBO=XB−TEMP
      XPO=XP−TEMP
      TEMP=(YA+YB)/2.
      YAO=YA−TEMP
      YBO=YB−TEMP
      YPO=YP−TEMP
      TEMP=(ZA+ZB)/2.
      ZAO=ZA−TEMP
      ZBO=ZB−TEMP
      ZPO=ZP−TEMP
      IC=1
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,XPO,YPO,ZPO,XPCG,YPCG,ZPCG,IC)
      VL1=SQRT((XA−XB)**2+(YA−YB)**2+(ZA−ZB)**2)
      CALL LNSCEQ(VM,XPCG,YPCG,ZPCG,VL1,DUG,DVG,DWG)
      IC=2
      CALL CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,DU,DV,DW,DUG,DVG,DWG,IC)
      CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LNSCEQ(VM,XPG,YPG,ZPG,VL,DUG,DVG,DWG)
C− LINE SOURCE EQUATION
      PI=3.141592654
      YZ2=YPG*YPG+ZPG*ZPG
      VL2=VL/2.
      XPL2=XPG+VL2
      XML2=XPG−VL2
      DN=XML2**2+YZ2
      DP=XPL2**2+YZ2
      RDN=SQRT(DN)
      RDP=SQRT(DP)
      CONST=.25*VM/PI
      TEMP=.01*VL
      IF(YZ2−TEMP)2,2,4
    2 TEMP=ABS(XPG)−VL2
      IF(TEMP)8,8,5
    8 DUG=0.
      DVG=0.
      DWG=0.
      GO TO 7
    5 DUG=CONST*(1./RDN−1./RDP)
      TEMP=.5*CONST*(1./XML2**2−1./XPL2**2)
      DVG=YPG*TEMP
      DWG=ZPG*TEMP
      GO TO 7
    4 DUG=CONST*(1./RDN−1./RDP)
      TEMP=CONST*(XML2/RDN−XPL2/RDP)/YZ2
      DVG=−YPG*TEMP
      DWG=−ZPG*TEMP
    7 CONTINUE
      RETURN
      END

      SUBROUTINE CRDTEG(XAO,YAO,ZAO,XBO,YBO,ZBO,XPO,YPO,ZPO,EXIP,ETAP,ZT
     1AP,IC)
C COORDINATE TRANSFORMATION 'ENGLISH TO GREEK'.IC=1
C COORDINATE TRANSFORMATION 'GREEK TO ENGLISH'.IC=2
C THE ORIGIN OF THE COORDINATE SYSTEM WILL BE AT THE MID−POINT OF A−B.
C SUBROUTINE TRANSFORMS (X,Y,Z) SET OF AXES TO (EXI,ETA,ZTA)AXES WITH
C THE 'EXI' AXIS PARALLEL TO THE CENTRE LINE OF THE DOUBLET.
C THE ENDS OF THE LINE DOUBLET ARE 'A' AND'B'.
C 'P' IS A GENERAL POINT IN SPACE.
C NOTE SYSTEM SET UP FOR POSITIVE_WINDING NUMBERS  FOR ROTATION OF  AXES
C EPSI AND ALFA ARE THE ANGLES OF ROTATION OF THE ETA OR (Y) AND ZTA OR
C (Z) AXES RESPECTIVELY.
      TEMP=SQRT((XAO−XBO)**2+(YAO−YBO)**2)
      IF(TEMP)4,4,5
    5 CEPSI=(XAO−XBO)/TEMP
      SEPSI=(YAO−YBO)/TEMP
      XAP=XAO*CEPSI+YAO*SEPSI
      XBP=XBO*CEPSI+YBO*SEPSI
      GO TO 6
    4 CEPSI=1.
      SEPSI=0.
      XAP=XAO
      XBP=XBO
    6 TEMP=SQRT((XAP−XBP)**2+(ZAO−ZBO)**2)
      IF(TEMP)7,7,8
    8 SALFA=(ZBO−ZAO)/TEMP
      CALFA=(XAP−XBP)/TEMP
      GO TO 9
    7 CALFA=1.
      SALFA=0.
    9 IF(IC−1)1,1,2
    1 XPP=XPO*CEPSI+YPO*SEPSI
      YPP=YPO*CEPSI−XPO*SEPSI
      ZPP=ZPO
      EXIP=XPP*CALFA−ZPP*SALFA
      ETAP=YPP
      ZTAP=XPP*SALFA+ZPP*CALFA
      GO TO 3
    2 XPP=EXIP*CALFA+ZTAP*SALFA
      YPP=ETAP
      ZPP=ZTAP*CALFA−EXIP*SALFA
      XPO=XPP*CEPSI−YPP*SEPSI
      YPO=XPP*SEPSI+YPP*CEPSI
      ZPO=ZPP
```

```
    3 CONTINUE
      RETURN
      END

      SUBROUTINE GJRV (A,N,NL,EPSIL,IERR)
C     . . . . . .
C     .                                                    .
C     . MATRIX INVERSION ROUTINE                           .
C     .                                                    .
C     . CALLING SEQUENCE.....                              .
C     . CALL        N,NL,EPSIL,IERR)                       .
C     . A IS THE INPUT ARRAY WHICH WILL BE DESTROYED, N IS THE RANK  .
C     . OF A, NL IS THE ROW DIMENSION OF A, EPSIL IS THE TEST       .
C     . VALUE FOR THE PIVOT POINT SINGULARITY CHECK,                .
C     . IERR IS NONZERO IF THE MATRIX IS SINGULAR                   .
C     . IF THE MATRIX IS NONSINGULAR, A CONTAINS A−INVERSE          .
C     .                                                    .
C     . . . . . C . . . . . .
      DIMENSION A(1),B(96),C(96),IP(96),IQ(96)
      IERR=0
      DO 160 K=1,N
      PIVOT=0.
C
C     GET LARGEST ELEMENT IN MATRIX  PLACE IN PIVOT
C
      DO 30 I=K,N
      DO 20 J=K,N
      IDEX=(J−1)*NL+I
      IF (ABS(A(IDEX))−ABS(PIVOT)) 20,20,10
   10 CONTINUE
      PIVOT=A(IDEX)
      IP(K)=I
      IQ(K)=J
   20 CONTINUE
   30 CONTINUE
      IF (ABS(PIVOT)−EPSIL) 230,230,40
   40 CONTINUE
      IF (IP(K)−K) 50,70,50
   50 CONTINUE
C
C     SWAP ROWS
C
      DO 60 J=1,N
      IPX=IP(K)
      IDEX=(J−1)*NL+IPX
      KDEX=(J−1)*NL+K
      Z=A(IDEX)
      A(IDEX)=A(KDEX)
      A(KDEX)=Z
   60 CONTINUE
   70 CONTINUE
      IF (IQ(K)−K) 80,100,80
   80 CONTINUE
C
C     SWAP COLUMNS
C
      DO 90 I=1,N
      IPX=IQ(K)
      IDEX=(IPX−1)*NL+I
      KDEX=(K−1)*NL+I
      Z=A(IDEX)
      A(IDEX)=A(KDEX)
      A(KDEX)=Z
   90 CONTINUE
  100 CONTINUE
      DO 140 J=1,N
      KDEX=(J−1)*NL+K
      JDEX=(K−1)*NL+J
      IF (J−K) 120,110,120
  110 CONTINUE
      B(J)=1./PIVOT
      C(J)=1.
      GO TO 130
  120 CONTINUE
      B(J)=−A(KDEX)/PIVOT
      C(J)=A(JDEX)
  130 CONTINUE
```

```
      A(KDEX)=0.
      A(JDEX)=0.
  140 CONTINUE
      DO 150 I=1,N
      DO 150 J=1,N
      IDEX=(J-1)*NL+I
      A(IDEX)=A(IDEX)+C(I)*B(J)
  150 CONTINUE
  160 CONTINUE
      DO 220 KP=1,N
      K=N+1-KP
      IF (IP(K)-K) 170,190,170
  170 CONTINUE
      DO 180 I=1,N
      IPX=IP(K)
      IDEX=(IPX-1)*NL+I
      KDEX=(K-1)*NL+I
      Z=A(IDEX)
      A(IDEX)=A(KDEX)
      A(KDEX)=Z
  180 CONTINUE
  190 CONTINUE
      IF (IQ(K)-K) 200,220,200
  200 CONTINUE
      DO 210 J=1,N
      IPX=IQ(K)
      IDEX=(J-1)*NL+IPX
      KDEX=(J-1)*NL+K
      Z=A(IDEX)
      A(IDEX)=A(KDEX)
      A(KDEX)=Z
  210 CONTINUE
  220 CONTINUE
      GO TO 240
  230 CONTINUE
      IERR=-1
  240 CONTINUE
      RETURN
      END
```

Appendix B

The following pages contain Hackett’s program, as converted to C by Luther Jenkins and myself for use at the Basic Aerodynamics Research Tunnel (BART) at NASA LaRC.

```
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

  Wall Correction Software: based on Fortran code LSQITER that was written by Lockheed−      GA and modified by Tony Washburn
for use at the Basic Aerodynamics Research Tunnel.
        Converted to C by Luther N. Jenkins (4/96) and James L. Buckingham, Jr. (2001)

  Iterative solution accounting for cross−effects of vortices and sources, Least−Squares   approach.

  Notable Changes

  1.  Some of the documentation and the variable names have been changed for clarity.  A
  cross−reference has been included to facilitate troubleshooting and debugging using the
  original code.  A listing of the original code can be found in NASA CR−166, 186−187.

  Nomenclature

  layers          − number of image layers
  irf                     − float for inclusion of floor signature
  nr                      − number of roof pressures
  nw                      − number of wall pressures
  nv                      − number of vortex singularities
  ns                      − number of source/sink singularities
  TunnelSpan      − tunnel breadth
  TunnelHeight    − tunnel height
  xpvor                   − pivot point for pitching swept vortex
  xpsrc                   − pivot point for swept source/sink

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "Subroutines.h"

/* GLOBAL VARIABLES AND CONSTANTS */

#define PI 3.142592654
#define MAX_PORTS 25

/* Influence Matrix Arrays */
float ug_ceiling[MAX_PORTS][MAX_PORTS], uq_ceiling[MAX_PORTS][MAX_PORTS];
float wg_ceiling[MAX_PORTS][MAX_PORTS], wq_ceiling[MAX_PORTS][MAX_PORTS];
float ug_wall[MAX_PORTS][MAX_PORTS], wg_wall[MAX_PORTS][MAX_PORTS];
float uq_roof[MAX_PORTS][MAX_PORTS];
//Necessary additions?
float ug_roof[MAX_PORTS][MAX_PORTS], uq_wall[MAX_PORTS][MAX_PORTS];

/* Inverse Matrix Arrays */
float aq[MAX_PORTS][MAX_PORTS], ag[MAX_PORTS][MAX_PORTS];

/* Coordinate and Aerodynamic Parameter Arrays */
float x_roof[MAX_PORTS], y_roof[MAX_PORTS], z_roof[MAX_PORTS];
float x_wall[MAX_PORTS], y_wall[MAX_PORTS], z_wall[MAX_PORTS];
float x_vortex[MAX_PORTS], y_vortex[MAX_PORTS], z_vortex[MAX_PORTS];
float sb_vortex[MAX_PORTS], psi_vortex[MAX_PORTS], alpha_vortex[MAX_PORTS];
float x_source[MAX_PORTS], y_source[MAX_PORTS], z_source[MAX_PORTS];
float sb_source[MAX_PORTS], psi_source[MAX_PORTS], alpha_source[MAX_PORTS];
float cpowl[MAX_PORTS], upc_wall[MAX_PORTS], ucp_roof[MAX_PORTS], uxq_roof[MAX_PORTS];
float uxg_wall[MAX_PORTS], wxg_wall[MAX_PORTS], ulift[MAX_PORTS], ublockage[MAX_PORTS];
float gamma[MAX_PORTS], sigma[MAX_PORTS];

float xarray[30], yarray[30], yarray2[30], yarray3[30];
float cp_empty_wall[25], cp_empty_roof[25];

float sarea, awb, bwb, ec1, ec2, ec3, xbcor;
float TunnelSpan, TunnelHeight, xp_vortex, xp_source, xpiv;
int   layers, irf;
```

```
/* BEGIN MAIN SOURCE CODE */

main ()
{
        /* Variables and constants */

        FILE *geom_file, *AeroCoeffFile, *MatrixFile, *ExpDataFile;

        char ans;
        char geometry_file[13], tape7[13], tape9[13], pfix[3], title[80];
        char LoadCoefficientFile[30], InfluenceMatrixFile[30], ExperimentalData[30];

        int i, j, k;
        int ipoint, icor, ierr;
        int itermax, matprt, SaveMatrix, iprt, krosg, krosq, icorr, jetefct;
        extern int layers, irf;
        int         nr, nw, nv, ns;
        int InitialImageLayer, ipdo, ipmin, ipmax;
        int itest, irun, ifloor, iroof, iwall1, iwall2, ip;

        /* Influence Matrix Arrays */
        extern float ug_ceiling[MAX_PORTS][MAX_PORTS], uq_ceiling[MAX_PORTS][MAX_PORTS];
        extern float wg_ceiling[MAX_PORTS][MAX_PORTS], wq_ceiling[MAX_PORTS][MAX_PORTS];
        extern float ug_wall[MAX_PORTS][MAX_PORTS], wg_wall[MAX_PORTS][MAX_PORTS];
        extern float uq_roof[MAX_PORTS][MAX_PORTS];
        //Necessary additions?
        extern float ug_roof[MAX_PORTS][MAX_PORTS], uq_wall[MAX_PORTS][MAX_PORTS];

        /* Inverse Matrix Arrays */
        extern float aq[MAX_PORTS][MAX_PORTS], ag[MAX_PORTS][MAX_PORTS];

        /* Coordinate and Aerodynamic Parameter Arrays */
        extern float x_roof[MAX_PORTS], y_roof[MAX_PORTS], z_roof[MAX_PORTS];
        extern float x_wall[MAX_PORTS], y_wall[MAX_PORTS], z_wall[MAX_PORTS];
        extern float x_vortex[MAX_PORTS], y_vortex[MAX_PORTS], z_vortex[MAX_PORTS];
        extern float sb_vortex[MAX_PORTS], psi_vortex[MAX_PORTS], alpha_vortex[MAX_PORTS];
        extern float x_source[MAX_PORTS], y_source[MAX_PORTS], z_source[MAX_PORTS];
        extern float sb_source[MAX_PORTS], psi_source[MAX_PORTS], alpha_source[MAX_PORTS];
        extern float cpowl[MAX_PORTS], ucp_wall[MAX_PORTS], ucp_roof[MAX_PORTS], uxq_roof[MAX_PORTS];
        extern float uxg_wall[MAX_PORTS], wxg_wall[MAX_PORTS], ulift[MAX_PORTS], ublockage[MAX_PORTS];
        extern float gamma[MAX_PORTS], sigma[MAX_PORTS];

        extern float xarray[30], yarray[30], yarray2[30], yarray3[30];
        extern float cp_empty_wall[25], cp_empty_roof[25];
        extern float sarea, awb, bwb, ec1, ec2, ec3, xbcor;
        extern float TunnelSpan, TunnelHeight, xp_vortex, xp_source, xpiv;
        float dum1[3], dum2[6];
        float x1, y1, z1, x2, y2, z2;
        float u1, v1, w1, u2, v2, w2, u3, v3, w3, u4, v4, w4;
        float eps;

        const double RAD = PI/180, DEG = 180/PI;
        const float SINGULARITY_STRENGTH = 0.5;
        const int TYPE_OF_SINGULARITY = 1, ITRUNC = 1;

        /* Read input */

        printf(" Enter the name of the main input file: ");
        scanf("%s", &geometry_file);
        //-------------------------------------------------------------
        printf(" %s \n", geometry_file);
        //-------------------------------------------------------------

        geom_file = fopen(geometry_file, "r");

        fgets(title, 73, geom_file);
        //-------------------------------------------------------------
        printf("%s \n", title);
        //-------------------------------------------------------------
        fscanf(geom_file, "%d %d %d %d %d %d %d %d", &itermax, &matprt, &SaveMatrix, &iprt,
                &krosg, &krosq, &icorr, &jetefct);
        //-------------------------------------------------------------
        printf("\nWorking fine.");
        printf("\n %d %d %d %d %d %d %d %d \n", itermax, matprt, SaveMatrix, iprt, krosg,
                krosq, icorr, jetefct);
        //-------------------------------------------------------------
```

```
	fscanf(geom_file, "%f %f %f %f %f %f %f", &sarea, &awb, &bwb, &ec1, &ec2, &ec3,
		&xbcor);
	//-----------------------------------------------------------
	printf("\nWorking fine.");
	printf("\n %f %f %f %f %f %f %f \n", sarea, awb, bwb, ec1, ec2, ec3, xbcor);
	//-----------------------------------------------------------

	printf("%s", title);
	printf("\t %d \t %d \t %d \t %d \t %d \t %d \t %d \t %d", itermax, matprt, SaveMatrix,
		iprt, krosg, krosq, icorr, jetefct);
	printf("\n \t %f \t %f \t %f \t %f \t %f \t %f \t %f", sarea, awb, bwb, ec1,
		ec2, ec3, xbcor);

	icor = abs(icorr);
	if (krosg+krosq == 0)
		itermax = 1;
	if (SaveMatrix == 2)
{
		fscanf(geom_file, "%d %d %d %d %d %d", &layers, &irf, &nr, &nw, &nv, &ns);
		fscanf(geom_file, "%f %f %f %f", &TunnelSpan, &TunnelHeight, &xp_vortex,
			&xp_source);
		printf("\n\t%d \t%d \t%d \t%d \t%d \t%d", layers, irf, nr, nw, nv, ns);
		printf("\n\t%f \t%f \t%f \t%f", TunnelSpan, TunnelHeight, xp_vortex, xp_source);

		for (i=0; i<nr; i++)
		{
			for (j=0; j<3; j++)
			{
				fscanf(geom_file, "%f", &dum1[j]);
			}
			printf("\n\t%f \t%f \t%f", dum1[0], dum1[1], dum1[2]);
		}

		for (i=0; i<nw; i++)
		{
			for (j=0; j<3; j++)
			{
				fscanf(geom_file, "%F", &dum1[j]);
			}
			printf("\n\t%f \t%f \t%f", dum1[0], dum1[1], dum1[2]);
		}

		for (i=0; i<nv; i++)
		{
			for (j=0; j<6; j++)
			{
				fscanf(geom_file, "%f", &dum2[j]);
			}
			printf("\n\t%f \t%f \t%f \t%f \t%f \t%f", dum2[0], dum2[1], dum2[2],
				dum2[3], dum2[4], dum2[5]);
		}

		for (i=0; i<ns; i++)
		{
			for (j=0; j<6; j++)
			{
				fscanf(geom_file, "%f", &dum2[j]);
			}
			printf("\n\t%f \t%f \t%f \t%f \t%f \t%f", dum2[0], dum2[1], dum2[2],
				dum2[3], dum2[4], dum2[5]);
		}

		for (i=0; i<=nr-1; i++)
		{
			fscanf(geom_file, "%f", &cp_empty_roof[i]);
			printf("\n%f", cp_empty_roof[i]);
		}

		for (i=0; i<=nw-1; i++)
		{
			fscanf(geom_file, "%f", &cp_empty_wall[i]);
			printf("\n%f", cp_empty_wall[i]);
		}
}

	if (SaveMatrix < 1)
		goto geometry_input;
```

```
		/* call tapeio */

		SaveMatrix = 1;
		goto FormAInverse;

geometry_input: ;

fscanf(geom_file, "%d %d %d %d %d %d", &layers, &irf, &nr, &nw, &nv, &ns);
fscanf(geom_file, "%f %f %f %f", &TunnelSpan, &TunnelHeight, &xp_vortex,
		&xp_source);

for (i=0; i<=nr−1; i++)
{
		fscanf(geom_file, "%f %f %f", &x_roof[i], &y_roof[i], &z_roof[i]);

		x_roof[i] = x_roof[i]*TunnelSpan;
		y_roof[i] = y_roof[i]*TunnelSpan;
		z_roof[i] = z_roof[i]*TunnelHeight;
}

for (i=0; i<=nw−1; i++)
{
		fscanf(geom_file, "%f %f %f", &x_wall[i], &y_wall[i], &z_wall[i]);

		x_wall[i] = x_wall[i]*TunnelSpan;
		y_wall[i] = y_wall[i]*TunnelSpan;
		z_wall[i] = z_wall[i]*TunnelHeight;
}

for (i=0; i<=nv−1; i++)
{
		fscanf(geom_file, "%f %f %f %f %f %f", &x_vortex[i], &y_vortex[i],
			&z_vortex[i], &sb_vortex[i], &psi_vortex[i], &alpha_vortex[i]);

		x_vortex[i] = x_vortex[i]*TunnelSpan;
		y_vortex[i] = y_vortex[i]*TunnelSpan;
		z_vortex[i] = z_vortex[i]*TunnelHeight;
		sb_vortex[i] = sb_vortex[i]*TunnelSpan;
		psi_vortex[i] = psi_vortex[i]*RAD;
		alpha_vortex[i] = alpha_vortex[i]*RAD;
}

for(i=0; i<=ns−1; i++)
{
		fscanf(geom_file, "%f %f %f %f %f %f", &x_source[i], &y_source[i],
			&z_source[i], sb_source[i], psi_source[i], alpha_source[i]);

		x_source[i] = x_source[i]*TunnelSpan;
		y_source[i] = y_source[i]*TunnelSpan;
		z_source[i] = z_source[i]*TunnelHeight;
		sb_source[i] = sb_source[i]*TunnelSpan;
		psi_source[i] = psi_source[i]*RAD;
		alpha_source[i] = alpha_source[i]*RAD;
}

for (i=0; i<=nr−1; i++)
{
		fscanf(geom_file, "%f", &cp_empty_roof[i]);
}

for (i=0; i<=nw−1; i++)
{
		fscanf(geom_file, "%f", &cp_empty_wall[i]);
}

printf(" Geometry Input Completed!! ");

/* Set up source influence matrices */

InitialImageLayer = 0;

for (j=0; j<=nw−1; j++)
{
		xpiv = xp_source*sb_source[i];
		pivot(x_source[j], y_source[j], z_source[j], xpiv, sb_source[j], psi_source[j],
			alpha_source[j], x1, y1, z1, x2, y2, z2);
```

```
                                        /* Cross Effect − Source/(Roof−Floor) */

                                        u3 = 0.0;
                                        u4 = 0.0;

                                        for (i=0; i<=nr−1; i++)
                                        {
                                                Influence(x1, y1, z1, x2, y2, z2, 0.5, 1, InitialImageLayer,
                                                        1, x_roof[i], y_roof[i], z_roof[i], u1, v1, w1);
                                                Influence(x2, −y2, z2, x1, −y1, z1, 0.5, 1, InitialImageLayer,
                                                        1, x_roof[i], y_roof[i], z_roof[i], u2, v2, w2);
                                                if (irf==0)
                                                {
                                                        goto Step38;
                                                }

                                                Influence(x1, y1, z1, x2, y2, z2, 0.5, 1, InitialImageLayer,
                                                        1, x_roof[i], y_roof[i], −z_roof[i], u3, v3, w3);
                                                Influence(x2, −y2, z2, x1, −y1, z1, 0.5, 1, InitialImageLayer,
                                                        1, x_roof[i], y_roof[i], −z_roof[i], u2, v2, w2);

Step38: ;

                        uq_roof[i][j] = u1 + u2 − (u3 + u4);
                                        }

                                        /* Direct Effect − Source/Wall */

                                        for (i=0; i<=nw−1; i++)
                                        {
                                                Influence(x1, y1, z1, x2, y2, z2, 0.5, 1, InitialImageLayer,
                                                        1, x_wall[i], y_wall[i], z_wall[i], u1, v1, w1);
                                                Influence(x2, −y2, z2, x1, −y1, z1, 0.5, 1, InitialImageLayer,
                                                        1, x_wall[i], y_wall[i], z_wall[i], u2, v2, w2);
                                                ug_wall[i][j] = u1 + u2;
                                        }

                                        /* Centerline Interference Matrix (X corresponds to wall points) */

                                        for (i=0; i<=nw−1; i++)
                                        {
                                                Influence(x1, y1, z1, x2, y2, z2, 0.5, 1, 1, 1, x_wall[i], 0.0, 0.0,
                                                        u1, v1, w1);
                                                Influence(x2, −y2, z2, x1, −y1, z1, 0.5, 1, 1, 1, x_wall[i], 0.0, 0.0,
                                                        u2, v2, w2);
                                                uq_ceiling[i][j] = u1 + u2;
                                                wq_ceiling[i][j] = w1 + w2;
                                        }

                                } /* End of loop to set up influence matrices for sources */

                                /* Set up influence matrices for vortices */

                                for (j=0; j<=nv−1; j++)
                                {
                                        xpiv = xp_vortex*sb_vortex[j];
                                        pivot(x_vortex[j], y_vortex[j], z_vortex[j], xpiv, sb_vortex[j]/2.0,
                                                psi_vortex[j], alpha_vortex[j], x1, y1, z1, x2, y2, z2);

                                        /* Direct Effect − Vortex/(Roof − Floor) */

                                        u3 = 0.0;
                                        u4 = 0.0;

                                        for (i=0; i<=nr−1; i++)
                                        {
                                                Influence(x1, y1, z1, x2, y2, z2, 1.0, 2, InitialImageLayer, 1,
                                                        x_roof[i], y_roof[i], z_roof[i], u1, v1, w1);
                                                Influence(x2, −y2, z2, x1, −y1, z1, 1.0, 2, InitialImageLayer, 1,
                                                        x_roof[i], y_roof[i], z_roof[i], u2, v2, w2);

                                                if (irf==0)
                                                {
                                                        goto Step68;
                                                }
```

```
                Influence(x1, y1, z1, x2, y2, z2, 1.0, 2, InitialImageLayer, 1,
                    x_roof[i], y_roof[i], −z_roof[i], u3, v3, w3);
                Influence(x2, −y2, z2, x1, −y1, z1, 1.0, 2, InitialImageLayer, 1,
                    x_roof[i], y_roof[i], −z_roof[i], u4, v4, w4);

Step68: ;

        uq_roof[i][j] = u1 + u2 − (u3 + u4);
            }

            /* Cross Effect − Vortex/Wall */

            for (i=0; i<=nw−1; i++)
            {
                Influence(x1, y1, z1, x2, y2, z2, 1.0, 2, InitialImageLayer, 1,
                    x_wall[i], y_wall[i], z_wall[i], u1, v1, w1);
                Influence(x2, −y2, z2, x1, −y1, z1, 1.0, 2, InitialImageLayer, 1,
                    x_wall[i], y_wall[i], z_wall[i], u2, v2, w2);
                ug_wall[i][j] = u1 + u2;
                wg_wall[i][j] = w1 + w2;
            }

            /* Centerline Interference Matrix (X corresponds to wall points) */

            for (i=0; i<=nw−1; i++)
            {
                Influence(x1, y1, z1, x2, y2, z2, 1.0, 2, 1, 1, x_wall[i], 0.0, 0.0,
                    u1, v1, w1);
                Influence(x2, −y2, z2, x1, −y1, z1, 1.0, 2, 1, 1, x_wall[i], 0.0, 0.0,
                    u2, v2, w2);
                uq_ceiling[i][j] = u1 + u2;
                wq_ceiling[i][j] = w1 + w2;
            }

        } /* End of loop to set up influence matrices for vortices */

FormAInverse: ;

    /* Form [A] inverse for source/wall */

    printf("\n Forming the inverse matrices ");
    eps = 0.0;

    for (i=0; i<=ns−1; i++)
    {
        for (j=0; j<=ns−1; j++)
        {
            aq[i][j] = 0.0;

            for (k=0; k<=nw−1; k++)
            {
                aq[i][j] = aq[i][j] + ug_wall[k][i]*ug_wall[k][j];
            }
        }
    }
    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
    printf("\n Formed the first one!");
    //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

    ierr = 0;
    /* Call gjrv(aq,ns,25,eps,ierr) */

    if (ierr != 0)
    {
        goto Step310;
    }

    /* Form [A] inverse for vortices/roof */

    eps = 0.0;

    for (i=0; i<=nv−1; i++)
    {
        for (j=0; j<=nv−1; j++)
        {
            ag[i][j] = 0.0;
```

```
                                        for (k=0; k<=nr−1; k++)
                                        {
                                                ag[i][j] = ag[i][j] + uq_roof[k][i]*uq_roof[k][j];
                                        }
                                }
                        }
                        //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
                        printf("\n Formed the second one!");
                        //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

                        ierr = 0;
                        /* Call gjrv(ag,nv,25,eps,ierr) */

                        if (ierr != 0)
                        {
                                goto Step310;
                        }

                        /* Optionally Print/Save Matrices */

                        if (SaveMatrix != 0)
                        {
                                /* call tapeio(SaveMatrix, 8) */
                                //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
                                printf("\n Call TapeIO(SaveMatrix, 8).");
                                //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
                        }
                        /*
                        if (iprt == 0)
                        {
                                goto Step305;
                        }
                        */
                        printf("\n %s", title);
                        printf("\n%f \t%f \t%f", TunnelSpan, TunnelHeight, layers);

Step305: ;
                goto Step320;

Step310: ;
                /* Call Out(ug_roof, 25, 25, nr, nv, 'UGammaRoof', 11) */

Step320: ; /* BEGIN INPUT SIGNATURE FROM TEST */

                /*
================================================================================================================
==
                 Read in run number and wall pressures.  Set up boudary contitions.

                        The structure of the program for inputting the run identification and wall
                        signatures in the next 15 statements are for the Knee−Blown Flap experiments
                        of Lockheed−Georgia.  The user should make appropriate changes in this section
                        and re−write the function "ReadCp" to define the arrays "ucp_roof" and "ucp_wall"
                        as indicated below:

                          ucp_roof[i] = u_roof[i] − u_floor[i],       if IRF = 1
                          u_roof[i],                  if IRF = 0

                          ucp_wall[i] = (u_wall1[i] + u_wall2[i])/2
                /*
================================================================================================================
== */

                fscanf(geom_file, "%d %d %d %d %d %d %d %d", &itest, &irun, &ipmin, &ipmax,
                        &ifloor, &iroof, &iwall1, &iwall2);
                printf("\n Irun = %d", irun);

                printf("\n Enter filename for load coefficients: ");
                scanf("%s", LoadCoefficientFile);
                AeroCoeffFile = fopen(LoadCoefficientFile, "w");

                ipdo = ipmin − 1;

Step330: ;

                ipdo = ipdo + 1;
```

```
                    if (ipdo > ipmax)
                    {
                              goto Step600;
                    }

                    ip = ipdo;

                    if(ip == 0)
                    {
                              printf("\n What file contains the experimental data ");
                              scanf("%s", ExperimentalData);
                              ExpDataFile = fopen(ExperimentalData, "r");
                    }

                    /* Call ReadCP(ucp_roof, nr, irun, ip, ifloor, iroof, 1, irf) */

                    if (irun < 0)
                    {
                              goto Step600;
                    }

                    if (ip == 0)
                    {
                              goto Step330;
                    }

                    /* Call ReadCP(ucp_wall, nw, irun, ip, iwall1, iwall2, 0, irf) */

                    if (irun < 0)
                    {
                              goto Step600;
                    }

                    /* ================================================== */

                    if (iprt != 0)
                    {
                              printf("\n RUN NUMBER %d \t POINT NUMBER %d \n", irun, ip);
                              printf("\n INPUT BOUNDARY CONDITIONS ( ROOF − FLOOR ) ");
                              printf("\n\t I \tX/B \t");
                    }

Step600: ;
                    fclose (geom_file);

                    return 0;

} /* end main source code */

/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
The following functions are commented out as they are not essential to the operation of this program and are therefore not implemented at this time.
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

void TapeIO(float SaveMatrix, float IU) {
          float x;
          int i, j;
          float Rad = PI/180;
          if (SaveMatrix <= 0)
                    return;
          if (SaveMatrix != 1) {
                    printf("\nOpen the matrix influence file. \n");
                    /*
                    OPEN(IU,file=*,status = 'old')
                    READ (IU,*) LAYERS,IRF,NR,NW,NV,NS
                    READ (IU,*) TunnelSpan,TunnelHeight,XPVOR,XPSRC

                    for (i = 0; i < NR; i++) {
                              /*READ (IU,*) x_roof[i],y_roof[i],z_roof[i]
                              x_roof[i] = x_roof[i]*TunnelSpan;
                              y_roof[i] = y_roof[i]*TunnelSpan;
                              z_roof[i] = z_roof[i]*TunnelHeight;
                    }//for
                    for (i = 0; i < NW; i++) {
                              /*READ (IU,*) x_wall[i],y_wall[i],z_wall[i]
                              x_wall[i] = x_wall[i]*TunnelSpan;
                              y_wall[i] = y_wall[i]*TunnelSpan;
```

```
                z_wall[i] = z_wall[i]*TunnelHeight;
        }//for
        for (i = 0; i < NV; i++) {
                /*READ (IU,*) x_vortex[i],y_vortex[i],z_vortex[i],sb_vortex[i],psi_vortex[i],
                alpha_vortex[i]
                x_vortex[i] = x_vortex[i]*TunnelSpan;
                y_vortex[i] = y_vortex[i]*TunnelSpan;
                z_vortex[i] = z_vortex[i]*TunnelHeight;
                sb_vortex[i] = sb_vortex[i]*TunnelSpan;
                psi_vortex[i] = psi_vortex[i]*Rad;
                alpha_vortex[i] = alpha_vortex[i]*Rad;
        }//for
        for (i = 0; i < NS; i++) {
                /*READ (IU,*) x_source[i],y_source[i],z_source[i],sb_source[i],psi_source[i],
                alpha_source[i]
                x_source[i] = x_source[i]*TunnelSpan;
                y_source[i] = y_source[i]*TunnelSpan;
                z_source[i] = z_source[i]*TunnelHeight;
                sb_source[i] = sb_source[i]*TunnelSpan;
                psi_source[i] = psi_source[i]*Rad;
                alpha_source[i] = alpha_source[i]*Rad;
        }//for
        for (i = 0; i < NR; i++) {
                for (j = 0; j < NV; j++)
                        /*READ (IU,*) ug_roof[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*READ (IU,*) ug_wall[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*READ (IU,*) wg_wall[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NS; j++)
                        /*READ (IU,*) uq_wall[i][j];
        }//for
        for (i = 0; i < NR; i++) {
                for (j = 0; j < NS; j++)
                        /*READ (IU,*) uq_roof[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NS; j++)
                        /*READ (IU,*) uq_ceiling[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NS; j++)
                        /*READ (IU,*) wq_ceiling[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*READ (IU,*) ug_ceiling[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*READ (IU,*) wg_ceiling[i][j];
        }//for
        if (SaveMatrix == 3)
                return;
        for (i = 0; i < NV; i++) {
                for (j = 0; j < NV; j++)
                        /*READ (IU,*) ag[i][j];
        }//for
        for (i = 0; i < NS; i++) {
                for (j = 0; j < NS; j++)
                        /*READ (IU,*) aq[i][j];
        }//for
        return;
}//endif
/*
OPEN(IU,file=*'Matrix storage file',status='unknown')
WRITE (IU,1100) Layers,IRF,NR,NW,NV,NS
WRITE (IU,1000) TunnelSpan,TunnelHeight,XPVOR,XPSRC
for (i = 0; i < NR; i++)
        WRITE (IU,1000) x_roof[i]/TunnelSpan,y_roof[i]/TunnelSpan,z_roof[i]/TunnelHeight;
for (i = 0; i < NW; i++)
```

```
                WRITE (IU,1000) x_wall[i]/TunnelSpan,y_wall[i]/TunnelSpan,z_wall[i]/TunnelHeight;
        for (i = 0; i < NV; i++)
                WRITE (IU,1000)
x_vortex[i]/TunnelSpan,y_vortex[i]/TunnelSpan,z_vortex[i]/TunnelHeight,sb_vortex[i]/TunnelSpan,psi_vortex[i]/Rad,
                alpha_vortex[i]/Rad;
        for (i = 0; i < NS; i++)
                WRITE (IU,1000)
x_source[i]/TunnelSpan,y_source[i]/TunnelSpan,z_source[i]/TunnelHeight,sb_source[i]/TunnelSpan,psi_source[i]/Rad,
                alpha_souce[i]/Rad;

        for (i = 0; i < NR; i++) {
                for (j = 0; j < NV; j++)
                        /*WRITE (IU,1000) ug_roof[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*WRITE (IU,1000) ug_wall[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*WRITE (IU,1000) wg_wall[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NS; j++)
                        /*WRITE (IU,1000) uq_wall[i][j];
        }//for
        for (i = 0; i < NR; i++) {
                for (j = 0; j < NS; j++)
                        /*WRITE (IU,1000) uq_roof[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NS; j++)
                        /*WRITE (IU,1000) uq_ceiling[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NS; j++)
                        /*WRITE (IU,1000) wq_ceiling[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*WRITE (IU,1000) ug_ceiling[i][j];
        }//for
        for (i = 0; i < NW; i++) {
                for (j = 0; j < NV; j++)
                        /*WRITE (IU,1000) wg_ceiling[i][j];
        }//for
        for (i = 0; i < NV; i++) {
                for (j = 0; j < NV; j++)
                        /*WRITE (IU,1000) ag[i][j];
        }//for
        for (i = 0; i < NS; i++) {
                for (j = 0; j < NS; j++)
                        /*WRITE (IU,1000) aq[i][j];
        }//for
        /*
        1000 FORMAT (5E16.8)
        1100 FORMAT (1615)

        return;
}//TapeIO

void ReadCP(float VCP, int NO, float IRun, float IP, int IRail, int JRail, float Lift,
                                                                        float irf) {
        float alpha_u, POU, QOU, CMUU, CLU, CDU, CMU, Q;
        float VCP[MAX_PORTS], VCP_a[30][8], XCP[25], XCP7[30];
        float NTest, NRun, NPnt, NRails, Ncp, NN;
        float Sign, Denom;
        int i, j, IRL;
Step10: ;
                /*READ (10,500,END=98) NTest,NRun,NPnt,NRails,Ncp
                if (NO != Ncp) {
                        /*WRITE (6,610) NO, Ncp
                        IRun = −IRun;
                        return;
                }//endif
                if ((1 − IRail)*(NRails − IRail) > 0 || (1 − JRail)*(NRails − JRail) > 0) {
                        /*WRITE (6,620) IRail, JRail, 1, NRails
```

```
                        IRun = −IRun;
                        return;
                }//endif
                for (j = 0; j < NRails; j++) {
                        for (i = 0; i < Ncp; i++)
                                /*READ (10,510) VCP_a[i][j];
                }//for
                /*READ (10,520) alpha_u,POU,QOU,CMUU,CLU,CDU,CMU
                if (NRun != IRun)
                        goto Step10;
                if (NPnt == IP)
                        goto Step25;
                if (NPnt < IP)
                        goto Step10;
                IP = 0;
                goto Step45;
Step25: ;
                Sign = 1;
                Denom = 2;
                if (Lift == 0)
                        goto Step26;
                Sign = −1*irf;
                Denom = 1;
Step26: ;
                IRL = IRail;
                for (i = 0; i < NO; i++) {
                        VCP[i] = (VCP_a[i][JRail] + Sign*VCP_a[i][IRL])/Denom;
                        Q = 1 − VCP[i];
                        if (Q < 0) {
                                /*WRITE (6,330) i, VCP[i]
                                IRun = −IRun;
                                return;
                        }//endif
                        VCP[i] = sqrt(Q) − 1;
                }//for
Step45: ;
                NN = (NO + 8)/9;
                NN = NN*NRails + 2;
                for (i = 0; i < NN; i++) {
                        /*BACKSPACE 10;
                }//for
                return;
Step98: ;
                /*WRITE (6,600) IRun
                IRun = −IRun;
                /*
                500 FORMAT (16I5)
                510 FORMAT (2X,9F8.4)
                520 FORMAT (7f10.2)
                600 FORMAT (///’ −−−_ EOF ON TAPE10 _−−−’//’ ** NO DATA WITH IDRUN −’, I3,’ **’//)
                610 FORMAT (’ −−: ERROR IN ROUTINE "ReadCP" :−−’/
                                        ’    MISMATCH BETWEEN AVAILABLE AND REQUESTED NO OF’,
                                        ’ CP−MEASUREMENT POINTS’/’    N−REQ =’,I4, ’  N−AVL =’,
                                        I4/)
                620 FORMAT (’ −−: ERROR IN ROUTINE "ReadCP" :−−’/
                                        ’    RAIL NUMBERS OUT OF RANGE’/’    RAILS REQUESTED =’,
                                        2I3/’    RAILS AVAILABLE = ’, 2I3/)
                630 FORMAT (’ −−: ERROR IN ROUTINE "ReadCP" :−−’/
                                        ’    SQRT(1−CP) IS NEGATIVE’/’    I =’, I3, ’(1−CP) =’,
                                        E13.5)

                return;
}//ReadCP

void Inter(float xd, float fd, float xi, float fi, float Max, float IDO) {
        float xd[1], fd[1], xi[1], fi[1];
        int i, j, jp, jm;
        j = 1;
        for (i = 0; i < IDO; i++) {
                if (x_i[i] > x_d[0]) {
                        do {
                                if (x_d[j−1] >= x_i[i])
                                        break;
                                j++;
                        }while (j <= Max);
                }//endif
                else {
```

```
                        j = 2;
                }//else
                jp = j;
                jm = j − 1;
                f_i[i] = f_d[jm] + (f_d[jp] − f_d[jm])*(x_i[i] − x_d[jm])/(x_d[jp] − x_d[jm]);
        }//for
        return;
}//Inter

void Output(int NO, int NS, float xm, float Span, float Lift, float vcp, float xcp, float Sigma,
                                                                        float DC, float DW, float ICorr, float Jet) {
        float SPB, L, Valu, x;
        float vcp[MAX_PORTS], xcp[MAX_PORTS], Sigma[MAX_PORTS], DC[25][3], DW[25][3], xm[MAX_PORTS], NP[61];
        int n, j;
        SPB = Span/TunnelSpan*2;
        L = 2*Lift + 1;
        if (Lift == 1) {
                //WRITE (6,11)
                Valu = sqrt(TunnelSpan*TunnelHeight);
        }//endif
        else {
                Valu = TunnelSpan*TunnelHeight;
                //WRITE (6,10)
        }//else
        for (n = 0; n < NO; n++) {
                x = x_cp[n]/TunnelSpan;
                /*
                WRITE (6,'(2X,6F8.4)') x,VCP[n]
                for (j = 1; j <= 3; j++)
                        WRITE (6,'(2X,6F8.4)') dw[n][j]
                WRITE (6,'(2X,6F8.4)') dc[n,1]

        }//for
        /*
        10 FORMAT (/6X,'POSN',3X,'INPUT,3(4X,'WALL'),5X,'C/L',4X,
                        'BLOCKAGE CORRECTION' /
                        7X,'X/B',2(4X,'DU/U'),4X,'DV/U',4X,'DW/U',4X,'DU/U'  I,
                        5X,'X/B',3X,' S' )
        11 FORMAT (/6X,'POSN',3X,'INPUT,3(4X,'WALL'),5X,'C/L',6X,
                        'LIFT CORRECTION' /
                        7X,'X/B',2(4X,'DU/U'),4X,'DV/U',4X,'DW/U',4X,'DU/U'  I,
                        5X,'X/B',3X,' S' )

        return;
}//Output

void Out(float A[][], int ID, int JD, float MM, float NN, float Titl, float NCAR) {
        float MLo, MHi, NLo, NHi, ALo, AHi;
        int m, n;
        char TITL[79];
        MLo = 1;
        MHi = (NCAR + 3)/4;
        /*
        for (m = 0; m < NCAR; m++)
                WRITE (6,600) Titl[m]

Step10: ;
                MHi = MLo + 8;
                if (MHi > MM)
                        MHi = MM;
                /*
                for (m = MLo; m <= MHi; m++)
                        WRITE (6,610) m
                WRITE (6,612)

                for (n = 0; n < NN; n++) {
                        /*
                        WRITE (6,620) n
                        for (m = MLo; m <= MHi; m++)
                                WRITE (6,620) A[m][n]

                }//for
        MLo = MLo + 9;
        if (MLo <= MM)
                goto Step10;
        MLo = 1;
        NLo = 1;
```

```
	MHi = 1;
	NHi = 1;
	AHi = a[1][1];
	ALo = a[1][1];
	for (m = 0; m < MM; m++) {
		for (n = 0; n < NN; n++) {
			if (a[m][n] > AHi) {
				AHi = a[m][n];
				MHi = m;
				NHi = n;
			}//endif
			else if (a[m][n] < ALo) {
				ALo = a[m][n];
				MLo = m;
				NLo = n;
			}//endif
		}//for
	}//for
	//WRITE (6,630) MLo,NLo,ALo,MHi,NHi,AHi
	/*
	600 FORMAT (1H1,79A1)
	610 FORMAT (/8X,'M = ',I2,9I13)
	612 FORMAT (3X,1HN/)
	620 FORMAT (I4,1X,9E13.5)
	630 FORMAT (//' MINIMUM AND MAXIMUM :'/26X,1HM,3X,1HN,5X,5HVALUE,
			/(23X,2I4,E13.5))

	return;
}//Out

void JetAdd(float NW, float NR, float VWall, float VRoof, float VCntr, float itest, float irun,
		float IP) {
	float NN, IRew, IT, IR, IPT, XOB, UWL, URF;
	float VCl[3], v_wall[25][3], v_roof[25][3], VCntr[25][3];
	int i, k, j;
	if (NW >= NR)
		NN = NW;
	else
		NN = NR;
	IRew = 0;
Step10: ;
		//READ (11,1000,END = 50) IT, IR, IPT
		if (IT == itest && IR == irun && IPT == IP) {
			for (i = 0; i < NN; i++) {
				/*
				READ (11,1000) j, XOB, UWL, URF
				for (k = 1; k <= 3; k++)
					READ (11,1000) v_cl[k]

				v_wall[i][1] = v_wall[i][1] + UWL;
				v_roof[i][1] = v_roof[i][1] + URF;
				for (k = 1; k <= 3; k++)
					v_cntr[i][k] = v_cntr[i][k] + v_cl[k];
			}//for
			return;
		}//endif
		else {
			for (i = 0; i < NN; i++) {
				//READ (11,1000)
			}//for
			goto Step10;
		}//else
		if (IRew == 0) {
			IRew = 1;
			//REWIND 11
			goto Step10;
		}//endif
		//WRITE (6,600) itest,irun,IP
		/*
		600  FORMAT (//' *** NO JET_EFFECT DATA FOUND IN TAPE11 FOR '
				'ITEST/IRUN/IPT = ',3(I3,1H/), ' ***' //)
		1000 FORMAT (16I5)
		1100 FORMAT (I5,7E15.8)

	return;
}//JetAdd
```

```
*/
```

```
/******************
/* Subroutines.h
/*  Author: James L. Buckingham, Jr. (2001)
/*  Contains function prototypes for functions used in wallcorrection.c
/******************/

void pivot(float x, float y, float z, float xpiv, float sp, float psi, float alfa,
                                        float x1, float y1, float z1, float x2, float y2, float z2);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Determines the end points of a line of length "sp" when it is pitched and yawed
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void Influence(float x1, float y1, float z1, float x2, float y2, float z2, float strength,
                                                 int SingularityType, int InitialImageLayer, int itrunc, float xp, float yp,
                                                         float zp, float du, float dv, float dw);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Parameters:
                x1, y1, z1                              :       Line−singularity endpoint coordinates, First point
                x2, y2, z2                              :       Line−singularity endpoint coordinates, Second
point
                strength                                        :       Singularity strength (total for
SingularityType = 1, 3

         per unit length for SingularityType = 2)
                SingularityType         :       Type of Singularity...  1 = Source/Sink

                2 = Horseshoe Vortex (gamma +ve y1−>y2)

                3 = Doublet
                InitialImageLayer       :       Initial image layer ( = 0 to include model in tunnel)
                xp, yp, zp                              :       Calculation Point
                du, dv, dw                              :       Incremental velocity components, modified and
returned
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void LFUNCS(float XX, float y, float z, float y1, float z1, float y2, float z2, float L1F,
                                                 float L2F);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Returns L1F and L2F functions for calculation of v and w velocities respectively at point
                XX,y,z due to the edges y1,z1 y2,z2 of a rectangular hole in an infinite sheet source
                at XX = 0
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

float TFUNC(float XX, float y, float z, float y1, float z1, float y2, float z2);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Returns value for computation of u−velocity at point XX,y,z due to a finite rectangular
                sheet source y1,z1 y2,z2 at XX = 0
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void LNDBEQ(float VMU2, float xc, float yc, float zc, float VL1, float du, float dv,
                                                         float dw, float IT);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Line Doublet equation
        The value of IT takes the value of 2 for type (1,2) and 3 for type (1,3)
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void LNDBGN(float xa, float ya, float za, float xb, float yb, float zb, float VMU,
                                                         float cx, float cy, float cz, float xp, float yp, float zp, float du,
float dv,
                                                         float dw, float VMUR, float cxR, float cyR, float czR);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Generates du, dv, dw velocities due to a line doublet
        Parameters:
                xa,ya,za xb,yb,zb               :       End coordinates of the line doublet
                VMU                                                     :       Couple of the doublet
                cx, cy, cz                              :       Direction cosines of the couple vector
```

```
                                                                                cx =
x/R
                                                                                cy =
y/R
                                                                                cz =
z/R
                                                                                R =
sqrt(x^2 + y^2 + z^2)
                VL1                                                             :       Length of the
line doublet
                xp, yp, zp                              :       Coordinates of any point in space
        Values Calculated:
                du, dv, dw              :       Velocity components at point P in the (x,y,z) system
                VMUR                            :       Resultant componant of the couple of the doublet
                                                                                Normal to the axis of the
doublet
                cxR, cyR, czR           :       Direction cosines of the couple vector
                                                                                Used in calculating du,dv,dw
        NOTE: The component of the couple vector along the axis of the doublet was not
        included in determining du, dv, dw. This eliminates point−source and point−sink effects
        in the  model due to the line doublet.
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void LNVXGN(float xa, float ya, float za, float xb, float yb, float zb, float Circ, float xp,
                                                                        float yp, float zp, float du, float dv, float dw);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Generates incremental velocity due to a line vortex
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void LNVXEQ(float Circ, float xpG, float ypG, float zpG, float VL, float duG, float dvG,
                        float dwG);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Line vortex equation
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void LNSCGN(float xa, float ya, float za, float xb, float yb, float zb, float VM, float xp,
                                                                        float yp, float zp, float du, float dv, float dw);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Generates incremental velocities due to a line source
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void LNSCEQ(float VM, float xpG, float ypG, float zpG, float VL, float duG, float dvG, float
                                                dwG);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Line source equation
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void CRDTEG(float xaO, float yaO, float zaO, float xbO, float ybO, float zbO, float xpO,
                                                                        float ypO, float zpO, float Exip, float Etap, float Ztap, float
IC);
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Coordinate translation"English to Greek" if IC = 1
                                                                                "Gre
ek to English" if IC = 2
        Origin of the coordinate system will be at the midpoint of A−B
        Transforms (x,y,z) set of axes to (Exi,Eta,Zta) axes with the Exi axis parallel to the center
                line of the doublet
        Parameters:
                A, B                    :       Ends of the line doublet
                P                               :       general point in space
        Calculated Values:
                EPSI, ALFA              :       Angles of rotation of the Eta (y) and Zta (z) axes, respectively
        NOTE: System set up for POSITIVE WINDING numbers for rotation of axes
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/

void GJRV(float A[], float N, float NL, float EPsil, float Ierr);
```

```
        /*-------------------------------------------------------------------------------------------
---
        Matrix inversion routine
        Parameters:
                A                       :       Input array which will be destroyed
                N                       :       Rank of A
                NL              :       Row dimension of A
                EPsil   :       Test value for the pivot point singularity check
                Ierr            :       Nonzero if matrix is singular
                                        If matrix is nonsingular, A contains A-inverse
        -------------------------------------------------------------------------------------------
--*/

//-------------------------------------------------------------------------------------------

void TapeIO(float SaveMatrix, float IU);

void ReadCP(float VCP, float NO, float IRun, float IP, int IRail, int JRail, float Lift,
                                float irf);

void Inter(float xd, float fd, float xi, float fi, float Max, float IDO);

void Output(int NO, float NS, float xm, float Span, float Lift, float vcp, float xcp, float Sigma,
                                                float DC, float DW, float ICorr, float Jet);

void Out(float A, float ID, float JD, float MM, float NN, float Titl, float NCAR);

void JetAdd(int NW, int NR, float VWall, float VRoof, float VCntr, float itest, float irun,
                                                float IP);
```

```
/******************
/* Pivot and Influence.c
/*  Author: James L. Buckingham, Jr. (2001)
/*  Contains functions used in wallcorrection.c
/******************/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include "Subroutines.h"

#define PI 3.142592654

void pivot(float x, float y, float z, float xpiv, float sp, float psi, float alfa,
                        float x1, float y1, float z1, float x2, float y2, float z2) {
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Determines the end points of a line of length "sp" when it is pitched and yawed
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
        float TanP, SinA, CosA;
        TanP = tan(psi);
        SinA = sin(alfa);
        CosA = cos(alfa);
        x1 = x + xpiv*(1 − CosA);
        y1 = y;
        z1 = z + xpiv*SinA;
        x2 = x + xpiv*(1 − CosA) + sp*TanP*CosA;
        y2 = y + sp;
        z2 = z − (sp*TanP − xpiv)*SinA;
        return;
}//pivot

void Influence(float x1, float y1, float z1, float x2, float y2, float z2, float strength,
                                                        int SingularityType, int InitialImageLayer, int itrunc, float xp,
float yp,
                                                        float zp, float du, float dv, float dw) {
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Parameters:
                x1, y1, z1                              :       Line−singularity endpoint coordinates, First point
                x2, y2, z2                              :       Line−singularity endpoint coordinates, Second
point
                strength                                        :       Singularity strength (total for
SingularityType = 1, 3

         per unit length for SingularityType = 2)
                SingularityType         :       Type of Singularity...  1 = Source/Sink

                2 = Horseshoe Vortex (gamma +ve y1−>y2)

                3 = Doublet
                InitialImageLayer       :       Initial image layer ( = 0 to include model in tunnel)
                xp, yp, zp                              :       Calculation Point
                du, dv, dw                              :       Incremental velocity components, modified and
returned
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
        float DCXO, DCYO, DCZO, IBug, IS, DCX, xInf, STPerl, LMax, QBH, xm,
                XX, SignX, yr, zu, yl, zl, TF, FL1, FL2;
        extern float TunnelSpan, TunnelHeight;
        extern int layers;
        int LDO;
        DCXO = 0;
        DCYO = 0;
        DCZO = 0;
        IBug = 0;
        IS = −1;
        du = 0;
        dv = 0;
        dw = 0;
        DCX = DCXO;
        xInf = 10000*TunnelSpan;
        STPerl = strength;
        LMax = layers + 1;
        LDO = 0;
```

```
	if (SingularityType != 2) {
		float ELS = sqrt((x2 − x1)*(x2 − x1) + (y2 − y1)*(y2 − y1) + (z2 − z1)*(z2 − z1));
		STPerl = strength/ELS;
	}//if
	for (LDO = 0; LDO < LMax; LDO++) {
		float FAC, L;
		FAC = 1;
		L = LDO − 1;
		if (L >= InitialImageLayer) {
			float MNDO, NOT, MN1;
			int MDO;
			MNDO = L*2 + 1;
			NOT = L + 1;
			MN1 = MNDO − 1;
			MDO = 0;
			if ((LDO == LMax) && (L > 0))
				FAC = (1/2) + 1/(4*L);
			if (SingularityType != 2)
				FAC = 1;
			for (MDO = 0; MDO < MNDO; MDO++) {
				float M, RevM, yOne, yTwo, NInc;
				int NDO;
				M = MDO − NOT;
				RevM = pow(IS,abs(M));
				yOne = M*TunnelSpan + RevM*y1;
				yTwo = M*TunnelSpan + RevM*y2;
				NInc = MN1;
				NDO = 0;
				if ((MDO == 1) || (MDO == MNDO))
					NInc = MN1;
				for (NDO = 0; NDO < MNDO; NDO = NDO + NInc) {
					float N, RevN, x1IM, x2IM, y1IM, y2IM, z1IM, z2IM, DCY, DCZ, Delu, Delv,
Delw;
					N = NDO − NOT;
					RevN = pow(IS,abs(N));
					x1IM = x1;
					x2IM = x2;
					y1IM = yOne;
					y2IM = yTwo;
					z1IM = N*TunnelHeight + RevN*z1;
					z2IM = N*TunnelHeight + RevN*z2;
					DCY = DCYO*RevN;
					DCZ = DCZO*RevM;
					Delu = du;
					Delv = dv;
					Delw = dw;
					if (RevM*RevN <= 0) {
						float Temp = x2IM;
						x2IM = x1IM;
						x1IM = Temp;
						Temp = y2IM;
						y2IM = y1IM;
						y1IM = Temp;
						Temp = z2IM;
						z2IM = z1IM;
						z1IM = Temp;
					}//if
					if (SingularityType == 1) {
						LNSCGN(x1IM, y1IM, z1IM, x2IM, y2IM, z2IM, STPerl, xp, yp, zp,
Delu, Delv,
	Delw);
					}//endif
					if (SingularityType == 2) {
						float duT1, duBN, duT2, dvT1, dvBN, dvT2, dwT1, dwBN, dwT2;
						duT1 = du;
						duBN = du;
						duT2 = du;
						dvT1 = dv;
						dvBN = dv;
						dvT2 = dv;
						dwT1 = dw;
						dwBN = dw;
						dwT2 = dw;
						LNVXGN(x1IM, y1IM, z1IM, xInf, y1IM, z1IM, STPerl, xp, yp, zp,
duT1, dvT1,
		dwT1);
						LNVXGN(x2IM, y2IM, z2IM, x1IM, y1IM, z1IM, STPerl, xp, yp, zp,
```

```
duBN, dvBN,
		dwBN);
							LNVXGN(xInf, y2IM, z2IM, x2IM, y2IM, z2IM, STPerl, xp, yp, zp,
duT2, dvT2,
			dwT2);
							Delu = duT1 + duBN + duT2;
							Delv = dvT1 + dvBN + dvT2;
							Delw = dwT1 + dwBN + dwT2;
						}//endif
						if (SingularityType == 3) {
							float VMUR, CXR, CYR, CZR;
							VMUR = 0;
							CXR = 0;
							CYR = 0;
							CZR = 0;
							LNDBGN(x1IM, y1IM, z1IM, x2IM, y2IM, z2IM, STPerl, DCX, DCY,
DCZ, xp, yp,
		zp, Delu, Delv,		Delw, VMUR, CXR, CYR, CZR);
						}//endif
						du = du + FAC*Delu;
						dv = dv + FAC*Delv;
						dw = dw + FAC*Delw;
						if (IBug != 0) {
							printf("\n\t%f \t%f \t%f \t%f \t%f \t%f \t%f \t%f \t%f \t%f \t%f\n", M, N,
								xp, yp, zp, Delu, Delv, Delw, du, dv, dw);
						}//endif
					}//for
				}//for
			}//endif
		}//for
		if (itrunc == 0)
			return;
		if ((layers == 0) || (SingularityType != 1))
			return;
		QBH = strength/(2*TunnelSpan*TunnelHeight);
		xm = (x1 + x2)/2;
		XX = xp − xm;
		if (XX < 0)
			SignX = −1;
		if (XX > 0)
			SignX = 1;
		if (abs(XX) < 0.1E−6)
			SignX = 0;
		yr = layers*TunnelSpan + TunnelSpan/2;
		zu = layers*TunnelHeight + TunnelHeight/2;
		yl = −yr;
		zl = −zu;
		TF = TFUNC(XX, yp, zp, yl, zl, yr, zu);
		FL1 = 0;
		FL2 = 0;
		LFUNCS(XX, yp, zp, yl, zl, yr, zu, FL1, FL2);
		du = du + QBH + QBH*(SignX − TF/(2*PI));
		dv = dv + QBH*FL1/(2*PI);
		dw = dw + QBH*FL2/(2*PI);
		return;
}//Influence

void LFUNCS(float XX, float y, float z, float y1, float z1, float y2, float z2, float L1F,
								float L2F) {
		/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
		Returns L1F and L2F functions for calculation of v and w velocities respectively at point
			XX,y,z due to the edges y1,z1 y2,z2 of a rectangular hole in an infinite sheet source
			at XX = 0
		−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
		float dy, dz, xq, y1q, y2q, z1q, z2q, r1, r2, r3, r4, n1, n2, n3, n4, d1, d2, d3, d4;
		dy = y2 − y1;
		dz = z2 − z1;
		xq = XX*XX;
		y1q = (y − y1)*(y − y1);
		y2q = (y − y2)*(y − y1);
		z1q = (z − z1)*(z − z1);
		z2q = (z − z2)*(z − z2);
		r1 = sqrt(xq + y1q + z2q);
		r2 = sqrt(xq + y2q + z2q);
		r3 = sqrt(xq + y2q + z1q);
```

```
		r4 = sqrt(xq + y1q + z1q);
		n1 = r4 + r1 − dz;
		n2 = r2 + r3 + dz;
		n3 = r3 + r4 − dy;
		n4 = r1 + r2 + dy;
		d1 = r4 + r1 + dz;
		d2 = r2 + r3 − dz;
		d3 = r3 + r4 + dy;
		d4 = r1 + r2 − dy;
		L1F = log(n1*n2/(d1*d2));
		L2F = log(n3*n4/(d3*d4));
		return;
}//LFUNCS

float TFUNC(float XX, float y, float z, float y1, float z1, float y2, float z2) {
		/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
		Returns value for computation of u−velocity at point XX,y,z due to a finite rectangular
				sheet source y1,z1 y2,z2 at XX = 0
		−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
		float Sign[4], YY[4], ZZ[4], Sum, Part, XJ;
		int j;
		Sum = 0;
		Part = 0;
		XJ = 0;
		j = 0;
		if (abs(XX) < 0.00000001)
				return Sum;
		Sign[0] = 1;
		Sign[1] = −1;
		Sign[2] = −1;
		Sign[3] = 1;
		YY[0] = y − y1;
		YY[1] = YY[0];
		YY[2] = y − y2;
		YY[3] = YY[2];
		ZZ[0] = z−z1;
		ZZ[1] = z−z2;
		ZZ[2] = ZZ[0];
		ZZ[3] = ZZ[1];
		for (j = 1; j <= 4; j++) {
				XJ = YY[j]*(ZZ[j])/(XX*sqrt(XX*XX + YY[j]*(YY[j]) + ZZ[j]*(ZZ[j])));
				Part = sin(j)*atan(XJ);
				Sum = Sum + Part;
		}//for
		return Sum;
}//TFUNC

void LNDBEQ(float VMU2, float xc, float yc, float zc, float VL1, float du, float dv,
										float dw, float IT) {
		/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
		Line Doublet equation
		The value of IT takes the value of 2 for type (1,2) and 3 for type (1,3)
		−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
		float CONST = VMU2/(4*PI);
		if (IT == 3) {
				float Temp = zc;
				zc = yc;
				yc = Temp;
		}//endif
		float VL12, xcPL, xcML, TMPP, TMPN, dn1, dp1, Temp, DNOMP, DNOMN;
		VL12 = VL1/2;
		xcPL = xc + VL12;
		xcML = xc − VL12;
		TMPP = yc*yc + zc*zc;
		TMPN = yc*yc − zc*zc;
		dn1 = xcML*xcML + TMPP;
		dp1 = xcPL*xcPL + TMPP;
		Temp = (0.01*VL1)*(0.01*VL1);
		if (TMPP − Temp <= 0) {
				Temp = abs(xc) − VL12;
				if (Temp <= 0) {
						du = 0;
						dv = 0;
```

```
				dw = 0;
			}//endif
			else {
				DNOMP = sqrt(dp1*dp1*dp1);
				DNOMN = sqrt(dn1*dp1*dp1);
				du = CONST*yc*(1/DNOMN − 1/DNOMP);
				dv = 5*CONST*(1/(xcPL*xcPL) − 1/(xcML*xcML));
				dw = 0;
			}//else
		}//endif
		else {
			DNOMP = sqrt(dp1*dp1);
			DNOMN = sqrt(dn1*dn1);
			du = CONST*yc*(1/DNOMN − 1/DNOMP);
			dv = CONST*((xcPL*(TMPN*dp1 + TMPP*yc*yc)/DNOMP − xcML*(TMPN*dn1 + TMPP*yc*yc)/DNOMN)
		/(TMPP*TMPP));
			dw = CONST*yc*zc*(xcPL*(2*dp1 + TMPP)/DNOMP − xcML*(2*dn1 + TMPP)/DNOMN)/(TMPP*TMPP);
			if (IT > 0) {
				Temp = dw;
				dw = dv;
				dv = Temp;
			}//endif
		}//else
		return;
}//LNDBEQ

void LNDBGN(float xa, float ya, float za, float xb, float yb, float zb, float VMU,
								float cx, float cy, float cz, float xp, float yp, float zp, float du,
float dv,
								float dw, float VMUR, float cxR, float cyR, float czR) {
		/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
		Generates du, dv, dw velocities due to a line doublet
		Parameters:
			xa,ya,za xb,yb,zb	:		End coordinates of the line doublet
			VMU								:		Couple of the doublet
			cx, cy, cz						:		Direction cosines of the couple	vector
												cx = x/R
												cy = y/R
												cz = z/R
												R = sqrt(x^2 + y^2 + z^2)
			VL1								:		Length of the line doublet
			xp, yp, zp		:		Coordinates of any point in space Values Calculated:
			du, dv, dw		:		Velocity components at point P in the (x,y,z) system
			VMUR			:		Resultant component of the couple		of the doublet Normal to the axis of
								the doublet
			cxR, cyR, czR	:		Direction cosines of the couple
								vector Used in calculating du,dv,dw
		NOTE: The component of the couple vector along the axis of the doublet was
			not included in determining du, dv, dw. This eliminates point−source
			and point−sink effects in the model due to the line doublet.
		−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
		float Temp, xaO, xbO, xpO, yaO, ybO, ypO, zaO, zbO, zpO, IC, xpCG, ypCG, zpCG,
			cxG, cyG, czG, VMU2, VMU3, VL1, IT, du2, dv2, dw2, du3, dv3, dw3,
			duG, dvG, dwG;
		Temp = (xa + xb)/2;
		xaO = xa − Temp;
		xbO = xb − Temp;
		xpO = xp − Temp;
		Temp = (ya + yb)/2;
		yaO = ya − Temp;
		ybO = yb − Temp;
		ypO = yp − Temp;
		Temp = (za + zb)/2;
		zaO = za − Temp;
		zbO = zb − Temp;
		zpO = zp − Temp;
		IC = 1;
		xpCG = 0;
		ypCG = 0;
		zpCG = 0;
		cxG = 0;
		cyG = 0;
		czG = 0;
		CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, xpO, ypO, zpO, xpCG, ypCG, zpCG, IC);
		CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, cx, cy, cz, cxG, cyG, czG, IC);
```

```
        VMU2 = VMU*cyG;
        VMU3 = VMU*czG;
        VMUR = sqrt(VMU2*VMU2 + VMU3*VMU3);
        VL1 = sqrt((xa − xb)*(xa − xb) + (ya − yb)*(ya − yb) + (za − zb)*(za − zb));
        IT = 2;
        du2 = 0;
        dv2 = 0;
        dw2 = 0;
        LNDBEQ(VMU2, xpCG, ypCG, zpCG, VL1, du2, dv2, dw2, IT);
        IT = 3;
        du3 = 0;
        dv3 = 0;
        dw3 = 0;
        LNDBEQ(VMU3, xpCG, ypCG, zpCG, VL1, du3, dv3, dw3, IT);
        duG = du2 + du3;
        dvG = dv2 + dv3;
        dwG = dw2 + dw3;
        cxG = 0;
        Temp = sqrt(cyG*cyG + czG*czG);
        cyG = cyG/Temp;
        czG = czG/Temp;
        IC = 2;
        CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, du, dv, dw, duG, dvG, dwG, IC);
        CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, cxR, cyR, czR, cxG, cyG, czG, IC);
        return;
}//LNDBGN

void LNVXGN(float xa, float ya, float za, float xb, float yb, float zb, float Circ, float xp,
                                                float yp, float zp, float du, float dv, float dw) {
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Generates incremental velocity due to a line vortex
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
        float Temp, xaO, xbO, xpO, yaO, ybO, ypO, zaO, zbO, zpO, IC, xpCG, ypCG, zpCG, VL1, duG,
                                dvG, dwG;
        Temp = (xa + xb)/2;
        xaO = xa − Temp;
        xbO = xb − Temp;
        xpO = xp − Temp;
        Temp = (ya − yb)/2;
        yaO = ya − Temp;
        ybO = yb − Temp;
        ypO = yp − Temp;
        Temp = (za − zb)/2;
        zaO = za − Temp;
        zbO = zb − Temp;
        zpO = zp − Temp;
        IC = 1;
        xpCG = 0;
        ypCG = 0;
        zpCG = 0;
        CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, xpO, ypO, zpO, xpCG, ypCG, zpCG, IC);
        VL1 = sqrt((xa − xb)*(xa − xb) + (ya − yb)*(ya − yb) + (za − zb)*(za − zb));
        duG = 0;
        dvG = 0;
        dwG = 0;
        LNVXEQ(Circ, xpCG, ypCG, zpCG, VL1, duG, dvG, dwG);
        IC = 2;
        CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, du, dv, dw, duG, dvG, dwG, IC);
        return;
}//LNVXGN

void LNVXEQ(float Circ, float xpG, float ypG, float zpG, float VL, float duG, float dvG,
                                                float dwG) {
        /*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
        Line vortex equation
        −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
        float yz2, VL2, xpL2, xmL2, dn, dp, Rdn, Rdp, CONST, Temp;
        duG = 0;
        yz2 = ypG*ypG + zpG*zpG;
        VL2 = VL/2;
        xpL2 = xpG + VL2;
        xmL2 = xpG − VL2;
        dn = xmL2*xmL2 + yz2;
```

```
	dp = xpL2*xpL2 + yz2;
	Rdn = sqrt(dn);
	Rdp = sqrt(dp);
	CONST = 0.25*Circ/PI;
	/*Temp = 0.01*VL;*/
	Temp = 0.000001;
	if (yz2 − Temp <= 0) {
		Temp = abs(xpG) − VL2;
		if (Temp <= 0) {
			dvG = 0;
			dwG = 0;
		}//endif
		else {
			Temp = 0.5*CONST*(1/(xmL2*xmL2) − 1/(xpL2*xpL2));
			dvG = −zpG*Temp;
			dwG = ypG*Temp;
		}//else
	}//endif
	else {
		Temp = CONST*(xmL2/Rdn − xpL2/Rdp)/yz2;
		dvG = zpG*Temp;
		dwG = −ypG*Temp;
	}//else
	return;
}//LNVXEQ

void LNSCGN(float xa, float ya, float za, float xb, float yb, float zb, float VM, float xp,
							float yp, float zp, float du, float dv, float dw) {
	/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
	Generates incremental velocities due to a line source
	−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
	float Temp, xaO, xbO, xpO, yaO, ybO, ypO, zaO, zbO, zpO, IC, xpCG, ypCG, zpCG, duG, dvG,
				dwG, VL1;
	Temp = (xa + xb)/2;
	xaO = xa − Temp;
	xbO = xb − Temp;
	xpO = xp − Temp;
	Temp = (ya + yb)/2;
	yaO = ya − Temp;
	ybO = yb − Temp;
	ypO = yp − Temp;
	Temp = (za + zb)/2;
	zaO = za − Temp;
	zbO = zb − Temp;
	zpO = zp − Temp;
	IC = 1;
	xpCG = 0;
	ypCG = 0;
	zpCG = 0;
	CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, xpO, ypO, zpO, xpCG, ypCG, zpCG, IC);
	VL1 = sqrt((xa − xb)*(xa − xb) + (ya − yb)*(ya − yb) + (za − zb)*(za − zb));
	duG = 0;
	dvG = 0;
	dwG = 0;
	LNSCEQ(VM, xpCG, ypCG, zpCG, VL1, duG, dvG, dwG);
	IC = 2;
	CRDTEG(xaO, yaO, zaO, xbO, ybO, zbO, du, dv, dw, duG, dvG, dwG, IC);
	return;
}//LNSCGN

void LNSCEQ(float VM, float xpG, float ypG, float zpG, float VL, float duG, float dvG,
							float dwG) {
	/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
	Line source equation
	−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
	float yz2, VL2, xpL2, xmL2, dn, dp, Rdn, Rdp, CONST, Temp;
	yz2 = ypG*ypG + zpG*zpG;
	VL2 = VL/2;
	xpL2 = xpG + VL2;
	xmL2 = xpG − VL2;
	dn = xmL2*xmL2 + yz2;
	dp = xpL2*xpL2 + yz2;
	Rdn = sqrt(dn);
```

```
		Rdp = sqrt(dp);
		CONST = 0.25*VM/PI;
		Temp = 0.01*VL;
		if (yz2 − Temp <= 0) {
			Temp = abs(xpG) − VL2;
			if (Temp <= 0) {
				duG = 0;
				dvG = 0;
				dwG = 0;
			}//endif
			else {
				duG = CONST*(1/Rdn − 1/Rdp);
				Temp = 0.5*CONST*(1/(xmL2*xmL2) − 1/(xpL2*xpL2));
				dvG = ypG*Temp;
				dwG = zpG*Temp;
			}//else
		}//endif
		else {
			duG = CONST*(1/Rdn − 1/Rdp);
			Temp = CONST*(xmL2/Rdn − xpL2/Rdp)/yz2;
			dvG = −ypG*Temp;
			dwG = −zpG*Temp;
		}//else
		return;
}//LNSCEQ

void CRDTEG(float xaO, float yaO, float zaO, float xbO, float ybO, float zbO, float xpO,
							float ypO, float zpO, float Exip, float Etap, float Ztap, float
IC) {
		/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
		Coordinate translation"English to Greek" if IC = 1
										"Gre
ek to English" if IC = 2
		Origin of the coordinate system will be at the midpoint of A−B
		Transforms (x,y,z) set of axes to (Exi,Eta,Zta) axes with the Exi axis parallel to the center
			line of the doublet
		Parameters:
			A, B			:		Ends of the line doublet
			P					:		General point in space
		Calculated Values:
			EPSI, ALFA		:		Angles of rotation of the Eta (y) and  Zta (z) axes, respectively
		NOTE: System set up for POSITIVE WINDING numbers for rotation of axes
		−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
		float Temp, CEPSI, SEPSI, xaP, xbP, SALFA, CALFA, xpP, ypP, zpP;
		Temp = sqrt((xaO − xbO)*(xaO − xbO) + (yaO − ybO)*(yaO − ybO));
		if (Temp > 0) {
			CEPSI = (xaO − xbO)/Temp;
			SEPSI = (yaO − ybO)/Temp;
			xaP = xaO*CEPSI + yaO*SEPSI;
			xbP = xbO*CEPSI + ybO*SEPSI;
		}//endif
		else {
			CEPSI = 1;
			SEPSI = 0;
			xaP = xaO;
			xbP = xbO;
		}//else
		Temp = sqrt((xaP − xbP)*(xaP − xbP) + (zaO − zbO)*(zaO − zbO));
		if (Temp > 0) {
			SALFA = (zbO − zaO)/Temp;
			CALFA = (xaP − xbP)/Temp;
		}//endif
		else {
			CALFA = 1;
			SALFA = 0;
		}//else
		if (IC − 1 <= 0) {
			xpP = xpO*CEPSI + ypO*SEPSI;
			ypP = ypO*CEPSI − xpO*SEPSI;
			zpP = zpO;
			Exip = xpP*CALFA − zpP*SALFA;
			Etap = ypP;
			Ztap = xpP*SALFA + zpP*CALFA;
		}//endif
		else {
```

```
			xpP = Exip*CALFA + Ztap*SALFA;
			ypP = Etap;
			zpP = Ztap*CALFA − Exip*SALFA;
			xpO = xpP*CEPSI − ypP*SEPSI;
			ypO = xpP*SEPSI + ypP*CEPSI;
			zpO = zpP;
		}//else
		return;
}//CRDTEG

void GJRV(float A[], float N, float NL, float EPsil, float Ierr) {
		/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−−
		Matrix inversion routine
		Parameters:
			A					:		Input array which will be destroyed
			N					:		Rank of A
			NL				:		Row dimension of A
			EPsil		:		Test value for the pivot point singularity check
			Ierr				:		Nonzero if matrix is singular
										If matrix is nonsingular, A contains A−inverse
		−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
−−*/
		float B[96], C[96], IP[96], IQ[96];
		int i, j, k, kp;
		Ierr = 0;
		for (k = 0; k < N; k++) {
			float Pivot = 0;
			for (i = k; i < N; k++) {
				for (j = k; j < N; j++) {
					int IDex = (j − 1)*NL + i;
					if (abs(A[IDex]) − abs(Pivot) > 0) {
						Pivot = A[IDex];
						IP[k] = i;
						IQ[k] = j;
					}//endif
				}//for
			}//for
			if (abs(Pivot) − EPsil <= 0) {
				Ierr = −1;
				return;
			}//endif
			else {
				if (IP[k−1] − k != 0) {
					for (j = 0; j < N; j++) {
						float IPx = IP[k−1];
						int IDex = (j − 1)*NL + IPx;
						int KDex = (j − 1)*NL + k;
						float Z = A[IDex];
						A[IDex] = A[KDex];
						A[KDex] = Z;
					}//for
				}//endif
				if (IQ[k−1] − k != 0) {
					for (i = 0; i < N; i++) {
						float IPx = IQ[k];
						int IDex = (IPx − 1)*NL + i;
						int KDex = (k − 1)*NL + i;
						float Z = A[IDex];
						A[IDex] = A[KDex];
						A[KDex] = Z;
					}//for
				}//endif
				for (j = 0; j < N; j++) {
					int KDex = (j − 1)*NL + k;
					int JDex = (k − 1)*NL + j;
					if (j − k == 0) {
						B[j] = 1/Pivot;
						C[j] = 1;
					}//endif
					else {
						B[j] = −A[KDex]/Pivot;
						C[j] = A[JDex];
					}//else
					A[KDex] = 0;
					A[JDex] = 0;
				}//for
```

```
                        for (i = 0; i < N; i++) {
                                for (j = 0; j < N; j++) {
                                        int IDex = (j − 1)*NL + i;
                                        A[IDex] = A[IDex] + C[i]*B[j];
                                }//for
                        }//for
                }//else
        }//for
        for (kp = N−1; kp >= 0; kp−−) {
                if (IP[k] − k != 0) {
                        for (i = 0; i < N; i++) {
                                float IPx = IP[kp];
                                int IDex = (IPx − 1)*NL + i;
                                int KDex = (k − 1)*NL + i;
                                float Z = A[IDex];
                                A[IDex] = A[KDex];
                                A[KDex] = Z;
                        }//for
                }//endif
                if (IQ[k] − k != 0) {
                        for (j = 0; j < N; j++) {
                                float IPx = IQ[k];
                                int IDex = (j − 1)*NL + IPx;
                                int KDex = (j − 1)*NL + k;
                                float Z = A[IDex];
                                A[IDex] = A[KDex];
                                A[KDex] = Z;
                        }//for
                }//endif
        }//for
        return;
}//GJRV
```