Appendix E

FORTRAN 95 Codes

VIX Head Files and Definitions

```
       parameter(NQA = 300)
       type node
           integer id
              real:: x,y,z
              real:: s,t,TAU
              real:: Up,U,Cp,Cpe
              real:: Dxt2,Dyt2,Dzt2
              real:: Dxs2,Dys2,Dzs2
              real:: Dps2,Dpt2
        end type node

       type section
            integer:: id,Nnd,Nlower
            integer:: istag,iblte(2),Nbl(2)
            integer,dimension(NqA,2):: IBL
            integer,dimension(2):: imatch,Nmatch,tr
            real:: sc,disp,leng,Vchord(3),aflow
            real,dimension(2):: xsep,xre,xtr
            real,dimension(NQA)::   s,t,curv
            real,dimension(3,NQA):: P,Vn
!          BOUNDARY LAYER VARS
            real,dimension(NQA,2):: TAU,UINV
            real,dimension(NQA):: Up,Cp
            real,dimension(4,NQA,2):: Q
              real:: cli,clv,cd,Cdf,Cdp
        end type section
!______________________________________________________________
!   NODE
!      id     -> pt identity
!      x,y,z  -> coordinates of node
!     Cp     -> pressure coefficient
!     Cpe    -> viscous pressure coefficient
!     U      -> absolute velocity
!    TAU     -> shear stress
!    s,t     -> surface coordinates
! Dxt2,Dyt2,Dzt2 -> spline 2nd derivatives on parametric
!                   t direction for each coordinate
! Dxs2,Dys2,Dzs2 -> spline 2nd derivatives on parametric
!                   s direction for each coordinate
!    Dps2,Dpt2    -> spline second derivative on parametric
!                   directions s and t for nodal Cp
!______________________________________________________________
!   SECTION
!     id        -> identity of section
!     istag     -> location of stagnation point
!     Nlower    -> index where leading edge is
!     Npt       -> No of points on section (max=100)
!     ite       -> index where trailing edge lives
!     p(..)     -> coordinates points
!     s(.)      -> curve coordinate
!     t(.)      -> spanwise curve coordinate
!     Vn(..)    -> normal vector on a section node
!     imatch(.) -> point of beginning of matching surface
```

```
!     Nmatch(.) -> point where ends matching surface
!     tr(.)     -> point where transition occurs
!     disp      -> displacement of section on x axis
!     sc        -> scale of section
!     leng      -> chord length
!     aflow     -> flow incidence on the particular section
!     Vchord    -> unitary vector of chord
!  xsep,ysep    -> point of separation  1 - upper
!                                       2 - lower
!   xtr,ytr     -> point of transition
!        BOUNDARY LAYER
!   Q      -> vector of boundary layer variables
!             1 - nr or Ct
!             2 - th
!             3 - dels
!             4 - Ue
!   Cli   -> inviscid lift coefficient
!   Clv   -> viscous lift coefficient
!   Cd    -> drag coefficient
!   Ue    -> edge velocity modulus (viscous)
!   Up    -> edge velocity modulus (potential)
!*******************************************************************
!   Program created by Augusto Veiga (University of Southampton 2003) *
!*******************************************************************

       type panel
           integer:: id,np,ibc
              integer,dimension(4)::ngb
              character(5):: tipo
              real:: s,t,area,fl,fd
              type (node),dimension(4):: nd,mid
              type (node):: co
              real:: Mx,My,Mz
              real:: cp,Up,rpv
              real:: u(3),vm(3)
        end type panel

!
!  PANEL
!      id     -> pt identity
!     ibc    -> panel type index:
!               -1  trailing edge panel
!                1  body panel
!                2  wake attached to trailing edge
!                3  free wake panels
!                4  fixed wake
!     ngb    -> neighbouring panels       (they are 4)
!            ngb(i)= -4  reflection plane
!            ngb(i)= -2  discontinuous
!
!      x,y,z  -> coordinates of node
!     Cp     -> pressure coefficient
!     U      -> absolute velocity
!     co     -> collocation pt of panel
!     mid    -> mid edge between nodes (follows right hand rule)
!     vm     -> vector velocity in m/s (not that useful)
!     nd     -> panel nodes (they are 4)
!     area   -> area of panel
!     fl     -> lifting force of panel
!     fd     -> drag force of panel
!     rpv    -> viscous pressure resistence
!   Mx,My,Mz -> momentum in relation to origin of root chord
!
!*******************************************************************
!   Program created by Augusto Veiga (University of Southampton 2003) *
!*******************************************************************
```

Two-Dimensional Panel Method

```
      subroutine panel (X,Y,IFLAG,Nlower
     &  ,Nupper,Nodtot,Nmax,V,alpha,
```

```
     &  naca,tau)

     integer Nmax
     real,dimension(Nmax)::x,y,xmid,ymid,costhe,sinthe
     real,dimension(Nmax)::V,CP

     real:: alpha,Tau,gamma,CD,CL1,CL2
     integer:: Nodtot,Nupper,Nlower,IFLAG


      real PI, PI2INV, XNU,cosalf,sinalf,thick,camber,beta
      real, dimension(Nodtot,Nodtot+1) :: A
      integer N

c................................................ Begin
c
      PI = 4. * atan(1.)
      PI2INV = 1. / (2. * PI)
      V = 0
      IF (ALPHA.gt.PI/2.) GOTO 400
!      if (ALPHA.gt.90.) goto 400
!      close (5)
c
c
c............................................ Initializing data
c
      print *, '*** Inicializacao dos dados - aguarde ... '
      call SETUP(x,y,xmid,ymid,costhe,sinthe,cosalf,sinalf,PI,
     &     PI2INV,Nodtot,Nmax,Nupper,Nlower,alpha,tau,
     &     NACA,IFLAG,XNU)
c
      COSALF = cos(ALPHA)   !*PI/180.)
      SINALF = sin(ALPHA)   !*PI/180.)

      A=0
c
c
c..................... Influence coefficient matrix assembly
c
      print *, '*** Montagem da matriz de coeficientes - aguarde ... '
      call COFISH(x,y,xmid,ymid,costhe,sinthe,Nodtot,pi,pi2inv,
     &   alpha,cosalf,sinalf,Nmax,A)
c
c................ Gauss Elimination solution system
c
      print *, '*** Solucao do sistema de equacoes - aguarde ... '
      call GAUSS2(A,Nodtot-1,1,Nodtot)
c
c
c................ Velocity and pressure coefficient
c
      print *, '*** Calculo das velocidades e pressoes - aguarde ... '
      call VELDIS(A,x,y,Nlower,Nupper,Nodtot,xmid,ymid,V,CP,gamma,
     &   Nmax,XNU)

c
c............................... Calculo dos coeficientes do aerofolio
c
      print *, '*** Calculo dos coefs. adimensionais - aguarde ... '
      call FANDM(A,x,y,Nlower,Nupper,Nodtot,xmid,ymid,V,CP,gamma,
     &   Nmax,sinalf,cosalf,CD,CL1,CL2)

  400 write (*,9999)
      !stop
 9999 format (//, ' End of panel method - Univ. Southampton/COPPE(C)
     & 2001')

c
c-------------------------------------------- Fim do programa principal
c
      end subroutine
```

```
cÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ SUBROUTINE  ³   SETUP   ³
c                                                          ÀÄÄÄÄÄÄÄÄÄÄÄÙ
c
      subroutine SETUP(x,y,xmid,ymid,costhe,sinthe,cosalf,sinalf,PI,
     &     PI2INV,Nodtot,Nmax,Nupper,Nlower,alpha,tau,
     &     NACA,IFLAG,XNU)
c
      integer Nmax
     real,dimension(Nmax)::x,y,xmid,ymid,costhe,sinthe
     real,dimension(Nmax)::V,CP

     real alpha,Tau,thick,camber,beta
     real Z
     integer Nodtot,Nupper,Nlower,IFLAG,SIGN


      real PI, PI2INV, XNU
     integer N,NACA
     real cosalf,sinalf

      XNU= .89292E-06
      if (IFLAG.ne.1) go to 120
      NPOINTS = NLOWER
      SIGN = -1.
      NSTART = 0
      do 110 NSURF=1,2
         do 100 N=1,NPOINTS
            FRACT = float(N-1) / float(NPOINTS)
            Z = .5 * (1. - cos(PI * FRACT))
            I = NSTART + N
            call BODY(Z,alpha,NACA,tau,Nlower,Nupper,SIGN,
     &     beta,X(I),Y(I))
  100    continue
         NPOINTS = NUPPER
         SIGN = 1.
         NSTART = NLOWER
  110 continue
c
c
c................................ panel slope
c
120   do 200 I=1,Nodtot-1
         DX = X(I+1) - X(I)
         DY = Y(I+1) - Y(I)
         DIST = sqrt(DX * DX + DY * DY)
         SINTHE(I) = DY / DIST
         COSTHE(I) = DX / DIST
  200 continue

c
c
c................................ collocation pts
c
      do 300 I=1,Nodtot-1
         XMID(I) = .5 * (X(I) + X(I+1))
         YMID(I) = .5 * (Y(I) + Y(I+1))
  300 continue

      return
c
c
c------------------------------------------------ End SETUP
c
      end
c
c                                                          ÚÄÄÄÄÄÄÄÄÄÄ¿
cÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ SUBROUTINE  ³   BODY   ³
c                                                          ÀÄÄÄÄÄÄÄÄÄÄÙ
c
      subroutine BODY (Z,alpha,NACA,tau,Nlower,Nupper,SIGN,
     & beta,xi,yi)
c
```

```
c ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ Descricao dos Parametros ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
c ³                                                                    ³
c ³   Z     - parametro de espacamento nodal (entrada)                ³
c ³   SIGN  - identificador da superficie: +1 - superficie superior    ³
c ³                                         -1 - superficie inferior    ³
c ³           (entrada)                                                 ³
c ³   X     - coordenada cartesiana X (entrada)                        ³
c ³   Y     - coordenada cartesiana Y (entrada)                        ³
c ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
     real alpha,Xi,Yi,tau,epsmax,ptmax,thick,camber,beta
     real Z
     integer Nlower, Nupper,Nmax,SIGN
      if (SIGN.lt.0.) Z = 1. - Z
      call NACA45(Z,tau,NACA,epsmax,ptmax,alpha,thick,camber)
      Xi = Z - SIGN * THICK * sin(BETA)
      Yi = CAMBER + SIGN * THICK * cos(BETA)
      return
c
c
c------------------------------------------------ End BODY
c
      end
c
c                                                          ÚÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
cÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ SUBROUTINE  ³   COFISH   ³
c                                                          ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
c
      subroutine COFISH(x,y,xmid,ymid,costhe,sinthe,Nodtot,pi,pi2inv,
     &   alpha,cosalf,sinalf,Nmax,A)
c
c
c ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ Descricao dos Parametros ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
c ³                                                                    ³
c ³   SINALF - valor de sin(Alpha) (entrada)                           ³
c ³   COSALF - valor de cos(Alpha) (entrada)                           ³
c ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
c
     integer Nmax,Nodtot,N
     real,dimension(Nmax)::x,y,xmid,ymid,costhe,sinthe
     real,dimension(Nodtot,Nodtot+1)::A
     real alpha,Tau,Epsmax,ptmax,cosalf,sinalf,pi,pi2inv

      N = NODTOT-1
      do 120 I=1,N
c
c.......................... panel contribution
c
         do 110 J=1,N
            FTAN = PI
            if (J.eq.I) goto 100
            DXJ = XMID(I) - X(J)
            DXJP = XMID(I) - X(J+1)
            DYJ = YMID(I) - Y(J)
            DYJP = YMID(I) - Y(J+1)
            FTAN = atan2(DYJP*DXJ-DXJP*DYJ,DXJP*DXJ+DYJP*DYJ)
  100       A(I,J) = FTAN * PI2INV
c
c.......... Kutta condition at trailing edge
c
            if (J.eq.1) then
               A(I,J) = A(I,J) - PI2INV * atan(YMID(I)/(1.-XMID(I)))
            endif
            if (J.eq.N) then
               A(I,J) = A(I,J) + PI2INV * atan(YMID(I)/(1.-XMID(I)))
            endif
            if (I.eq.J) A(I,J) = A(I,J) - 1.
  110    continue
c
c...................................... Free vars
c
         A(I,N+1) = - (XMID(I) * COSALF + YMID(I) * SINALF)
  120 continue
```

```
c
      return
c
c
c----------------------------------------------- End COFISH
c
      end
c
c                                                              ÚÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
cÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ SUBROUTINE  ³   VELDIS   ³
c                                                              ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
c
      subroutine VELDIS(A,x,y,Nlower,Nupper,Nodtot,xmid,ymid,V,CP,gamma,
     &   Nmax,XNU)
      real gamma
     integer Nmax
     real,dimension(Nmax)::x,y,xmid,ymid,costhe,sinthe,V,CP,FI,CF
     real,dimension(Nodtot,Nodtot+1)::A
     real alpha,Tau,Epsmax,ptmax,cosalf,sinalf,pi,pi2inv,XNU
     integer Nodtot,N

     N=Nodtot-1  !number of panels

      do 50 I=1,N
         FI(I) = A(I,N+1)
   50 continue
   GAMMA = FI(N) - FI(1)
c
c
c............. tangential velocity and pressure
c
c................................................. lower surface
c

      do 100 K=1,NLOWER-1
         if (K.eq.1.or.K.eq.NLOWER-1) then
            if (K.eq.1) then
               XK1 = .5 * (X(K) + X(K+1))
               YK1 = .5 * (Y(K) + Y(K+1))
               XK2 = .5 * (X(K+1) + X(K+2))
               YK2 = .5 * (Y(K+1) + Y(K+2))
               XK3 = .5 * (X(K+2) + X(K+3))
               YK3 = .5 * (Y(K+2) + Y(K+3))
               S1 = 0.
               F1 = FI(K)
               S2 = sqrt((XK2 - XK1)**2 + (YK2 - YK1)**2)
               F2 = FI(K+1)
               S3 = S2 + sqrt((XK3 - XK2)**2 + (YK3 - YK2)**2)
               F3 = FI(K+2)
            endif
            if (K.eq.NLOWER-1) then
               XK1 = .5 * (X(K-2) + X(K-1))
               YK1 = .5 * (Y(K-2) + Y(K-1))
               XK2 = .5 * (X(K-1) + X(K))
               YK2 = .5 * (Y(K-1) + Y(K))
               XK3 = .5 * (X(K) + X(K+1))
               YK3 = .5 * (Y(K) + Y(K+1))
               S3 = 0.
               F3 = FI(K)
               S2 = - sqrt((XK3 - XK2)**2 + (YK3 - YK2)**2)
               F2 = FI(K-1)
               S1 = S2 - sqrt((XK2 - XK1)**2 + (YK2 - YK1)**2)
               F1 = FI(K-2)
            endif
         else
            XK1 = .5 * (X(K-1) + X(K))
            YK1 = .5 * (Y(K-1) + Y(K))
            XK2 = .5 * (X(K) + X(K+1))
            YK2 = .5 * (Y(K) + Y(K+1))
            XK3 = .5 * (X(K+1) + X(K+2))
            YK3 = .5 * (Y(K+1) + Y(K+2))
            S1 = - sqrt((XK2 - XK1)**2 + (YK2 - YK1)**2)
```

```
            F1 = FI(K-1)
            S2 = 0.
            F2 = FI(K)
            S3 = sqrt((XK3 - XK2)**2 + (YK3 - YK2)**2)
            F3 = FI(K+1)
         endif
         DELTA = (S3 - S1) * (S2 - S1) * (S2 - S3)
         DELTB = (S2**2 - S1**2) * (F3 - F1)
     *         - (S3**2 - S1**2) * (F2 - F1)
         V(K) = DELTB / DELTA    !- DELTB / DELTA
         CP(K) = 1. - V(K) * V(K)
!        CF(K) = 0.075/(log(V(K)/XNU) -2.0)**2
  100 continue
c
c
c.............................................. upper surface
c
      do 110 K=Nlower,NUPPER-1
         L = k       !    NLOWER
         if (K.eq.Nlower.or.K.eq.NUPPER-1) then
            if (K.eq.Nlower) then
               XK1 = .5 * (X(L) + X(L+1))
               YK1 = .5 * (Y(L) + Y(L+1))
               XK2 = .5 * (X(L+1) + X(L+2))
               YK2 = .5 * (Y(L+1) + Y(L+2))
               XK3 = .5 * (X(L+2) + X(L+3)) ! x e y com o nº de nós
               YK3 = .5 * (Y(L+2) + Y(L+3))
               S1 = 0.
               F1 = FI(L)   !Fi varia com o nº de painéis
               S2 = sqrt((XK2 - XK1)**2 + (YK2 - YK1)**2)
               F2 = FI(L+1)
               S3 = S2 + sqrt((XK3 - XK2)**2 + (YK3 - YK2)**2)
               F3 = FI(L+2)
            endif
            if (K.eq.NUPPER-1) then
               XK1 = .5 * (X(L-2) + X(L-1))
               YK1 = .5 * (Y(L-2) + Y(L-1))
               XK2 = .5 * (X(L-1) + X(L))
               YK2 = .5 * (Y(L-1) + Y(L))
               XK3 = .5 * (X(L) + X(L+1))
               YK3 = .5 * (Y(L) + Y(L+1))
               S3 = 0.
               F3 = FI(L)
               S2 = - sqrt((XK3 - XK2)**2 + (YK3 - YK2)**2)
               F2 = FI(L-1)
               S1 = S2 - sqrt((XK2 - XK1)**2 + (YK2 - YK1)**2)
               F1 = FI(L-2)
            endif
         else
            XK1 = .5 * (X(L-1) + X(L))
            YK1 = .5 * (Y(L-1) + Y(L))
            XK2 = .5 * (X(L) + X(L+1))
            YK2 = .5 * (Y(L) + Y(L+1))
            XK3 = .5 * (X(L+1) + X(L+2))
            YK3 = .5 * (Y(L+1) + Y(L+2))
            S1 = - sqrt((XK2 - XK1)**2 + (YK2 - YK1)**2)
            F1 = FI(L-1)
            S2 = 0.
            F2 = FI(L)
            S3 = sqrt((XK3 - XK2)**2 + (YK3 - YK2)**2)
            F3 = FI(L+1)
        endif
         DELTA = (S3 - S1) * (S2 - S1) * (S2 - S3)
         DELTB = (S2**2 - S1**2) * (F3 - F1)
     *         - (S3**2 - S1**2) * (F2 - F1)
         V(L) = DELTB / DELTA
         CP(L) = 1. - V(L) * V(L)
!        CF(L) = 0.075/(log(V(L)/XNU) -2.0)**2
  110 continue
c
      return
c
```

```
c
c------------------------------------------------ End VELDIS
c
      end
c
c                                                                   ÚÄÄÄÄÄÄÄÄÄÄÄ¿
cÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ SUBROUTINE  ³   FANDM   ³
c                                                                   ÀÄÄÄÄÄÄÄÄÄÄÄÙ
c
      subroutine FANDM(A,x,y,Nlower,Nupper,Nodtot,xmid,ymid,V,CP,gamma,
     &   Nmax,sinalf,cosalf,CD,CL1,CL2)
     integer Nmax,Nodtot,N
      real gamma,sinalf,cosalf,CD,CL1,CL2
     real,dimension(Nmax)::x,y,xmid,ymid,costhe,sinthe,V,CP
     real,dimension(Nodtot,Nodtot+1)::A
     real alpha,Tau,Epsmax,ptmax,pi,pi2inv
     !integer Nodtot,N

      CM = 0.
      do 100 I=1,NODTOT-1
         DX = X(I+1) - X(I)
         DY = Y(I+1) - Y(I)
         CM = CM + CP(I) * (DX * XMID(I) + DY * YMID(I))
  100 continue
      CD=0.0
      CL1=0.0
      CL2 = 2. * GAMMA
      return
c
c
c------------------------------------------------- End FANDM
c
      end subroutine
c
c                                                                  ÚÄÄÄÄÄÄÄÄÄÄÄÄ¿
cÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ SUBROTINA  ³   NACA45   ³
c                                                                  ÀÄÄÄÄÄÄÄÄÄÄÄÄÙ
c
      subroutine NACA45(Z,tau,NACA,epsmax,ptmax,alpha,thick,camber)
     real tau,epsmax,ptmax,thick,camber,beta,alpha
     integer NACA
     real Z
      THICK = 0.
      if (Z.lt.1.e-10) goto 100
      THICK = 5. * TAU * (.2969 * sqrt(Z) - Z * (.126 + Z * (.3537
     *             - Z * (.2843 - Z * .1015))))
c
  100 if (EPSMAX.eq.0.) goto 130
      if (NACA.gt.9999) goto 140
      if (Z.gt.PTMAX) goto 110
c
      CAMBER = EPSMAX / PTMAX / PTMAX * (2 * PTMAX - Z) * Z
      DCAMDX = 2. * EPSMAX / PTMAX / PTMAX * (PTMAX - Z)
      goto 120
c
  110 CAMBER = EPSMAX / (1. - PTMAX)**2 * (1. + Z - 2. * PTMAX)
     *         * (1. - Z)
      DCAMDX = 2. * EPSMAX / (1. - PTMAX)**2 * (PTMAX - Z)
c
  120 BETA = atan(DCAMDX)
      return
c
  130 CAMBER = 0.
      BETA = 0.
      return
c
  140 if (Z.gt.PTMAX) goto 150
      W = Z / PTMAX
      CAMBER = EPSMAX * W *((W - 3.) * W + 3. - PTMAX)
      DCAMDX = EPSMAX * 3. * W * (1. - W) / PTMAX
      goto 120
c
```

```
  150 CAMBER = EPSMAX * (1. - Z)
      DCAMDX = - EPSMAX
      goto 120
c
c
c----------------------------------------------- End NACA45
c
      end
```

VIX 3D Main Code

```
     Program VII3d
      !use AVDef
     !use DFLib

!     This program reads the streamlines given by PALISUPAN
!     as object str where str has the properties
!     id        -> identity of streamline
!     Npt       -> No of points on streamline (max=100)
!     p(..)     -> coordinates of points
!     u(.)      -> stream velocity on each point
!     V(..)     -> vector of velocities
!     imatch(.) -> point of beginning of matching surface
!     Nmatch(.) -> point where ends matching surface
!     tr(.)     -> point where transition occurs
!     Nsec      -> number of sections (z cte)

!     flags     -> false if Xsep=0
!                  true, otherwise
!
!     With data on streamlines, the program calculates:
!     - influence matrix for each streamline
!     - boundary layer var distribution
!     - make a Newton-Raphson solver for Lag entrainment method
!     - make viscous corrections for potential stream velocity
!     - print out viscous flow characteristics for each velocity
!*********************************************************************
!*     Program created by Augusto Elisio Lessa Veiga               *
!*     FSIG - University of Southampton/2003                        *
!*     Sugestions are welcome                                       *
!*********************************************************************
     include 'section.inc'
     include 'panel.inc'
     include 'xfoil.inc'

     integer Nstr,Npan,Npb,Nmax,Nte,Nsec
     real:: Rey,visc,alpha,Dlimit,pi,EPS1
     real,dimension(3):: tol
     real,dimension(2,150):: xsep,Xre,Xtr
     type (node),allocatable,dimension(:):: nd,bnd,wnd
     type (section),allocatable,dimension(:):: sec,wsec
     type (panel),allocatable,dimension(:):: pan,bpan,wpan
     character*70 :: arqname,arqname2
     logical:: fl,inviscid,fsharp
     BIJ = 0
     CIJ = 0
     DIJ = 0
     ! pflag  -> flag that indicates if it's to apply
     !           wall pressure correction
      pi = 4. * atan(1.)
     call read_set(arqname,arqname2,Rey,visc,alpha,ACRIT,
     &                 Nit,Nsec,Nmax,
     &              xtr,TFORCE,VACCEL,EPS1,fl,inviscid,fsharp)
     alpha = alpha*pi/180
     ALFA  = alpha
      open(1,file ='BL_log.txt')
       write(1,*) 'New problem'
      close(1)

     if (inviscid) then
        call read_prev(Nsec,Nmax,Nw)
```

```
   allocate (sec(Nsec),wsec(Nsec))
   call read_section(sec,wsec,Nsec,Nmax,Nw)
 write(*,*) '****************************************************'
write(*,*) '*                    V  I  X                        *'
write(*,*) '*        Copyright, Augusto E. L. Veiga             *'
write(*,*) '*        University of Southampton, 2004            *'
write(*,*) '*                       Version 1.0                 *'
write(*,*) '*****************************************************'
write(*,*)
write(*,*) 'previously interpolated sections...'
    N = Nmax
 else
  call read_N(arqname,Nt,Npan)  !reads the number of sections
                                           !and nodes
  allocate(nd(Nt))            !allocate vector str
  allocate(pan(Npan),sec(Nsec),wsec(Nsec))
   !allocate(sec(Nsec))
   !initialize structure sec
  call read_nodes(arqname,nd,pan,Nt,Npan) !read body nodes
  if (mod(nmax,2)==0) then
     Nmax=Nmax+1
  Else
     Nmax=Nmax
  endif
  call read_uns(arqname2,pan,Npan,nd,nt,Nte)
  call find_wk(pan,Npan,Npw,Iwake)
  allocate(wpan(Npw))
! sections are interpolated on this routine
 write(*,*) '****************************************************'
write(*,*) '*                    V  I  X                        *'
write(*,*) '*        Copyright, Augusto E. L. Veiga             *'
write(*,*) '*        University of Southampton, 2004            *'
write(*,*) '*                       Version 1.0                 *'
write(*,*) '*****************************************************'
write(*,*)
write(*,*) 'Interpolating sections...'
  call wk_surface(pan,wpan,Npan,Npw,Iwake,wsec,Nsec,fl)
  call surface(pan,Npan,sec,Nsec,Nmax,Nte,fl)
  ! there is no need anymore for such heavy structure
  deallocate(nd,pan,wpan)
   N  = sec(1).Nnd
  Nw = wsec(1).Nnd
endif
 write(*,*)
write(*,*) '...calculating viscous flow'
!i=int(Nsec/2)
do i=1,Nsec
  !calculate geometric curvature for each node
  call G_curv(sec(i),Nmax,pflag)
  !find very first point on leading edge
  call find_Nlower(sec(i),Nmax)
  REINF = Rey
  QINF = 1.0

   XSTRIP(1) = xtr(1,i) !localizes transition upper part
  XSTRIP(2) = xtr(2,i) !lower part
  XTE = sec(i).p(1,1)
  XLE = sec(i).p(1,sec(i).nlower)
  YTE = sec(i).p(2,1)
  YLE = sec(i).p(2,sec(i).nlower)
  SLE = sec(i).s(sec(i).nlower)
  !fill up vars for xlib library
  call secset(sec(i),wsec(i),N,Nw,QINV,x,y,s)
  !ALFA = -sec(i).aflow
   open(2,file ='BL_log.txt',position = 'APPEND')
     write(2,10) i
  close(2)
    ! This part solves the viscous flow for each section
  call VIX(sec(i),wsec(i),Nmax,EPS1,NQX,Nit,Nsec,i,fsharp)
   sec(i).tr(1) = ITRAN(1)
   sec(i).tr(2) = ITRAN(2)
   sec(i).Nbl(1) = NBL(1)
```

```
        sec(i).Nbl(2) = NBL(2)
        do is = 1,2
          do ip =2,iblte(is)
            sec(i).ibl(ip,is) = IPAN(ip,is)
          enddo
          sec(i).iblte(is) = iblte(is)
          do in = 1,sec(i).Nbl(is)
            sec(i).TAU(in,is) = TAU(in,is)
          enddo
        enddo
            enddo
      !perform lift, drag and viscous pressure resistance calculations
      !call trefftz(wsec,Nsec,20,Dtrefftz) !calculates inviscid induced drag
      call panmk(wsec,sec,Nsec,Nw,Nmax,CLvis,CLinv,cdf,cd,
      &          cdi,cdiv,area,Cmx,Cmy,Cmz,zcp)
      !call vii_graph(sec,Nsec,Nmax)
       write(*,*)
      write(*,*) '...printing results'
      call print_result(sec,Nsec,Nw,CLvis,CLinv,cdf,cd,
      &                 cdi,cdiv,area,Cmx,Cmy,Cmz,zcp)
      call print_blvar(sec,Nsec,Nmax)
      call post_process(sec,Nsec,Nmax)  !organize sectional plots
      !reorganise panels using the sections again and print
      ! files to be used by PANVISE
      !call makepan
      deallocate(sec,wsec)
 10    format('Section =',i4)
      END

       subroutine VIX(sec,wsec,Nmax,EPS1,NQX,NIT,Nsec,isec,fsharp)
!      This subroutine receives the following variables:
!      Geometry:            x,y,z and s of each section
!      Inviscid flow:       QINV for each section
!                           QINV for wake section
!      Data                 N -> number of section points
!                           Nw-> number of wake section points
!                           Nlower -> leading edge point that
!                                   divides upper and lower parts
!                           tr(.) -> transition point on lower and upper
!                                    parts
!                           Nsep  -> separation points on lower and upper parts
!                           Nre   -> reattachment points on lower and upper parts
!                           isec  ->  index of section
!                                     if isec=1 or isec=Nsec, then viscous flow is not cal
!                                     culated
!       ...And spills out the following:
!       Viscous flow:
!       QVIS        -> viscous velocity
!       Dstr        -> displacement thickness
!       Thet        -> momentum thickness
!       Ctau        -> sqrt (max shear coefficient
!       H           -> shape parameter
!       Cf          -> friction coefficient
!       Dis         -> dissipation coefficient
!
!*****************************************************************************
!       A  T  E  N  T  I  O  N  !  !  !
!
!      Points are input on counterclockwise order and this continues like that.
!      Do not use the inverse order or you may experience problems
!*****************************************************************************
!      This program was modified by Augusto Elisio Lessa Veiga and
!      uses parts of the GNU software XFOIL
!*****************************************************************************
!*    Author: Augusto Elisio Lessa Veiga                                     *
!*            University of Southampton, 2004                                *
!*                         (Made in Brasil)                                  *
!*****************************************************************************
       include 'section.inc'
      include 'xfoil.inc'
      include 'xbl.inc'
```

```
      type(section) :: sec,wsec
      real,dimension(N+Nw):: Up,xbd,sbd
      real,dimension(Nw):: Upw
      real,dimension(3,N):: p
      real,dimension(3,Nw):: pw
      real,dimension(Izx+Iwx):: z
       real:: clsec,cdsec,EPS1
      real,dimension(iqx,2):: pcor
      integer N,Nw,Nlower,Nsec,isec
      logical:: fsharp

      do i = 1,N
         Up(i) = sec.Up(i)
         xbd(i) = X(i)
         sbd(i) = S(i)
      enddo
      j = 0
      do i = N+1,N+Nw
        j = j+1
        Up(i) = wsec.up(j)
      enddo
      !main settings
       pcor = 0 !22/05/2005
       PI = 4.0*ATAN(1.0)
       HOPI = 0.50/PI
       QOPI = 0.25/PI
       DTOR = PI/180.0
C---- default Cp/Cv (air)
       GAMMA = 1.4
       GAMM1 = GAMMA - 1.0
C---- initialize freestream Mach number to zero
       MATYP = 1
       MINF1 = 0.
       MINF  = 0.

       CL = 0.
       CM = 0.
       CD = 0.

       SIGTE = 0.0
       GAMTE = 0.0
       SIGTE_A = 0.
       GAMTE_A = 0.

         SIG = 0.

      SHARP   = .true.  !if trailing edge is sharp
       LIMAGE  = .FALSE. !if image airfoil is present
       LGAMU   = .TRUE.  !if GAMU  arrays exist for current airfoil geometry
       LQINU   = .TRUE.  !if QINVU arrays exist for current airfoil geometry
       LVISC   = .TRUE.  !if viscous option is invoked
       LALFA   = .TRUE.  !if alpha is specifed, .FALSE. if CL is specified
       LWAKE   = .TRUE.  !if wake geometry has been calculated
C   LPACC        .TRUE. if each point calculated is to be saved
       LBLINI  = .FALSE. !if BL has been initialized
       LIPAN   = .TRUE.  !if BL-&gt;panel pointers IPAN have been calculated
C   LQAIJ        .TRUE. if dPsi/dGam matrix has been computed and factored
       LADIJ   = .FALSE.  !if dQ/dSig matrix for the airfoil has been computed
       LWDIJ   = .FALSE.  !if dQ/dSig matrix for the wake has been computed
C   LQVDES       .TRUE. if viscous Ue is to be plotted in QDES routines
C   LQSPEC       .TRUE. if Qspec has been initialized
C   LQREFL       .TRUE. if reflected Qspec is to be plotted in QDES routines
       LVCONV  = .FALSE. !if converged BL solution exists
C   LCPREF       .TRUE. if reference data is to be plotted on Cp vs x/c plots
C   LCLOCK       .TRUE. if source airfoil coordinates are clockwise
C   LPFILE       .TRUE. if polar file is ready to be appended to
C   LPFILX       .TRUE. if polar dump file is ready to be appended to
C   LPPSHO       .TRUE. if CL-CD polar is plotted during point sequence
C   LBFLAP       .TRUE. if buffer  airfoil flap parameters are defined
C   LFLAP        .TRUE. if current airfoil flap parameters are defined
C   LEIW         .TRUE. if unit circle complex number array is initialized
C   LSCINI       .TRUE. if old-airfoil circle-plane arc length s(w) exists
```

```
C   LFOREF      .TRUE. if CL,CD... data is to be plotted on Cp vs x/c plots
C   LNORM       .TRUE. if input buffer airfoil is to be normalized
C   LGSAME      .TRUE. if current and buffer airfoils are identical
C
C   LPLCAM      .TRUE. if thickness and camber are to be plotted
C   LQSYM       .TRUE. if symmetric Qspec will be enforced
C   LGSYM       .TRUE. if symmetric geometry will be enforced
C   LQGRID      .TRUE. if grid is to overlaid on Qspec(s) plot
C   LGGRID      .TRUE. if grid is to overlaid on buffer airfoil geometry plot
C   LGTICK      .TRUE. if node tick marks are to be plotted on buffer airfoil
C   LQSLOP      .TRUE. if modified Qspec(s) segment is to match slopes
C   LGSLOP      .TRUE. if modified geometry segment is to match slopes
C   LCSLOP      .TRUE. if modified camber line segment is to match slopes
C   LQSPPL      .TRUE. if current Qspec(s) in in plot
C   LGEOPL      .TRUE. if current geometry in in plot
C   LCPGRD      .TRUE. if grid is to be plotted on Cp plots
C   LBLGRD      .TRUE. if grid is to be plotted on BL variable plots
C   LBLSYM      .TRUE. if symbols are to be plotted on BL variable plots
C   LCMINP      .TRUE. if min Cp is to be written to polar file for cavitation
C   LHMOMP      .TRUE. if hinge moment is to be written to polar file
C
C   LPGRID      .TRUE. if polar grid overlay is enabled
C   LPCDW       .TRUE. if polar CDwave is plotted
C   LPLIST      .TRUE. if polar listing lines (at top of plot) are enabled
C   LPLEGN      .TRUE. if polar legend is enabled
C
C   LPLOT       .TRUE. if plot page is open
C   LSYM        .TRUE. if symbols are to be plotted in QDES routines
C   LIQSET      .TRUE. if inverse target segment is marked off in QDES
C   LCLIP       .TRUE. if line-plot clipping is to be performed
C   LVLAB       .TRUE. if label is to be plotted on viscous-variable plots
C   LCURS       .TRUE. if cursor input is to be used for blowups, etc.
C   LLAND       .TRUE. if Landscape orientation for PostScript is used

     call stagpoint(Up,sbd,GAM,xbd,N,Nw,IST,SST,SST_GO,SST_GP,fsharp)
      sec.istag = IST

      call SIC(sec,wsec,N,Nw)
!      DATA EPS1 / 1.0E-4 /
C
      NITER = 10
     QINF = 1.0
C
C
C---- set velocities on wake from airfoil vorticity for alpha=0, 90
C      CALL QWCALC
C
C---- set velocities on airfoil and wake for initial alpha
C      CALL QISET
C
C
C----- locate stagnation point arc length position and panel index
!       CALL STFIND
C
C----- set  BL position -> panel position  pointers
       CALL IBLPAN
      sec.iblte(1) = iblte(1)
      sec.iblte(2) = iblte(2)
C
C----- calculate surface arc length array for current stagnation point location
       CALL XICALC
C
C----- set  BL position -> system line  pointers
       CALL IBLSYS
C
C
C---- set inviscid BL edge velocity UINV from QINV
      CALL UICALC
C


      IF(.NOT.LBLINI) THEN
```

```
C
C----- set initial Ue from inviscid Ue
       DO IBL=1, NBL(1)
         UEDG(IBL,1) = UINV(IBL,1)
       ENDDO
C
       DO IBL=1, NBL(2)
         UEDG(IBL,2) = UINV(IBL,2)
       ENDDO
C
      ENDIF
     !initial lift calculation (inviscid)
       Nref = iqx
       call clcalc2(N,Nref,sec,gam_a,alfa,minf,qinf,pcor,
     &               XCMREF,YCMREF,CL,CM,CDP,CL_ALF,CL_MSQ)
!        CALL CLCALC(N,X,Y,GAM,GAM_A,ALFA,MINF,QINF, XCMREF,YCMREF,
!     &              CL,CM,CDP,CL_ALF,CL_MSQ)
      sec.cli = CL
C
C
C---- Newton iteration for entire BL solution
      NITER = Nit
     if (isec>1 .and. isec<Nsec) then

      WRITE(*,*) 'Solving BL system ...'
     open(1,file = 'BL_log.txt',position = 'APPEND')
      DO 1000 ITER=1, NITER
C
C------ fill Newton system for BL variables
        CALL SETBL
C
C------ solve Newton system with custom solver
        CALL BLSOLV
C
C------ update BL variables
        CALL UPDATE(sec,pcor) !output pcor (22/05/2005)
C
C        IF(LALFA) THEN
C------- set new freestream Mach, Re from new CL
C         CALL MRCL(CL,MINF_CL,REINF_CL)
C         CALL COMSET
C        ELSE
C------- set new inviscid speeds QINV and UINV for new alpha
C         CALL QISET
C         CALL UICALC
C        ENDIF
C
C------ calculate edge velocities QVIS(.) from UEDG(..)
        CALL QVFUE
C
C------ set GAM distribution from QVIS
        CALL GAMQV
C
C------ relocate stagnation point
!        CALL STMOVE
C
C------ set updated CL,CD
        !Nref = iqx
       call clcalc2(N,Nref,sec,gam_a,alfa,minf,qinf,pcor,
     &               XCMREF,YCMREF,CL,CM,CDP,CL_ALF,CL_MSQ)
        sec.clv = CL
!        CALL CLCALC(N,X,Y,GAM,GAM_A,ALFA,MINF,QINF, XCMREF,YCMREF,
!     &              CL,CM,CDP,CL_ALF,CL_MSQ)
        CALL CDCALC
       sec.cdf = CDF  !sectional frict. Cd
       sec.cd = CD
!      if (iter==1) then
!        sec.cli = CL
!      endif
C
C------ display changes and test for convergence
        IF(RLX.LT.1.0)
```

```
     &   WRITE(1,2000) ITER, RMSBL, RMXBL, VMXBL,IMXBL,ISMXBL,RLX
        IF(RLX.EQ.1.0)
     &   WRITE(1,2010) ITER, RMSBL, RMXBL, VMXBL,IMXBL,ISMXBL
         CDP = CD - CDF
         WRITE(1,2020) ALFA/DTOR, CL, CM, CD, CDF, CDP
C
        IF(RMSBL .LT. EPS1) THEN
         LVCONV = .TRUE.
         AVISC = ALFA
         MVISC = MINF
         GO TO 90
        ENDIF
C
 1000 CONTINUE
      WRITE(1,*) 'VISCAL:  Convergence failed'
C
   90 CONTINUE
      close(1)

     endif !avoiding tip sections
      !filling up vectors
     sec.clv = CL
      DO IS=1, 2
        DO IBL=2, NBL(IS)
          I = ibl  !IPAN(IBL,IS)
         sec.Q(1,i,is) = Ctau(i,is)   !shear stress or critical amp
         sec.Q(2,i,is) = thet(i,is)   !momentum thick
         sec.Q(3,i,is) = dstr(i,is)   !recording displacement thick
         sec.Q(4,i,is) = uedg(i,is)   !recording viscous velocity
         sec.UINV(i,is) = UINV(i,is)  !recording inviscid velocity
       enddo
       ! wake variables
       if (is==2) then
         iwk=0
         do ibl=iblte(is)+1,iblte(is)+wsec.Nnd
           iwk = iwk+1
           i = ibl
           wsec.Q(1,iwk,is) = Ctau(i,is)   !shear stress or critical amp
           wsec.Q(2,iwk,is) = thet(i,is)   !momentum thick
           wsec.Q(3,iwk,is) = dstr(i,is)   !recording displacement thick
           wsec.Q(4,iwk,is) = uedg(i,is)   !recording viscous velocity
         enddo
       endif
     enddo
     if (isec>1 .and. isec<Nsec) then
       clsec = cl
       cdsec = cd
        sec.CD = CD
       sec.Cdf = Cdf
       sec.Cdp = Cdp
       sec.iter = iter
     else
       clsec = sec.cli
       cdsec = 0
       sec.CD = 0
       sec.Cdf = 0
       sec.Cdp = 0
       sec.iter = 0
     endif
      RETURN
C...................................................................
 2000   FORMAT
     &   (/1X,I3,'   rms: ',E10.4,'   max: ',E10.4,3X,A1,' at ',I4,I3,
     &      '   RLX:',F6.3)
 2010   FORMAT
     &   (/1X,I3,'   rms: ',E10.4,'   max: ',E10.4,3X,A1,' at ',I4,I3)
 2020   FORMAT
     &   ( 1X,3X,'   a =', F7.3,'      CL =',F8.4  /
     &      1X,3X,'  Cm =', F8.4, '      CD =',F9.5,
     &            '   =>   CDf =',F9.5,'    CDp =',F9.5)
      END subroutine ! VIX
```

```
!**************************************************************************
! This subroutine was taken from XFOIL code                                *

      SUBROUTINE CDCALC
      INCLUDE 'XFOIL.INC'
C
      SA = SIN(ALFA)
      CA = COS(ALFA)
C
      IF(LVISC .AND. LBLINI) THEN
C
C----- set variables at the end of the wake
       THWAKE = THET(NBL(2),2)
       URAT   = UEDG(NBL(2),2)/QINF
       UEWAKE = UEDG(NBL(2),2) * (1.0-TKLAM) / (1.0 - TKLAM*URAT**2)
       SHWAKE = DSTR(NBL(2),2)/THET(NBL(2),2)
C
C----- extrapolate wake to downstream infinity using Squire-Young relation
C      (reduces errors of the wake not being long enough)
       CD = 2.0*THWAKE * (UEWAKE/QINF)**(0.5*(5.0+SHWAKE))
C
      ELSE
C
       CD = 0.0
C
      ENDIF
C
C---- calculate friction drag coefficient
      CDF = 0.0
      DO 20 IS=1, 2
        DO 205 IBL=3, IBLTE(IS)
          I  = IPAN(IBL  ,IS)
          IM = IPAN(IBL-1,IS)
          DX = (X(I) - X(IM))*CA + (Y(I) - Y(IM))*SA
          CDF = CDF + 0.5*(TAU(IBL,IS)+TAU(IBL-1,IS))*DX * 2.0/QINF**2
 205    CONTINUE
 20   CONTINUE
C
      RETURN
      END ! CDCALC
!***************************************************************************
! This subroutine include on CL the three-dimensional effects               *

      SUBROUTINE CLCALC2(N,Nref,sec,gam_a,ALFA,MINF,QINF,pcor,
     &                  XREF,YREF,
     &                  CL,CM,CDP, CL_ALF,CL_MSQ)
     include 'section.inc'
C-----------------------------------------------------------
C     Integrates surface pressures to get CL and CM.
C     Integrates skin friction to get CDF.
C     Calculates dCL/dAlpha for prescribed-CL routines.
C      Modified by Augusto Veiga
C-----------------------------------------------------------
     real,dimension(Nref,2):: pcor
      type(section):: sec
      REAL:: MINF,v
     real:: dui(N),gam_a(N),x(N),y(N)
C
C---- moment-reference coordinates
ccc      XREF = 0.25
ccc      YREF = 0.
C
C     transforming Vpot into pressure coefficient
      do i = 1,N
       sum = 0
       x(i) = sec.p(1,i)
       y(i) = sec.p(2,i)
       do k = 1,3
         v = sec.vpot(k,i)
         sum = sum + v**2
       enddo
```

```
        sec.cp(i) = 1.0-sum
      enddo
!transforming pcor into dui
       j = sec.istag
      do i = 1,sec.iblte(1)
        dui(j) = pcor(i,1)
        j = j-1
      enddo
      j = sec.istag
      do i = 2,sec.iblte(2)
        j = j+1
        dui(j) = pcor(i,2)
      enddo


       SA = SIN(ALFA)
       CA = COS(ALFA)
C
       BETA = SQRT(1.0 - MINF**2)
       BETA_MSQ = -0.5/BETA
C
       BFAC     = 0.5*MINF**2 / (1.0 + BETA)
       BFAC_MSQ = 0.5         / (1.0 + BETA)
      &         - BFAC        / (1.0 + BETA) * BETA_MSQ
C
       CL = 0.0
       CM = 0.0

       CDP = 0.0
C
       CL_ALF = 0.
       CL_MSQ = 0.
C
       I = 1
       CGINC = sec.cp(i) + dui(i)**2
       CPG1     = CGINC/(BETA + BFAC*CGINC)
       CPG1_MSQ = -CPG1/(BETA + BFAC*CGINC)*(BETA_MSQ + BFAC_MSQ*CGINC)
C
       CPI_GAM = -2.0*cginc
       CPC_CPI = (1.0 - BFAC*CPG1)/ (BETA + BFAC*CGINC)
       CPG1_ALF = CPC_CPI*CPI_GAM*GAM_A(I)
C
       DO 10 I=1, N
         IP = I+1
         IF(I.EQ.N) IP = 1
C
         CGINC = sec.cp(i) + dui(i)**2
         CPG2     = CGINC/(BETA + BFAC*CGINC)
         CPG2_MSQ = -CPG2/(BETA + BFAC*CGINC)*(BETA_MSQ + BFAC_MSQ*CGINC)
C
         CPI_GAM = -2.0*cginc
         CPC_CPI = (1.0 - BFAC*CPG2)/ (BETA + BFAC*CGINC)
         CPG2_ALF = CPC_CPI*CPI_GAM*GAM_A(IP)
C
         DX = (X(IP) - X(I))*CA + (Y(IP) - Y(I))*SA
         DY = (Y(IP) - Y(I))*CA - (X(IP) - X(I))*SA
         DG = CPG2 - CPG1
C
         AX = (0.5*(X(IP)+X(I))-XREF)*CA + (0.5*(Y(IP)+Y(I))-YREF)*SA
         AY = (0.5*(Y(IP)+Y(I))-YREF)*CA - (0.5*(X(IP)+X(I))-XREF)*SA
         AG = 0.5*(CPG2 + CPG1)
C
         DX_ALF = -(X(IP) - X(I))*SA + (Y(IP) - Y(I))*CA
         AG_ALF = 0.5*(CPG2_ALF + CPG1_ALF)
         AG_MSQ = 0.5*(CPG2_MSQ + CPG1_MSQ)
C
         CL     = CL     + DX* AG
         CDP    = CDP    - DY* AG
         CM     = CM     - DX*(AG*AX + DG*DX/12.0)
      &                  - DY*(AG*AY + DG*DY/12.0)
C
         CL_ALF = CL_ALF + DX*AG_ALF + AG*DX_ALF
```

```
        CL_MSQ = CL_MSQ + DX*AG_MSQ
C
        CPG1 = CPG2
        CPG1_ALF = CPG2_ALF
        CPG1_MSQ = CPG2_MSQ
   10 CONTINUE
C
      RETURN
      END ! CLCALC2
```

VIX Surface Interpolation

```
      subroutine surface(pan,Npan,sec,Nsec,Nmax,Nte,fl)
     include 'section.inc'
     include 'panel.inc'
!     This subroutine calculates collocation points for each panel
!     calculates mean edges
!     calculates surface coordinate for each collocation point on s and t
!     gets points and interpolate cp using a spline distribution
!***********************************************************************
!*     (C) Augusto Veiga, University of Southampton 2003              *
!***********************************************************************
     integer Npan,Nsec,Nv,Nh,Nb,Nmax
     type(panel),dimension(Npan):: pan,panb
     type(section),dimension(Nsec):: sec
     real:: S(3),Dmax
      logical:: fl
     !Calculating collocation points
     Do i=1,Npan
       S=0
       do j=1,4
         S(1)=S(1)+pan(i).nd(j).x
         S(2)=S(2)+pan(i).nd(j).y
         S(3)=S(3)+pan(i).nd(j).z
       enddo
       pan(i).co.x=S(1)/4.
       pan(i).co.y=S(2)/4.
       pan(i).co.z=S(3)/4.
     enddo
      !call smooth_cp(pan,Npan)
     !Calculating surface coordinates s and t
     call org_pan(pan,panb,Npan,Nb,Nv,Nh,Nte)
     call surf_coord(panb,Nb,Nv,Nh)
     call surf_spl(panb,Nb,Nv,Nh,fl)   !Makes a spline surface
     do i = 1,3
       call press_int(panb,Nb,Nv,Nh,i,.false.)!Makes a V(x,y,z) surface
       ! i ->  1 = x
!             2 = y
!             3 = z
     enddo
     !dividing surface into sections with 100 points equaly
     ! spaced each and following a plane which normal is the
     ! slope at yz plane
     do i=1,Nsec
        sec(i).Nnd=Nmax
     enddo
     call interpol_surf(panb,Nb,Nv,Nsec,sec,Nmax)
     call sec_normal(sec,Nsec,Nmax) !calculate normals on each point of section
     !call calc_Up(sec,Nsec,Nmax) !calculate modular potential velocity
     call vsec_plot(Sec,Nsec,Nmax) !plots spanwise velocity

     return
     end subroutine

      subroutine interpol_surf(pan,Npan,Nv,Nsec,sec,Nmax)
     include 'section.inc'
     include 'panel.inc'

     integer :: Npan,Nv,Nsec,Nmax,iv,ih
     type(panel),dimension(Npan) :: pan
     type (section),dimension(Nsec):: sec
```

```
real:: tc,sc,t

! divide body onto spanwise sections with tc spacement
tc= (pan(Nv).nd(4).t-pan(1).nd(1).t)/(Nsec-1)
t=pan(1).nd(1).t
k=1
 id=0
ih=1
iv=1
i=1
do while (i<=Nsec)
  !This loop makes geometry interpolation just
   !finding s last points
    sc=1.0
    sec(i).id=id
    id=id+1
    sec(i).s(1)=0
    sec(i).s(Nmax)=sc
    sec(i).t(1)=t
    sc=(sec(i).s(Nmax)-sec(i).s(1))/(Nmax-1)
    if ( t<=pan(iv).nd(4).t) then
       ih=iv
       call interpol_sec(sec(i),1,pan,Npan,ih,iv,Nv,1)  !x
       call interpol_sec(sec(i),1,pan,Npan,ih,iv,Nv,2)  !y
       call interpol_sec(sec(i),1,pan,Npan,ih,iv,Nv,3)  !z
       !call interpol_Vs(sec(i),1,pan,Npan,ih,iv,Nv)  !Vm
       do j=2,Nmax
         if (j==Nmax) then
            sec(i).s(j)=pan(Npan-(Nv-iv)).nd(2).s
            sec(i).t(j)=t
         else
            sec(i).s(j)=sc+sec(i).s(j-1)
            sec(i).t(j)=t
         endif
         call interpol_sec(sec(i),j,pan,Npan,ih,iv,Nv,1)  !x
         call interpol_sec(sec(i),j,pan,Npan,ih,iv,Nv,2)  !y
         call interpol_sec(sec(i),j,pan,Npan,ih,iv,Nv,3)  !z
         !call interpol_Vs(sec(i),j,pan,Npan,ih,iv,Nv)  !Vm
      enddo
      t=t+tc
      i=i+1
    else if (i==Nsec) then
       ih=Nv          !Last section
       call interpol_sec(sec(i),1,pan,Npan,ih,iv,Nv,1)  !x
       call interpol_sec(sec(i),1,pan,Npan,ih,iv,Nv,2)  !y
       call interpol_sec(sec(i),1,pan,Npan,ih,iv,Nv,3)  !z
       !call interpol_Vs(sec(i),1,pan,Npan,ih,iv,Nv)  !Vm
       do j=2,Nmax
         if (j==Nmax) then
            sec(i).s(j)=pan(Npan).nd(2).s
            sec(i).t(j)=pan(Npan).nd(3).t
         else
            sec(i).s(j)=sc+sec(i).s(j-1)
            sec(i).t(j)=t
         endif
         call interpol_sec(sec(i),j,pan,Npan,ih,iv,Nv,1)  !x
         call interpol_sec(sec(i),j,pan,Npan,ih,iv,Nv,2)  !y
         call interpol_sec(sec(i),j,pan,Npan,ih,iv,Nv,3)  !z
         !call interpol_Vs(sec(i),j,pan,Npan,ih,iv,Nv)  !Vm
       enddo
       i=i+1
    else if (iv<Nv) then
       iv=iv+1
    endif
enddo
!Now we calculate the cossine of sectional segments
do isec = 1,Nsec
  do j = 1,Nmax
     if (j==Nmax) then
        do k = 1,3
          sec(isec).vcos(k,j) = sec(isec).vcos(k,j-1)
```

```
           enddo
         else
           !Calculate length
           soma = 0
           do k = 1,3
             soma = soma+ (sec(isec).p(k,j+1)-sec(isec).p(k,j))**2
           enddo
           !calculate cossine
           do k = 1,3
             sec(isec).vcos(k,j)=(sec(isec).p(k,j+1)-sec(isec).p(k,j))
     &         /sqrt(soma)
           enddo
        endif
      enddo
    enddo
    !Now we interpolate sectional velocity (Vs)
    !...and calculate the tangential velocity q
    iv = 1
    ih = 1
    i =1
    do while(i<=Nsec)
        sc=1.0
         t = sec(i).t(1)
        if ( t<=pan(iv).nd(4).t) then
           ih=iv
           call interpol_Vs(sec(i),1,pan,Npan,ih,iv,Nv)  !Vm
           do j=2,Nmax
             call interpol_Vs(sec(i),j,pan,Npan,ih,iv,Nv)  !Vm
          enddo
          i = i+1
        else if (i==Nsec) then
           ih=Nv          !Last section
           call interpol_Vs(sec(i),1,pan,Npan,ih,iv,Nv)  !Vm
           do j=2,Nmax
             call interpol_Vs(sec(i),j,pan,Npan,ih,iv,Nv)  !Vm
           enddo
           i = i+1
        else if (iv<Nv) then
           iv=iv+1
        endif
    enddo

     ! Writing a scratch file of potential tangential velocity
    open(1,file='potential_scratch.txt')
      write(1,100)
      do isec = 1,Nsec
        do j = 1,Nmax
          write(1,200) j,sec(isec).Up(j),sec(isec).vcos(1,j),
     &                 sec(isec).vcos(2,j),sec(isec).vcos(3,j)
        enddo
        write(1,*)
      enddo
    close(1)

    return
100  format('node , velocity , cos x , cos y , cos z')
200  format(i4,1x,f8.3,1x,f8.3,1x,f8.3,1x,f8.3)
    end subroutine

     subroutine interpol_sec(sec,j,pan,Npan,ih,iv,Nv,Nflag)
    include 'section.inc'
    include 'panel.inc'
!    This subroutine gets the current section with s,t variables and,
!    marching along chordwise panels, interpolates x,y,z using Coons
!    bicubic spline surface
!    As the solution marches, ih is changed  to the next panel on
!    chordwise direction.
!    Nflag chooses what is going to be interpolated
!     1  ->  x coordinate
!     2  ->  y coordinate
!     3  ->  z coordinate
!     4  ->  cp
```

```
!
!      Bd1, Bd2 -> first derivative coefficients
!*****************************************************************
!*       (C) Augusto Veiga, University of Southampton, 2003       *
!*****************************************************************
      integer j,ih,iv,Nv,Nflag
      type(section):: sec
     type(panel),dimension(Npan)::pan
     real::s,w,T,U,A(4),B1(4),B2(4),soma
     real:: delta1,delta2,Vn(4),C(4),D(4),dxs2(4),dxt2(4),x(4)
     real:: tiny

     tiny=1.0E-10
     soma=0;A=0;B1=0;B2=0;Bd1=0;Bd2=0
     s=sec.s(j)
     w=sec.t(j)
     i=ih
     lp1: do
      select case (Nflag)
        case(1) !options to interpolate variables
          do k=1,4
             dxs2(k)=pan(i).nd(k).dxs2
             dxt2(k)=pan(i).nd(k).dxt2
             x(k)=pan(i).nd(k).x
          enddo
        case(2)
          do k=1,4
             dxs2(k)=pan(i).nd(k).dys2
             dxt2(k)=pan(i).nd(k).dyt2
             x(k)=pan(i).nd(k).y
          enddo
        case(3)
          do k=1,4
             dxs2(k)=pan(i).nd(k).dzs2
             dxt2(k)=pan(i).nd(k).dzt2
             x(k)=pan(i).nd(k).z
          enddo
        case(4)
          do k=1,4
             dxs2(k)=0
             dxt2(k)=0
             x(k)=pan(i).nd(k).up
          enddo
      end select
        if (s<pan(i).nd(2).s) then
           !calculate normal coefficients T and U
             T=(s-pan(i).nd(1).s)/(pan(i).nd(2).s-pan(i).nd(1).s)
             U=(w-pan(i).nd(1).t)/(pan(i).nd(4).t-pan(i).nd(1).t)
             !calculate coefficients A
             A(1)=(1-T)*(1-U)
             A(2)=T*(1-U)
             A(3)=T*U
             A(4)=(1-T)*U
             !calculate coefficients B
          delta1=(pan(i).nd(2).s-pan(i).nd(1).s)
          delta2=(pan(i).nd(4).t-pan(i).nd(1).t)
          do k=1,4
                B1(k)=(A(k)**3-A(k))/6.*delta1
            B2(k)=(A(k)**3-A(k))/6.*delta2
          enddo
          !calculating normal vectors of 2nd derivatives
          C(1)=B1(2)*dxs2(2)-B1(1)*dxs2(1)
          C(2)=B1(3)*dxs2(3)-B1(2)*dxs2(2)
         C(3)=B1(4)*dxs2(4)-B1(3)*dxs2(3)
         C(4)=B1(4)*dxs2(4)-B1(1)*dxs2(1)
          D(1)=B2(2)*dxt2(2)-B2(1)*dxt2(1)
          D(2)=B2(3)*dxt2(3)-B2(2)*dxt2(2)
         D(3)=B2(4)*dxt2(4)-B2(3)*dxt2(3)
         D(4)=B2(4)*dxt2(4)-B2(1)*dxt2(1)
         Vn(1)=c(1)*d(4)-d(1)*c(4)
         Vn(2)=c(1)*d(2)-d(1)*c(2)
         Vn(3)=c(3)*d(2)-d(3)*c(2)
```

```
    Vn(4)=c(3)*d(4)-d(3)*c(4)

    !calculate final value for interpolation
    soma=0
    do k=1,4
       soma=soma+A(k)*x(k)+Vn(k)
    enddo
    if (abs(soma)<tiny) then
       soma=0
    endif
    exit lp1
  else if (s==pan(Npan-(Nv-iv)).nd(2).s) then
      !calculate normal coefficients T and U
        T=(s-pan(i).nd(1).s)/(pan(i).nd(2).s-pan(i).nd(1).s)
        U=(w-pan(i).nd(1).t)/(pan(i).nd(4).t-pan(i).nd(1).t)
        !calculate coefficients A
        A(1)=(1-T)*(1-U)
        A(2)=T*(1-U)
        A(3)=T*U
        A(4)=(1-T)*U
        !calculate coefficients B
     delta1=(pan(i).nd(2).s-pan(i).nd(1).s)
     delta2=(pan(i).nd(4).t-pan(i).nd(1).t)
     do k=1,4
           B1(k)=(A(k)**3-A(k))*delta1
       B2(k)=(A(k)**3-A(k))*delta2
     enddo
     !calculating normal vectors of 2nd derivatives
     C(1)=B1(2)*dxs2(2)-B1(1)*dxs2(1)
     C(2)=B1(3)*dxs2(3)-B1(2)*dxs2(2)
    C(3)=B1(4)*dxs2(4)-B1(3)*dxs2(3)
    C(4)=B1(4)*dxs2(4)-B1(1)*dxs2(1)
     D(1)=B2(2)*dxt2(2)-B2(1)*dxt2(1)
     D(2)=B2(3)*dxt2(3)-B2(2)*dxt2(2)
    D(3)=B2(4)*dxt2(4)-B2(3)*dxt2(3)
    D(4)=B2(4)*dxt2(4)-B2(1)*dxt2(1)
    Vn(1)=c(1)*d(4)-d(1)*c(4)
    Vn(2)=c(1)*d(2)-d(1)*c(2)
    Vn(3)=c(3)*d(2)-d(3)*c(2)
    Vn(4)=c(3)*d(4)-d(3)*c(4)
    !calculate final value for interpolation
    soma=0
    do k=1,4
       soma=soma+A(k)*x(k)+Vn(k)
    enddo
    if (abs(soma)<tiny) then
       soma=0
    endif
    exit lp1
  else if(ih>Npan-(Nv-iv)) then
    exit lp1
  else if (ih<=Npan-(Nv-iv)) then
    ih=ih+Nv
    i=ih
  endif
enddo lp1

select case (Nflag)
   case(1) !options to interpolate variables
     sec.p(1,j)=soma
   case(2)
      sec.p(2,j)=soma
   case(3)
      sec.p(3,j)=soma
!   case(4)
!        sec.cp(j)=soma
end select


return
end subroutine
```

```
!___________________________________________________________________
       subroutine interpol_Vs(sec,j,pan,Npan,ih,iv,Nv)
      include 'section.inc'
      include 'panel.inc'
!      This subroutine gets the current section with s,t variables and,
!      marching along chordwise panels, interpolates x,y,z using Coons
!      bicubic spline surface
!      As the solution marches, ih is changed  to the next panel on
!      chordwise direction.
!      Nflag chooses what is going to be interpolated
!       1  ->  x coordinate
!       2  ->  y coordinate
!       3  ->  z coordinate
!       4  ->  cp
!*****************************************************************
!*       (C) Augusto Veiga, University of Southampton, 2003         *
!*****************************************************************
       integer j,ih,iv,Nv,Nflag
       type(section):: sec
      type(panel),dimension(Npan)::pan
      real::s,w,T,U,A(4),B1(4),B2(4),soma,V(3)
      real:: delta1,delta2,Vn(4),C(4),D(4),dxs2(4),dxt2(4),x(4)
      real:: tiny

      tiny=1.0E-10
      soma=0;A=0;B1=0;B2=0
      s=sec.s(j)
      w=sec.t(j)
      i=ih
      do ind =1,3
      lp1: do
         do k=1,4
            dxs2(k)=0
            dxt2(k)=0
            x(k)=pan(i).nd(k).vm(ind)
         enddo
         if (s<pan(i).nd(2).s) then
            !calculate normal coefficients T and U
               T=(s-pan(i).nd(1).s)/(pan(i).nd(2).s-pan(i).nd(1).s)
               U=(w-pan(i).nd(1).t)/(pan(i).nd(4).t-pan(i).nd(1).t)
               !calculate coefficients A
               A(1)=(1-T)*(1-U)
               A(2)=T*(1-U)
               A(3)=T*U
               A(4)=(1-T)*U
               !calculate coefficients B
           !calculating normal vectors of 2nd derivatives
          !calculate final value for interpolation
          soma=0
          do k=1,4
             soma=soma+A(k)*x(k)
          enddo
          if (abs(soma)<tiny) then
             soma=0
          endif
          exit lp1
        else if (s==pan(Npan-(Nv-iv)).nd(2).s) then
            !calculate normal coefficients T and U
               T=(s-pan(i).nd(1).s)/(pan(i).nd(2).s-pan(i).nd(1).s)
               U=(w-pan(i).nd(1).t)/(pan(i).nd(4).t-pan(i).nd(1).t)
               !calculate coefficients A
               A(1)=(1-T)*(1-U)
               A(2)=T*(1-U)
               A(3)=T*U
               A(4)=(1-T)*U
               !calculate coefficients B
           !calculating normal vectors of 2nd derivatives
          !calculate final value for interpolation
          soma=0
          do k=1,4
             soma=soma+A(k)*x(k)
```

```
          enddo
          if (abs(soma)<tiny) then
             soma=0
          endif
          exit lp1
        else if(ih>Npan-(Nv-iv)) then
          exit lp1
        else if (ih<=Npan-(Nv-iv)) then
          ih=ih+Nv
          i=ih
        endif
      enddo lp1
       V(ind) = soma
      enddo !indexes

       soma = 0
       do k = 1,3
        sec.vpot(k,j) = v(k)                  !potential 3D velocity
        soma = soma+ (sec.vcos(k,j)*V(k))**2  !velocity on section
        !projects velocity on each segment of section
      enddo

      sec.Up(j) = sqrt(soma) !tangential velocity on section pt

      return
      end subroutine
```

XFOIL Routines for Initial Boundary Layer Solution that Were Added to VIX

```
      SUBROUTINE SETBL
C-------------------------------------------------
C     Sets up the BL Newton system coefficients
C     for the current BL variables and the edge
C     velocities received from SETUP. The local
C     BL system coefficients are then
C     incorporated into the global Newton system.
C-------------------------------------------------
      INCLUDE 'XFOIL.INC'
      INCLUDE 'XBL.INC'
      REAL USAV(IVX,2)
      REAL U1_M(2*IVX), U2_M(2*IVX)
      REAL D1_M(2*IVX), D2_M(2*IVX)
      REAL ULE1_M(2*IVX), ULE2_M(2*IVX)
      REAL UTE1_M(2*IVX), UTE2_M(2*IVX)
      REAL MA_CLMR, MSQ_CLMR, MDI
C
C---- set the CL used to define Mach, Reynolds numbers
      IF(LALFA) THEN
       CLMR = CL
      ELSE
       CLMR = CLSPEC
      ENDIF
C
C---- set current MINF(CL)
      !CALL MRCL(CLMR,MA_CLMR,RE_CLMR)
     MINF = 0
     RE_CLMR = 0
      MA_CLMR = 0
     CLMR = 0.000001
      MSQ_CLMR = 2.0*MINF*MA_CLMR
C
C---- set compressibility parameter TKLAM and derivative TK_MSQ
      !CALL COMSET
C
C---- set gas constant (= Cp/Cv)
      GAMBL = GAMMA
      GM1BL = GAMM1
C
C---- set parameters for compressibility correction
      QINFBL = QINF
      TKBL    = TKLAM
      TKBL_MS = TKL_MSQ
```

```
C
C---- stagnation density and 1/enthalpy
      RSTBL    = (1.0 + 0.5*GM1BL*MINF**2) ** (1.0/GM1BL)
      RSTBL_MS = 0.5*RSTBL/(1.0 + 0.5*GM1BL*MINF**2)
C
      HSTINV    = GM1BL*(MINF/QINFBL)**2 / (1.0 + 0.5*GM1BL*MINF**2)
      HSTINV_MS = GM1BL*( 1.0/QINFBL)**2 / (1.0 + 0.5*GM1BL*MINF**2)
     &                - 0.5*GM1BL*HSTINV / (1.0 + 0.5*GM1BL*MINF**2)
C
C---- Sutherland's const./To   (assumes stagnation conditions are at STP)
      HVRAT = 0.35
C
C---- set Reynolds number based on freestream density, velocity, viscosity
      HERAT    = 1.0 - 0.5*QINFBL**2*HSTINV
      HERAT_MS =     - 0.5*QINFBL**2*HSTINV_MS
C
      REYBL    = REINF * SQRT(HERAT**3) * (1.0+HVRAT)/(HERAT+HVRAT)
      REYBL_RE =         SQRT(HERAT**3) * (1.0+HVRAT)/(HERAT+HVRAT)
      REYBL_MS = REYBL * (1.5/HERAT - 1.0/(HERAT+HVRAT))*HERAT_MS
C
      AMCRIT = ACRIT
C
C---- save TE thickness
      DWTE = WGAP(1)
C
      IF(.NOT.LBLINI) THEN
C----- initialize BL by marching with Ue (fudge at separation)
       WRITE(*,*)
       WRITE(*,*) 'Initializing BL ...'
       CALL MRCHUE
       LBLINI = .TRUE.
      ENDIF
C
      WRITE(*,*)
C
C---- march BL with current Ue and Ds to establish transition
      CALL MRCHDU
C
      DO 5 IS=1, 2
        DO 6 IBL=2, NBL(IS)
          USAV(IBL,IS) = UEDG(IBL,IS)
    6   CONTINUE
    5 CONTINUE
C
      CALL UESET
C
      DO 7 IS=1, 2
        DO 8 IBL=2, NBL(IS)
          TEMP = USAV(IBL,IS)
          USAV(IBL,IS) = UEDG(IBL,IS)
          UEDG(IBL,IS) = TEMP
    8   CONTINUE
    7 CONTINUE
C
      ILE1 = IPAN(2,1)
      ILE2 = IPAN(2,2)
      ITE1 = IPAN(IBLTE(1),1)
      ITE2 = IPAN(IBLTE(2),2)
C
      JVTE1 = ISYS(IBLTE(1),1)
      JVTE2 = ISYS(IBLTE(2),2)
C
      DULE1 = UEDG(2,1) - USAV(2,1)
      DULE2 = UEDG(2,2) - USAV(2,2)
C
C---- set LE and TE Ue sensitivities wrt all m values
      DO 10 JS=1, 2
        DO 110 JBL=2, NBL(JS)
          J  = IPAN(JBL,JS)
          JV = ISYS(JBL,JS)
          ULE1_M(JV) = -VTI(       2,1)*VTI(JBL,JS)*DIJ(ILE1,J)
          ULE2_M(JV) = -VTI(       2,2)*VTI(JBL,JS)*DIJ(ILE2,J)
```

```
          UTE1_M(JV) = -VTI(IBLTE(1),1)*VTI(JBL,JS)*DIJ(ITE1,J)
          UTE2_M(JV) = -VTI(IBLTE(2),2)*VTI(JBL,JS)*DIJ(ITE2,J)
  110   CONTINUE
   10 CONTINUE
C
      ULE1_A = UINV_A(2,1)
      ULE2_A = UINV_A(2,2)
C
C**** Go over each boundary layer/wake
      DO 2000 IS=1, 2
C
C---- there is no station "1" at similarity, so zero everything out
      DO 20 JS=1, 2
        DO 210 JBL=2, NBL(JS)
          JV = ISYS(JBL,JS)
          U1_M(JV) = 0.
          D1_M(JV) = 0.
  210   CONTINUE
   20 CONTINUE
      U1_A = 0.
      D1_A = 0.
C
      DUE1 = 0.
      DDS1 = 0.
C
C---- similarity station pressure gradient parameter  x/u du/dx
      IBL = 2
      BULE = 1.0
C
C---- set forced transition arc length position
      CALL XIFSET(IS)
C
      TRAN = .FALSE.
      TURB = .FALSE.
C
C**** Sweep downstream setting up BL equation linearizations
      DO 1000 IBL=2, NBL(IS)
C
      IV  = ISYS(IBL,IS)
C
      SIMI = IBL.EQ.2
      WAKE = IBL.GT.IBLTE(IS)
      TRAN = IBL.EQ.ITRAN(IS)
      TURB = IBL.GT.ITRAN(IS)
C
      I = IPAN(IBL,IS)
C
C---- set primary variables for current station
      XSI = XSSI(IBL,IS)
      IF(IBL.LT.ITRAN(IS)) AMI = CTAU(IBL,IS)
      IF(IBL.GE.ITRAN(IS)) CTI = CTAU(IBL,IS)
      UEI = UEDG(IBL,IS)
      THI = THET(IBL,IS)
      MDI = MASS(IBL,IS)
C
      DSI = MDI/UEI
C
      IF(WAKE) THEN
       IW = IBL - IBLTE(IS)
       DSWAKI = WGAP(IW)
      ELSE
       DSWAKI = 0.
      ENDIF
C
C---- set derivatives of DSI (= D2)
      D2_M2 =  1.0/UEI
      D2_U2 = -DSI/UEI
C
      DO 30 JS=1, 2
        DO 310 JBL=2, NBL(JS)
          J  = IPAN(JBL,JS)
          JV = ISYS(JBL,JS)
```

```
          U2_M(JV) = -VTI(IBL,IS)*VTI(JBL,JS)*DIJ(I,J)
          D2_M(JV) = D2_U2*U2_M(JV)
  310   CONTINUE
   30 CONTINUE
      D2_M(IV) = D2_M(IV) + D2_M2
C
      U2_A = UINV_A(IBL,IS)
      D2_A = D2_U2*U2_A
C
C---- "forced" changes due to mismatch between UEDG and USAV=UINV+dij*MASS
      DUE2 = UEDG(IBL,IS) - USAV(IBL,IS)
      DDS2 = D2_U2*DUE2
C
      CALL BLPRV(XSI,AMI,CTI,THI,DSI,DSWAKI,UEI)
      CALL BLKIN
C
C---- check for transition and set TRAN, XT, etc. if found
      IF(TRAN) THEN
        CALL TRCHEK
        AMI = AMPL2
      ENDIF
      IF(IBL.EQ.ITRAN(IS) .AND. .NOT.TRAN) THEN
       WRITE(*,*) 'SETBL: Xtr???  n1 n2: ', AMPL1, AMPL2
      ENDIF
C
C---- assemble 10x4 linearized system for dCtau, dTh, dDs, dUe, dXi
C     at the previous "1" station and the current "2" station
C
      IF(IBL.EQ.IBLTE(IS)+1) THEN
C
C----- define quantities at start of wake, adding TE base thickness to Dstar
       TTE = THET(IBLTE(1),1) + THET(IBLTE(2),2)
       DTE = DSTR(IBLTE(1),1) + DSTR(IBLTE(2),2) + ANTE
       CTE = ( CTAU(IBLTE(1),1)*THET(IBLTE(1),1)
     &       + CTAU(IBLTE(2),2)*THET(IBLTE(2),2) ) / TTE
       CALL TESYS(CTE,TTE,DTE)
C
       TTE_TTE1 = 1.0
       TTE_TTE2 = 1.0
       DTE_MTE1 =               1.0 / UEDG(IBLTE(1),1)
       DTE_UTE1 = -DSTR(IBLTE(1),1) / UEDG(IBLTE(1),1)
       DTE_MTE2 =               1.0 / UEDG(IBLTE(2),2)
       DTE_UTE2 = -DSTR(IBLTE(2),2) / UEDG(IBLTE(2),2)
       CTE_CTE1 = THET(IBLTE(1),1)/TTE
       CTE_CTE2 = THET(IBLTE(2),2)/TTE
       CTE_TTE1 = (CTAU(IBLTE(1),1) - CTE)/TTE
       CTE_TTE2 = (CTAU(IBLTE(2),2) - CTE)/TTE
C
C----- re-define D1 sensitivities wrt m since D1 depends on both TE Ds values
       DO 35 JS=1, 2
         DO 350 JBL=2, NBL(JS)
           J  = IPAN(JBL,JS)
           JV = ISYS(JBL,JS)
           D1_M(JV) = DTE_UTE1*UTE1_M(JV) + DTE_UTE2*UTE2_M(JV)
  350    CONTINUE
   35  CONTINUE
       D1_M(JVTE1) = D1_M(JVTE1) + DTE_MTE1
       D1_M(JVTE2) = D1_M(JVTE2) + DTE_MTE2
C
C----- "forced" changes from  UEDG --- USAV=UINV+dij*MASS  mismatch
       DUE1 = 0.
       DDS1 = DTE_UTE1*(UEDG(IBLTE(1),1) - USAV(IBLTE(1),1))
     &      + DTE_UTE2*(UEDG(IBLTE(2),2) - USAV(IBLTE(2),2))
C
      ELSE
C
       CALL BLSYS
C
      ENDIF
C
C
C---- Save wall shear and equil. max shear coefficient for plotting output
```

```
      TAU(IBL,IS) = 0.5*R2*U2*U2*CF2
      DIS(IBL,IS) =     R2*U2*U2*U2*DI2*HS2*0.5
      CTQ(IBL,IS) = CQ2
      DELT(IBL,IS) = DE2
      USLP(IBL,IS) = 1.60/(1.0+US2)
C
C---- set XI sensitivities wrt LE Ue changes
      IF(IS.EQ.1) THEN
       XI_ULE1 =  SST_GO
       XI_ULE2 = -SST_GP
      ELSE
       XI_ULE1 = -SST_GO
       XI_ULE2 =  SST_GP
      ENDIF
C
C---- stuff BL system coefficients into main Jacobian matrix
C
      DO 40 JV=1, NSYS
        VM(1,JV,IV) = VS1(1,3)*D1_M(JV) + VS1(1,4)*U1_M(JV)
     &              + VS2(1,3)*D2_M(JV) + VS2(1,4)*U2_M(JV)
     &              + (VS1(1,5) + VS2(1,5) + VSX(1))
     &               *(XI_ULE1*ULE1_M(JV) + XI_ULE2*ULE2_M(JV))
   40 CONTINUE
C
      VB(1,1,IV) = VS1(1,1)
      VB(1,2,IV) = VS1(1,2)
C
      VA(1,1,IV) = VS2(1,1)
      VA(1,2,IV) = VS2(1,2)
C
      IF(LALFA) THEN
       VDEL(1,2,IV) = VSR(1)*RE_CLMR + VSM(1)*MSQ_CLMR
      ELSE
       VDEL(1,2,IV) =
     &       (VS1(1,4)*U1_A + VS1(1,3)*D1_A)
     &     + (VS2(1,4)*U2_A + VS2(1,3)*D2_A)
     &     + (VS1(1,5) + VS2(1,5) + VSX(1))
     &      *(XI_ULE1*ULE1_A + XI_ULE2*ULE2_A)
      ENDIF
C
      VDEL(1,1,IV) = VSREZ(1)
     &   + (VS1(1,4)*DUE1 + VS1(1,3)*DDS1)
     &   + (VS2(1,4)*DUE2 + VS2(1,3)*DDS2)
     &   + (VS1(1,5) + VS2(1,5) + VSX(1))
     &    *(XI_ULE1*DULE1 + XI_ULE2*DULE2)
C
C
      DO 50 JV=1, NSYS
        VM(2,JV,IV) = VS1(2,3)*D1_M(JV) + VS1(2,4)*U1_M(JV)
     &              + VS2(2,3)*D2_M(JV) + VS2(2,4)*U2_M(JV)
     &              + (VS1(2,5) + VS2(2,5) + VSX(2))
     &               *(XI_ULE1*ULE1_M(JV) + XI_ULE2*ULE2_M(JV))
   50 CONTINUE
C
      VB(2,1,IV)  = VS1(2,1)
      VB(2,2,IV)  = VS1(2,2)
C
      VA(2,1,IV) = VS2(2,1)
      VA(2,2,IV) = VS2(2,2)
C
      IF(LALFA) THEN
       VDEL(2,2,IV) = VSR(2)*RE_CLMR + VSM(2)*MSQ_CLMR
      ELSE
       VDEL(2,2,IV) =
     &       (VS1(2,4)*U1_A + VS1(2,3)*D1_A)
     &     + (VS2(2,4)*U2_A + VS2(2,3)*D2_A)
     &     + (VS1(2,5) + VS2(2,5) + VSX(2))
     &      *(XI_ULE1*ULE1_A + XI_ULE2*ULE2_A)
      ENDIF
C
      VDEL(2,1,IV) = VSREZ(2)
     &   + (VS1(2,4)*DUE1 + VS1(2,3)*DDS1)
```

```
     &   + (VS2(2,4)*DUE2 + VS2(2,3)*DDS2)
     &   + (VS1(2,5) + VS2(2,5) + VSX(2))
     &    *(XI_ULE1*DULE1 + XI_ULE2*DULE2)
C
C
      DO 60 JV=1, NSYS
        VM(3,JV,IV) = VS1(3,3)*D1_M(JV) + VS1(3,4)*U1_M(JV)
     &              + VS2(3,3)*D2_M(JV) + VS2(3,4)*U2_M(JV)
     &              + (VS1(3,5) + VS2(3,5) + VSX(3))
     &               *(XI_ULE1*ULE1_M(JV) + XI_ULE2*ULE2_M(JV))
   60 CONTINUE
C
      VB(3,1,IV) = VS1(3,1)
      VB(3,2,IV) = VS1(3,2)
C
      VA(3,1,IV) = VS2(3,1)
      VA(3,2,IV) = VS2(3,2)
C
      IF(LALFA) THEN
       VDEL(3,2,IV) = VSR(3)*RE_CLMR + VSM(3)*MSQ_CLMR
      ELSE
       VDEL(3,2,IV) =
     &       (VS1(3,4)*U1_A + VS1(3,3)*D1_A)
     &     + (VS2(3,4)*U2_A + VS2(3,3)*D2_A)
     &     + (VS1(3,5) + VS2(3,5) + VSX(3))
     &      *(XI_ULE1*ULE1_A + XI_ULE2*ULE2_A)
      ENDIF
C
      VDEL(3,1,IV) = VSREZ(3)
     &   + (VS1(3,4)*DUE1 + VS1(3,3)*DDS1)
     &   + (VS2(3,4)*DUE2 + VS2(3,3)*DDS2)
     &   + (VS1(3,5) + VS2(3,5) + VSX(3))
     &    *(XI_ULE1*DULE1 + XI_ULE2*DULE2)
C
C
      IF(IBL.EQ.IBLTE(IS)+1) THEN
C
C----- redefine coefficients for TTE, DTE, etc
       VZ(1,1)    = VS1(1,1)*CTE_CTE1
       VZ(1,2)    = VS1(1,1)*CTE_TTE1 + VS1(1,2)*TTE_TTE1
       VB(1,1,IV) = VS1(1,1)*CTE_CTE2
       VB(1,2,IV) = VS1(1,1)*CTE_TTE2 + VS1(1,2)*TTE_TTE2
C
       VZ(2,1)    = VS1(2,1)*CTE_CTE1
       VZ(2,2)    = VS1(2,1)*CTE_TTE1 + VS1(2,2)*TTE_TTE1
       VB(2,1,IV) = VS1(2,1)*CTE_CTE2
       VB(2,2,IV) = VS1(2,1)*CTE_TTE2 + VS1(2,2)*TTE_TTE2
C
       VZ(3,1)    = VS1(3,1)*CTE_CTE1
       VZ(3,2)    = VS1(3,1)*CTE_TTE1 + VS1(3,2)*TTE_TTE1
       VB(3,1,IV) = VS1(3,1)*CTE_CTE2
       VB(3,2,IV) = VS1(3,1)*CTE_TTE2 + VS1(3,2)*TTE_TTE2
C
      ENDIF
C
C---- turbulent intervals will follow if currently at transition interval
      IF(TRAN) THEN
        TURB = .TRUE.
C
C------ save transition location
        ITRAN(IS) = IBL
        TFORCE(IS) = TRFORC
        XSSITR(IS) = XT
C
C------ interpolate airfoil geometry to find transition x/c
C-      (for user output)
        IF(IS.EQ.1) THEN
         STR = SST - XT
        ELSE
         STR = SST + XT
        ENDIF
        CHX = XTE - XLE
```

```
        CHY = YTE - YLE
        CHSQ = CHX**2 + CHY**2
        XTR = SEVAL(STR,X,XP,S,N)
        YTR = SEVAL(STR,Y,YP,S,N)
        XOCTR(IS) = ((XTR-XLE)*CHX + (YTR-YLE)*CHY)/CHSQ
        YOCTR(IS) = ((YTR-YLE)*CHX - (XTR-XLE)*CHY)/CHSQ
      ENDIF
C
      TRAN = .FALSE.
C
      IF(IBL.EQ.IBLTE(IS)) THEN
C----- set "2" variables at TE to wake correlations for next station
C
       TURB = .TRUE.
       WAKE = .TRUE.
       CALL BLVAR(3)
       CALL BLMID(3)
      ENDIF
C
      DO 80 JS=1, 2
        DO 810 JBL=2, NBL(JS)
          JV = ISYS(JBL,JS)
          U1_M(JV) = U2_M(JV)
          D1_M(JV) = D2_M(JV)
  810   CONTINUE
   80 CONTINUE
C
      U1_A = U2_A
      D1_A = D2_A
C
      DUE1 = DUE2
      DDS1 = DDS2
C
C---- set BL variables for next station
      DO 190 ICOM=1, NCOM
        COM1(ICOM) = COM2(ICOM)
  190 CONTINUE
C
C---- next streamwise station
 1000 CONTINUE
C
      IF(TFORCE(IS)) THEN
       WRITE(*,9100) IS,XOCTR(IS),ITRAN(IS)
 9100  FORMAT(1X,'Side',I2,' forced transition at x/c = ',F7.4,I5)
      ELSE
       WRITE(*,9200) IS,XOCTR(IS),ITRAN(IS)
 9200  FORMAT(1X,'Side',I2,'  free  transition at x/c = ',F7.4,I5)
      ENDIF
C
C---- next airfoil side
 2000 CONTINUE
C
      RETURN
      END


      SUBROUTINE IBLSYS
C---------------------------------------------
C     Sets the BL Newton system line number
C     corresponding to each BL station.
C---------------------------------------------
      INCLUDE 'XFOIL.INC'
      INCLUDE 'XBL.INC'
C
      IV = 0
      DO 10 IS=1, 2
        DO 110 IBL=2, NBL(IS)
          IV = IV+1
          ISYS(IBL,IS) = IV
  110   CONTINUE
   10 CONTINUE
C
```

```
      NSYS = IV
      IF(NSYS.GT.2*IVX) STOP '*** IBLSYS: BL system array overflow. ***'
C
      RETURN
      END


      SUBROUTINE MRCHUE
C----------------------------------------------------
C     Marches the BLs and wake in direct mode using
C     the UEDG array. If separation is encountered,
C     a plausible value of Hk extrapolated from
C     upstream is prescribed instead.  Continuous
C     checking of transition onset is performed.
C----------------------------------------------------
      INCLUDE 'XFOIL.INC'
      INCLUDE 'XBL.INC'
      LOGICAL DIRECT
      REAL MSQ
C
C---- shape parameters for separation criteria
      HLMAX = 3.8
      HTMAX = 2.5
C
      DO 2000 IS=1, 2
C
      WRITE(*,*) '   side ', IS, ' ...'
C
C---- set forced transition arc length position
      CALL XIFSET(IS)
C
C---- initialize similarity station with Thwaites' formula
      IBL = 2
      XSI = XSSI(IBL,IS)
      UEI = UEDG(IBL,IS)
C      BULE = LOG(UEDG(IBL+1,IS)/UEI) / LOG(XSSI(IBL+1,IS)/XSI)
C      BULE = MAX( -.08 , BULE )
      BULE = 1.0
      UCON = UEI/XSI**BULE
      TSQ = 0.45/(UCON*(5.0*BULE+1.0)*REYBL) * XSI**(1.0-BULE)
      THI = SQRT(TSQ)
      DSI = 2.2*THI
      AMI = 0.0
C
C---- initialize Ctau for first turbulent station
      CTI = 0.03
C
      TRAN = .FALSE.
      TURB = .FALSE.
      ITRAN(IS) = IBLTE(IS)
C
C---- march downstream
      DO 1000 IBL=2, NBL(IS)
        IBM = IBL-1
C
        IW = IBL - IBLTE(IS)
C
        SIMI = IBL.EQ.2
        WAKE = IBL.GT.IBLTE(IS)
C
C------ prescribed quantities
        XSI = XSSI(IBL,IS)
        UEI = UEDG(IBL,IS)
C
        IF(WAKE) THEN
         IW = IBL - IBLTE(IS)
         DSWAKI = WGAP(IW)
        ELSE
         DSWAKI = 0.
        ENDIF
C
        DIRECT = .TRUE.
```

```
C
C------ Newton iteration loop for current station
        DO 100 ITBL=1, 25
C
C-------- assemble 10x3 linearized system for dCtau, dTh, dDs, dUe, dXi
C         at the previous "1" station and the current "2" station
C         (the "1" station coefficients will be ignored)
C
C
          CALL BLPRV(XSI,AMI,CTI,THI,DSI,DSWAKI,UEI)
          CALL BLKIN
C
C-------- check for transition and set appropriate flags and things
          IF((.NOT.SIMI) .AND. (.NOT.TURB)) THEN
           CALL TRCHEK
           AMI = AMPL2
C
C--------- fixed BUG   MD 7 Jun 99
           IF(TRAN) THEN
            ITRAN(IS) = IBL
            IF(CTI.LE.0.0) THEN
             CTI = 0.03
             S2 = CTI
            ENDIF
           ELSE
            ITRAN(IS) = IBL+2
           ENDIF
C
C
          ENDIF
C
          IF(IBL.EQ.IBLTE(IS)+1) THEN
           TTE = THET(IBLTE(1),1) + THET(IBLTE(2),2)
           DTE = DSTR(IBLTE(1),1) + DSTR(IBLTE(2),2) + ANTE
           CTE = ( CTAU(IBLTE(1),1)*THET(IBLTE(1),1)
     &           + CTAU(IBLTE(2),2)*THET(IBLTE(2),2) ) / TTE
           CALL TESYS(CTE,TTE,DTE)
          ELSE
           CALL BLSYS
          ENDIF
C
          IF(DIRECT) THEN
C
C--------- try direct mode (set dUe = 0 in currently empty 4th line)
           VS2(4,1) = 0.
           VS2(4,2) = 0.
           VS2(4,3) = 0.
           VS2(4,4) = 1.0
           VSREZ(4) = 0.
C
C--------- solve Newton system for current "2" station
           CALL GAUSS(4,4,VS2,VSREZ,1)
C
C--------- determine max changes and underrelax if necessary
           DMAX = MAX( ABS(VSREZ(2)/THI),
     &                 ABS(VSREZ(3)/DSI)  )
           IF(IBL.LT.ITRAN(IS)) DMAX = MAX(DMAX,ABS(VSREZ(1)/10.0))
           IF(IBL.GE.ITRAN(IS)) DMAX = MAX(DMAX,ABS(VSREZ(1)/CTI ))
C
           RLX = 1.0
           IF(DMAX.GT.0.3) RLX = 0.3/DMAX
C
C--------- see if direct mode is not applicable
           IF(IBL .NE. IBLTE(IS)+1) THEN
C
C---------- calculate resulting kinematic shape parameter Hk
            MSQ = UEI*UEI*HSTINV / (GM1BL*(1.0 - 0.5*UEI*UEI*HSTINV))
            HTEST = (DSI + RLX*VSREZ(3)) / (THI + RLX*VSREZ(2))
            CALL HKIN( HTEST, MSQ, HKTEST, DUMMY, DUMMY)
C
C---------- decide whether to do direct or inverse problem based on Hk
            IF(IBL.LT.ITRAN(IS)) HMAX = HLMAX
```

```
            IF(IBL.GE.ITRAN(IS)) HMAX = HTMAX
            DIRECT = HKTEST.LT.HMAX
           ENDIF
C
           IF(DIRECT) THEN
C---------- update as usual
ccc            IF(IBL.LT.ITRAN(IS)) AMI = AMI + RLX*VSREZ(1)
            IF(IBL.GE.ITRAN(IS)) CTI = CTI + RLX*VSREZ(1)
            THI = THI + RLX*VSREZ(2)
            DSI = DSI + RLX*VSREZ(3)
           ELSE
C---------- set prescribed Hk for inverse calculation at the current station
            IF(IBL.LT.ITRAN(IS)) THEN
C----------- laminar case: relatively slow increase in Hk downstream
             HTARG = HK1 + 0.03*(X2-X1)/T1
            ELSE IF(IBL.EQ.ITRAN(IS)) THEN
C----------- transition interval: weighted laminar and turbulent case
             HTARG = HK1 + (0.03*(XT-X1) - 0.15*(X2-XT))/T1
            ELSE IF(WAKE) THEN
C----------- turbulent wake case:
C-           asymptotic wake behavior with approximate Backward Euler
             CONST = 0.03*(X2-X1)/T1
             HK2 = HK1
             HK2 = HK2 - (HK2 +     CONST*(HK2-1.0)**3 - HK1)
     &                  /(1.0 + 3.0*CONST*(HK2-1.0)**2)
             HK2 = HK2 - (HK2 +     CONST*(HK2-1.0)**3 - HK1)
     &                  /(1.0 + 3.0*CONST*(HK2-1.0)**2)
             HK2 = HK2 - (HK2 +     CONST*(HK2-1.0)**3 - HK1)
     &                  /(1.0 + 3.0*CONST*(HK2-1.0)**2)
             HTARG = HK2
            ELSE
C----------- turbulent case: relatively fast decrease in Hk downstream
             HTARG = HK1 - 0.15*(X2-X1)/T1
            ENDIF
C
C---------- limit specified Hk to something reasonable
            IF(WAKE) THEN
             HTARG = MAX( HTARG , 1.01 )
            ELSE
             HTARG = MAX( HTARG , HMAX )
            ENDIF
C
            WRITE(*,1300) IBL, HTARG
 1300       FORMAT(' MRCHUE: Inverse mode at', I4, '     Hk =', F8.3)
C
C---------- try again with prescribed Hk
            GO TO 100
C
           ENDIF
C
          ELSE
C
C-------- inverse mode (force Hk to prescribed value HTARG)
           VS2(4,1) = 0.
           VS2(4,2) = HK2_T2
           VS2(4,3) = HK2_D2
           VS2(4,4) = HK2_U2
           VSREZ(4) = HTARG - HK2
C
           CALL GAUSS(4,4,VS2,VSREZ,1)
C
           DMAX = MAX( ABS(VSREZ(2)/THI),
     &                 ABS(VSREZ(3)/DSI)  )
           IF(IBL.GE.ITRAN(IS)) DMAX = MAX( DMAX , ABS(VSREZ(1)/CTI))
C
           RLX = 1.0
           IF(DMAX.GT.0.3) RLX = 0.3/DMAX
C
C--------- update variables
ccc           IF(IBL.LT.ITRAN(IS)) AMI = AMI + RLX*VSREZ(1)
           IF(IBL.GE.ITRAN(IS)) CTI = CTI + RLX*VSREZ(1)
           THI = THI + RLX*VSREZ(2)
```

```
           DSI = DSI + RLX*VSREZ(3)
           UEI = UEI + RLX*VSREZ(4)
C
          ENDIF
C
C-------- eliminate absurd transients
          IF(IBL.GE.ITRAN(IS)) THEN
           CTI = MIN(CTI , 0.30 )
           CTI = MAX(CTI , 0.0000001 )
          ENDIF
C
          IF(IBL.LE.IBLTE(IS)) THEN
            HKLIM = 1.02
          ELSE
            HKLIM = 1.00005
          ENDIF
          MSQ = UEI*UEI*HSTINV / (GM1BL*(1.0 - 0.5*UEI*UEI*HSTINV))
          DSW = DSI - DSWAKI
          CALL DSLIM(DSW,THI,UEI,MSQ,HKLIM)
          DSI = DSW + DSWAKI
C
          IF(DMAX.LE.1.0E-5) GO TO 110
C
  100   CONTINUE
        WRITE(*,1350) IBL, IS, DMAX
 1350   FORMAT(' MRCHUE: Convergence failed at',I4,'  side',I2,
     &         '    Res =', E12.4)
C
C------ the current unconverged solution might still be reasonable...
CCC        IF(DMAX .LE. 0.1) GO TO 110
        IF(DMAX .LE. 0.1) GO TO 109
C
C------- the current solution is garbage --> extrapolate values instead
         IF(IBL.GT.3) THEN
          IF(IBL.LE.IBLTE(IS)) THEN
           THI = THET(IBM,IS) * (XSSI(IBL,IS)/XSSI(IBM,IS))**0.5
           DSI = DSTR(IBM,IS) * (XSSI(IBL,IS)/XSSI(IBM,IS))**0.5
          ELSE IF(IBL.EQ.IBLTE(IS)+1) THEN
           CTI = CTE
           THI = TTE
           DSI = DTE
          ELSE
           THI = THET(IBM,IS)
           RATLEN = (XSSI(IBL,IS)-XSSI(IBM,IS)) / (10.0*DSTR(IBM,IS))
           DSI = (DSTR(IBM,IS) + THI*RATLEN) / (1.0 + RATLEN)
          ENDIF
          IF(IBL.EQ.ITRAN(IS)) CTI = 0.05
          IF(IBL.GT.ITRAN(IS)) CTI = CTAU(IBM,IS)
C
          UEI = UEDG(IBL,IS)
          IF(IBL.GT.2 .AND. IBL.LT.NBL(IS))
     &     UEI = 0.5*(UEDG(IBL-1,IS) + UEDG(IBL+1,IS))
         ENDIF
C
 109     CALL BLPRV(XSI,AMI,CTI,THI,DSI,DSWAKI,UEI)
         CALL BLKIN
C
C------- check for transition and set appropriate flags and things
         IF((.NOT.SIMI) .AND. (.NOT.TURB)) THEN
          CALL TRCHEK
          AMI = AMPL2
          IF(     TRAN) ITRAN(IS) = IBL
          IF(.NOT.TRAN) ITRAN(IS) = IBL+2
         ENDIF
C
C------- set all other extrapolated values for current station
         IF(IBL.LT.ITRAN(IS)) CALL BLVAR(1)
         IF(IBL.GE.ITRAN(IS)) CALL BLVAR(2)
         IF(WAKE) CALL BLVAR(3)
C
         IF(IBL.LT.ITRAN(IS)) CALL BLMID(1)
         IF(IBL.GE.ITRAN(IS)) CALL BLMID(2)
```

```
         IF(WAKE) CALL BLMID(3)
C
C------ pick up here after the Newton iterations
  110   CONTINUE
C
C------ store primary variables
        IF(IBL.LT.ITRAN(IS)) CTAU(IBL,IS) = AMI
        IF(IBL.GE.ITRAN(IS)) CTAU(IBL,IS) = CTI
        THET(IBL,IS) = THI
        DSTR(IBL,IS) = DSI
        UEDG(IBL,IS) = UEI
        MASS(IBL,IS) = DSI*UEI
        TAU(IBL,IS)  = 0.5*R2*U2*U2*CF2
        DIS(IBL,IS)  =     R2*U2*U2*U2*DI2*HS2*0.5
        CTQ(IBL,IS)  = CQ2
        DELT(IBL,IS) = DE2
C
C------ set "1" variables to "2" variables for next streamwise station
        CALL BLPRV(XSI,AMI,CTI,THI,DSI,DSWAKI,UEI)
        CALL BLKIN
        DO 310 ICOM=1, NCOM
          COM1(ICOM) = COM2(ICOM)
  310   CONTINUE
C
C------ turbulent intervals will follow transition interval or TE
        IF(TRAN .OR. IBL.EQ.IBLTE(IS)) THEN
         TURB = .TRUE.
C
C------- save transition location
         TFORCE(IS) = TRFORC
         XSSITR(IS) = XT
        ENDIF
C
        TRAN = .FALSE.
C
        IF(IBL.EQ.IBLTE(IS)) THEN
         THI = THET(IBLTE(1),1) + THET(IBLTE(2),2)
         DSI = DSTR(IBLTE(1),1) + DSTR(IBLTE(2),2) + ANTE
        ENDIF
C
 1000 CONTINUE
 2000 CONTINUE
C
      RETURN
      END


      SUBROUTINE MRCHDU
C----------------------------------------------------
C     Marches the BLs and wake in mixed mode using
C     the current Ue and Hk.  The calculated Ue
C     and Hk lie along a line quasi-normal to the
C     natural Ue-Hk characteristic line of the
C     current BL so that the Goldstein or Levy-Lees
C     singularity is never encountered.  Continuous
C     checking of transition onset is performed.
C----------------------------------------------------
      INCLUDE 'XFOIL.INC'
      INCLUDE 'XBL.INC'
      REAL VTMP(4,5), VZTMP(4)
      REAL MSQ
ccc   REAL MDI
C
      DATA DEPS / 5.0E-6 /
C
C---- constant controlling how far Hk is allowed to deviate
C-    from the specified value.
      SENSWT = 1000.0
C
      DO 2000 IS=1, 2
C
C---- set forced transition arc length position
```

```
      CALL XIFSET(IS)
C
C---- set leading edge pressure gradient parameter  x/u du/dx
      IBL = 2
      XSI = XSSI(IBL,IS)
      UEI = UEDG(IBL,IS)
CCC      BULE = LOG(UEDG(IBL+1,IS)/UEI) / LOG(XSSI(IBL+1,IS)/XSI)
CCC      BULE = MAX( -.08 , BULE )
      BULE = 1.0
C
C---- old transition station
      ITROLD = ITRAN(IS)
C
      TRAN = .FALSE.
      TURB = .FALSE.
      ITRAN(IS) = IBLTE(IS)
C
C---- march downstream
      DO 1000 IBL=2, NBL(IS)
        IBM = IBL-1
C
        SIMI = IBL.EQ.2
        WAKE = IBL.GT.IBLTE(IS)
C
C------ initialize current station to existing variables
        XSI = XSSI(IBL,IS)
        UEI = UEDG(IBL,IS)
        THI = THET(IBL,IS)
        DSI = DSTR(IBL,IS)
CCC        MDI = MASS(IBL,IS)
C
C------ fixed BUG   MD 7 June 99
        IF(IBL.LT.ITROLD) THEN
         AMI = CTAU(IBL,IS)
         CTI = 0.03
        ELSE
         CTI = CTAU(IBL,IS)
         IF(CTI.LE.0.0) CTI = 0.03
        ENDIF
C
CCC        DSI = MDI/UEI
C
        IF(WAKE) THEN
         IW = IBL - IBLTE(IS)
         DSWAKI = WGAP(IW)
        ELSE
         DSWAKI = 0.
        ENDIF
C
        IF(IBL.LE.IBLTE(IS)) DSI = MAX(DSI-DSWAKI,1.02000*THI) + DSWAKI
        IF(IBL.GT.IBLTE(IS)) DSI = MAX(DSI-DSWAKI,1.00005*THI) + DSWAKI
C
C------ Newton iteration loop for current station
        DO 100 ITBL=1, 25
C
C-------- assemble 10x3 linearized system for dCtau, dTh, dDs, dUe, dXi
C         at the previous "1" station and the current "2" station
C         (the "1" station coefficients will be ignored)
C
C
          CALL BLPRV(XSI,AMI,CTI,THI,DSI,DSWAKI,UEI)
          CALL BLKIN
C
C-------- check for transition and set appropriate flags and things
          IF((.NOT.SIMI) .AND. (.NOT.TURB)) THEN
           CALL TRCHEK
           AMI = AMPL2
           IF(     TRAN) ITRAN(IS) = IBL
           IF(.NOT.TRAN) ITRAN(IS) = IBL+2
          ENDIF
C
          IF(IBL.EQ.IBLTE(IS)+1) THEN
```

```
           TTE = THET(IBLTE(1),1) + THET(IBLTE(2),2)
           DTE = DSTR(IBLTE(1),1) + DSTR(IBLTE(2),2) + ANTE
           CTE = ( CTAU(IBLTE(1),1)*THET(IBLTE(1),1)
     &           + CTAU(IBLTE(2),2)*THET(IBLTE(2),2) ) / TTE
           CALL TESYS(CTE,TTE,DTE)
          ELSE
           CALL BLSYS
          ENDIF
C
C-------- set stuff at first iteration...
          IF(ITBL.EQ.1) THEN
C
C--------- set "baseline" Ue and Hk for forming  Ue(Hk)  relation
           UEREF = U2
           HKREF = HK2
C
C--------- if current point IBL was turbulent and is now laminar, then...
           IF(IBL.LT.ITRAN(IS) .AND. IBL.GE.ITROLD ) THEN
C---------- extrapolate baseline Hk
            UEM = UEDG(IBL-1,IS)
            DSM = DSTR(IBL-1,IS)
            THM = THET(IBL-1,IS)
            MSQ = UEM*UEM*HSTINV / (GM1BL*(1.0 - 0.5*UEM*UEM*HSTINV))
            CALL HKIN( DSM/THM, MSQ, HKREF, DUMMY, DUMMY )
           ENDIF
C
C--------- if current point IBL was laminar, then...
           IF(IBL.LT.ITROLD) THEN
C---------- reinitialize or extrapolate Ctau if it's now turbulent
            IF(TRAN) CTAU(IBL,IS) = 0.03
            IF(TURB) CTAU(IBL,IS) = CTAU(IBL-1,IS)
            IF(TRAN .OR. TURB) THEN
             CTI = CTAU(IBL,IS)
             S2 = CTI
            ENDIF
           ENDIF
C
          ENDIF
C
C
          IF(SIMI .OR. IBL.EQ.IBLTE(IS)+1) THEN
C
C--------- for similarity station or first wake point, prescribe Ue
           VS2(4,1) = 0.
           VS2(4,2) = 0.
           VS2(4,3) = 0.
           VS2(4,4) = U2_UEI
           VSREZ(4) = UEREF - U2
C
          ELSE
C
C********* calculate Ue-Hk characteristic slope
C
           DO 20 K=1, 4
             VZTMP(K) = VSREZ(K)
             DO 201 L=1, 5
               VTMP(K,L) = VS2(K,L)
  201        CONTINUE
   20      CONTINUE
C
C--------- set unit dHk
           VTMP(4,1) = 0.
           VTMP(4,2) = HK2_T2
           VTMP(4,3) = HK2_D2
           VTMP(4,4) = HK2_U2*U2_UEI
           VZTMP(4)  = 1.0
C
C--------- calculate dUe response
           CALL GAUSS(4,4,VTMP,VZTMP,1)
C
C--------- set  SENSWT * (normalized dUe/dHk)
           SENNEW = SENSWT * VZTMP(4) * HKREF/UEREF
```

```
           IF(ITBL.LE.5) THEN
            SENS = SENNEW
           ELSE IF(ITBL.LE.15) THEN
            SENS = 0.5*(SENS + SENNEW)
           ENDIF
C
C--------- set prescribed Ue-Hk combination
           VS2(4,1) = 0.
           VS2(4,2) =  HK2_T2 * HKREF
           VS2(4,3) =  HK2_D2 * HKREF
           VS2(4,4) =( HK2_U2 * HKREF  +  SENS/UEREF )*U2_UEI
           VSREZ(4) = -(HKREF**2)*(HK2 / HKREF - 1.0)
     &                     - SENS*(U2  / UEREF - 1.0)
C
          ENDIF
C
C-------- solve Newton system for current "2" station
          CALL GAUSS(4,4,VS2,VSREZ,1)
C
C-------- determine max changes and underrelax if necessary
          DMAX = MAX( ABS(VSREZ(2)/THI),
     &                ABS(VSREZ(3)/DSI)  )
          IF(IBL.GE.ITRAN(IS)) DMAX = MAX(DMAX,ABS(VSREZ(1)/(10.0*CTI)))
C
          RLX = 1.0
          IF(DMAX.GT.0.3) RLX = 0.3/DMAX
C
C-------- update as usual
          IF(IBL.LT.ITRAN(IS)) AMI = AMI + RLX*VSREZ(1)
          IF(IBL.GE.ITRAN(IS)) CTI = CTI + RLX*VSREZ(1)
          THI = THI + RLX*VSREZ(2)
          DSI = DSI + RLX*VSREZ(3)
          UEI = UEI + RLX*VSREZ(4)
C
C-------- eliminate absurd transients
          IF(IBL.GE.ITRAN(IS)) THEN
           CTI = MIN(CTI , 0.30 )
           CTI = MAX(CTI , 0.0000001 )
          ENDIF
C
          IF(IBL.LE.IBLTE(IS)) THEN
            HKLIM = 1.02
          ELSE
            HKLIM = 1.00005
          ENDIF
          MSQ = UEI*UEI*HSTINV / (GM1BL*(1.0 - 0.5*UEI*UEI*HSTINV))
          DSW = DSI - DSWAKI
          CALL DSLIM(DSW,THI,UEI,MSQ,HKLIM)
          DSI = DSW + DSWAKI
C
          IF(DMAX.LE.DEPS) GO TO 110
C
  100   CONTINUE
C
        WRITE(*,1350) IBL, IS, DMAX
 1350   FORMAT(' MRCHDU: Convergence failed at',I4,'  side',I2,
     &         '    Res =', E12.4)
C
C------ the current unconverged solution might still be reasonable...
CCC        IF(DMAX .LE. 0.1) GO TO 110
        IF(DMAX .LE. 0.1) GO TO 109
C
C------- the current solution is garbage --> extrapolate values instead
         IF(IBL.GT.3) THEN
          IF(IBL.LE.IBLTE(IS)) THEN
           THI = THET(IBM,IS) * (XSSI(IBL,IS)/XSSI(IBM,IS))**0.5
           DSI = DSTR(IBM,IS) * (XSSI(IBL,IS)/XSSI(IBM,IS))**0.5
           UEI = UEDG(IBM,IS)
          ELSE IF(IBL.EQ.IBLTE(IS)+1) THEN
           CTI = CTE
           THI = TTE
           DSI = DTE
```

```
           UEI = UEDG(IBM,IS)
          ELSE
           THI = THET(IBM,IS)
           RATLEN = (XSSI(IBL,IS)-XSSI(IBM,IS)) / (10.0*DSTR(IBM,IS))
           DSI = (DSTR(IBM,IS) + THI*RATLEN) / (1.0 + RATLEN)
           UEI = UEDG(IBM,IS)
          ENDIF
          IF(IBL.EQ.ITRAN(IS)) CTI = 0.05
          IF(IBL.GT.ITRAN(IS)) CTI = CTAU(IBM,IS)
         ENDIF
C
 109     CALL BLPRV(XSI,AMI,CTI,THI,DSI,DSWAKI,UEI)
         CALL BLKIN
C
C------- check for transition and set appropriate flags and things
         IF((.NOT.SIMI) .AND. (.NOT.TURB)) THEN
          CALL TRCHEK
          AMI = AMPL2
          IF(     TRAN) ITRAN(IS) = IBL
          IF(.NOT.TRAN) ITRAN(IS) = IBL+2
         ENDIF
C
C------- set all other extrapolated values for current station
         IF(IBL.LT.ITRAN(IS)) CALL BLVAR(1)
         IF(IBL.GE.ITRAN(IS)) CALL BLVAR(2)
         IF(WAKE) CALL BLVAR(3)
C
         IF(IBL.LT.ITRAN(IS)) CALL BLMID(1)
         IF(IBL.GE.ITRAN(IS)) CALL BLMID(2)
         IF(WAKE) CALL BLMID(3)
C
C------ pick up here after the Newton iterations
  110   CONTINUE
C
        SENS = SENNEW
C
C------ store primary variables
        IF(IBL.LT.ITRAN(IS)) CTAU(IBL,IS) = AMI
        IF(IBL.GE.ITRAN(IS)) CTAU(IBL,IS) = CTI
        THET(IBL,IS) = THI
        DSTR(IBL,IS) = DSI
        UEDG(IBL,IS) = UEI
        MASS(IBL,IS) = DSI*UEI
        TAU(IBL,IS)  = 0.5*R2*U2*U2*CF2
        DIS(IBL,IS)  =     R2*U2*U2*U2*DI2*HS2*0.5
        CTQ(IBL,IS)  = CQ2
C
C------ set "1" variables to "2" variables for next streamwise station
        CALL BLPRV(XSI,AMI,CTI,THI,DSI,DSWAKI,UEI)
        CALL BLKIN
        DO 310 ICOM=1, NCOM
          COM1(ICOM) = COM2(ICOM)
  310   CONTINUE
C
C
C------ turbulent intervals will follow transition interval or TE
        IF(TRAN .OR. IBL.EQ.IBLTE(IS)) THEN
         TURB = .TRUE.
C
C------- save transition location
         TFORCE(IS) = TRFORC
         XSSITR(IS) = XT
        ENDIF
C
        TRAN = .FALSE.
C
 1000 CONTINUE
C
 2000 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE XIFSET(IS)
C-----------------------------------------------------
C     Sets forced-transition BL coordinate locations.
C-----------------------------------------------------
      INCLUDE 'XFOIL.INC'
      INCLUDE 'XBL.INC'
C
      IF(XSTRIP(IS).GE.1.0) THEN
       XIFORC = XSSI(IBLTE(IS),IS)
       RETURN
      ENDIF
C
      CHX = XTE - XLE
      CHY = YTE - YLE
      CHSQ = CHX**2 + CHY**2
C
C---- calculate chord-based x/c, y/c
      DO 10 I=1, N
        W1(I) = ((X(I)-XLE)*CHX + (Y(I)-YLE)*CHY) / CHSQ
        W2(I) = ((Y(I)-YLE)*CHX - (X(I)-XLE)*CHY) / CHSQ
 10   CONTINUE
C
      CALL SPLIND(W1,W3,S,N,-999.0,-999.0)
      CALL SPLIND(W2,W4,S,N,-999.0,-999.0)
C
      IF(IS.EQ.1) THEN
C
C----- set approximate arc length of forced transition point for SINVRT
       STR = SLE + (S(1)-SLE)*XSTRIP(IS)
C
C----- calculate actual arc length
       CALL SINVRT(STR,XSTRIP(IS),W1,W3,S,N)
C
C----- set BL coordinate value
       XIFORC = MIN( (SST - STR) , XSSI(IBLTE(IS),IS) )
C
      ELSE
C----- same for bottom side
C
       STR = SLE + (S(N)-SLE)*XSTRIP(IS)
       CALL SINVRT(STR,XSTRIP(IS),W1,W3,S,N)
       XIFORC = MIN( (STR - SST) , XSSI(IBLTE(IS),IS) )
C
      ENDIF
C
      IF(XIFORC .LT. 0.0) THEN
       WRITE(*,1000) IS
 1000  FORMAT(/' ***  Stagnation point is past trip on side',I2,'  ***')
       XIFORC = XSSI(IBLTE(IS),IS)
      ENDIF
C
      RETURN
      END



      SUBROUTINE UPDATE(sec,pcor)
C------------------------------------------------------------------
C      Adds on Newton deltas to boundary layer variables.
C      Checks for excessive changes and underrelaxes if necessary.
C      Calculates max and rms changes.
C      Also calculates the change in the global variable "AC".
C        If LALFA=.TRUE. , "AC" is CL
C        If LALFA=.FALSE., "AC" is alpha
C------------------------------------------------------------------
      INCLUDE 'XFOIL.INC'
      include 'section.inc' !02/06/2005

      type(section):: sec
```

```
      REAL UNEW(IVX,2), U_AC(IVX,2)
     real:: pcor(iqx,2)  !mass defect correction (22/05/2005)
      REAL:: QNEW(IQX), Q_AC(IQX), Qcorr(IQX) !viscous correction
      EQUIVALENCE (VA(1,1,1), UNEW(1,1)) ,
     &            (VB(1,1,1), QNEW(1)  )
      EQUIVALENCE (VA(1,1,IVX), U_AC(1,1)) ,
     &            (VB(1,1,IVX), Q_AC(1)  )
      REAL MSQ
C
C---- max allowable alpha changes per iteration
      DALMAX =  0.5*DTOR
      DALMIN = -0.5*DTOR
C
C---- max allowable CL change per iteration
      DCLMAX =  0.5
      DCLMIN = -0.5
      IF(MATYP.NE.1) DCLMIN = MAX(-0.5 , -0.9*CL)
C
      HSTINV = GAMM1*(MINF/QINF)**2 / (1.0 + 0.5*GAMM1*MINF**2)
C
C---- calculate new Ue distribution assuming no under-relaxation
C-    also set the sensitivity of Ue wrt to alpha or Re
      DO 1 IS=1, 2
        DO 10 IBL=2, NBL(IS)
          I = IPAN(IBL,IS)
C
          DUI    = 0.
          DUI_AC = 0.
          DO 100 JS=1, 2
            DO 1000 JBL=2, NBL(JS)
              J  = IPAN(JBL,JS)
              JV = ISYS(JBL,JS)
              UE_M = -VTI(IBL,IS)*VTI(JBL,JS)*DIJ(I,J)
              DUI    = DUI    + UE_M*(MASS(JBL,JS)+VDEL(3,1,JV))
              DUI_AC = DUI_AC + UE_M*(            -VDEL(3,2,JV))
 1000       CONTINUE
  100     CONTINUE
C
C-------- UINV depends on "AC" only if "AC" is alpha
          IF(LALFA) THEN
           UINV_AC = 0.
          ELSE
           UINV_AC = UINV_A(IBL,IS)
          ENDIF
C
          pcor(ibl,is) = DUI !viscous correction vector (22/05/2005)
            UNEW(IBL,IS) = UINV(IBL,IS) + DUI
          U_AC(IBL,IS) = UINV_AC      + DUI_AC
C
   10   CONTINUE
    1 CONTINUE
C
C---- set new Qtan from new Ue with appropriate sign change
      DO 2 IS=1, 2
        DO 20 IBL=2, IBLTE(IS)
          I = IPAN(IBL,IS)
          Qcorr(i) = VTI(IBL,IS)*pcor(IBL,IS) !added 02/06/2005
          QNEW(I) = VTI(IBL,IS)*UNEW(IBL,IS)
          Q_AC(I) = VTI(IBL,IS)*U_AC(IBL,IS)
   20   CONTINUE
    2 CONTINUE
      QNEW(IST) = 0.  !correction on 30/03/2004
C
C---- calculate new CL from this new Qtan
      SA = SIN(ALFA)
      CA = COS(ALFA)
C
      BETA = SQRT(1.0 - MINF**2)
      BETA_MSQ = -0.5/BETA
C
      BFAC     = 0.5*MINF**2 / (1.0 + BETA)
      BFAC_MSQ = 0.5         / (1.0 + BETA)
```

```
     &         - BFAC        / (1.0 + BETA) * BETA_MSQ
C
      CLNEW = 0.
      CL_A  = 0.
      CL_MS = 0.
      CL_AC = 0.
C
      I = 1
      !CGINC = 1.0 - (QNEW(I)/QINF)**2
      CGINC = sec.cp(i)+ Qcorr(i)**2
      CPG1  = CGINC / (BETA + BFAC*CGINC)
      CPG1_MS = -CPG1/(BETA + BFAC*CGINC)*(BETA_MSQ + BFAC_MSQ*CGINC)
C
      !CPI_Q = -2.0*QNEW(I)/QINF**2
      CPI_Q = -2.0*(1-sec.cp(i))
      CPC_CPI = (1.0 - BFAC*CPG1)/ (BETA + BFAC*CGINC)
      CPG1_AC = CPC_CPI*CPI_Q*Q_AC(I)
C
      DO 3 I=1, N
        IP = I+1
        IF(I.EQ.N) IP = 1
C
        !CGINC = 1.0 - (QNEW(IP)/QINF)**2
        CGINC = sec.cp(i)+ Qcorr(i)**2
        CPG2  = CGINC / (BETA + BFAC*CGINC)
        CPG2_MS = -CPG2/(BETA + BFAC*CGINC)*(BETA_MSQ + BFAC_MSQ*CGINC)
C
        CPI_Q = -2.0*(1-sec.cp(i))
        CPC_CPI = (1.0 - BFAC*CPG2)/ (BETA + BFAC*CGINC)
        CPG2_AC = CPC_CPI*CPI_Q*Q_AC(IP)
C
        DX   =  (X(IP) - X(I))*CA + (Y(IP) - Y(I))*SA
        DX_A = -(X(IP) - X(I))*SA + (Y(IP) - Y(I))*CA
C
        AG    = 0.5*(CPG2    + CPG1   )
        AG_MS = 0.5*(CPG2_MS + CPG1_MS)
        AG_AC = 0.5*(CPG2_AC + CPG1_AC)
C
        CLNEW = CLNEW + DX  *AG
        CL_A  = CL_A  + DX_A*AG
        CL_MS = CL_MS + DX  *AG_MS
        CL_AC = CL_AC + DX  *AG_AC
C
        CPG1    = CPG2
        CPG1_MS = CPG2_MS
        CPG1_AC = CPG2_AC
    3 CONTINUE
C
C---- initialize under-relaxation factor
      RLX = 1.0
C
      IF(LALFA) THEN
C===== alpha is prescribed: AC is CL
C
C----- set change in Re to account for CL changing, since Re = Re(CL)
       DAC = (CLNEW - CL) / (1.0 - CL_AC - CL_MS*2.0*MINF*MINF_CL)
C
C----- set under-relaxation factor if Re change is too large
       IF(RLX*DAC .GT. DCLMAX) RLX = DCLMAX/DAC
       IF(RLX*DAC .LT. DCLMIN) RLX = DCLMIN/DAC
C
      ELSE
C===== CL is prescribed: AC is alpha
C
C----- set change in alpha to drive CL to prescribed value
       DAC = (CLNEW - CLSPEC) / (0.0 - CL_AC - CL_A)
C
C----- set under-relaxation factor if alpha change is too large
       IF(RLX*DAC .GT. DALMAX) RLX = DALMAX/DAC
       IF(RLX*DAC .LT. DALMIN) RLX = DALMIN/DAC
C
      ENDIF
```

```
C
      RMSBL = 0.
      RMXBL = 0.
C
      DHI = 1.5
      DLO = -.5
C
C---- calculate changes in BL variables and under-relaxation if needed
      DO 4 IS=1, 2
        DO 40 IBL=2, NBL(IS)
          IV = ISYS(IBL,IS)
C
C-------- set changes without underrelaxation
          DCTAU = VDEL(1,1,IV) - DAC*VDEL(1,2,IV)
          DTHET = VDEL(2,1,IV) - DAC*VDEL(2,2,IV)
          DMASS = VDEL(3,1,IV) - DAC*VDEL(3,2,IV)
          DUEDG = UNEW(IBL,IS) + DAC*U_AC(IBL,IS)  -  UEDG(IBL,IS)
          DDSTR = (DMASS - DSTR(IBL,IS)*DUEDG)/UEDG(IBL,IS)
C
C-------- normalize changes  !all corrected 27/06/2004
          IF(IBL.LT.ITRAN(IS)) DN1 = DCTAU / 10.0
          IF(IBL.GE.ITRAN(IS)) DN1 = DCTAU / CTAU(IBL,IS)
          DN2 = DTHET / THET(IBL,IS)
          DN3 = DDSTR / DSTR(IBL,IS)
          DN4 = ABS(DUEDG)/0.25

         if (iv<=iblte(is)) then  !just for body 30/03/2004
C
C-------- accumulate for rms change
          RMSBL = RMSBL + DN1**2 + DN2**2 + DN3**2 + DN4**2
          endif
C
C-------- see if Ctau needs underrelaxation
          RDN1 = RLX*DN1
          IF(ABS(DN1) .GT. ABS(RMXBL)) THEN
           RMXBL = DN1
           IF(IBL.LT.ITRAN(IS)) VMXBL = 'n'
           IF(IBL.GE.ITRAN(IS)) VMXBL = 'C'
           IMXBL = IBL
           ISMXBL = IS
          ENDIF
          IF(RDN1 .GT. DHI) RLX = DHI/DN1
          IF(RDN1 .LT. DLO) RLX = DLO/DN1
C
C-------- see if Theta needs underrelaxation
          RDN2 = RLX*DN2
          IF(ABS(DN2) .GT. ABS(RMXBL)) THEN
           RMXBL = DN2
           VMXBL = 'T'
           IMXBL = IBL
           ISMXBL = IS
          ENDIF
          IF(RDN2 .GT. DHI) RLX = DHI/DN2
          IF(RDN2 .LT. DLO) RLX = DLO/DN2
C
C-------- see if Dstar needs underrelaxation
          RDN3 = RLX*DN3
          IF(ABS(DN3) .GT. ABS(RMXBL)) THEN
           RMXBL = DN3
           VMXBL = 'D'
           IMXBL = IBL
           ISMXBL = IS
          ENDIF
          IF(RDN3 .GT. DHI) RLX = DHI/DN3
          IF(RDN3 .LT. DLO) RLX = DLO/DN3
C
C-------- see if Ue needs underrelaxation
          RDN4 = RLX*DN4
          IF(ABS(DN4) .GT. ABS(RMXBL)) THEN
           RMXBL = DUEDG
           VMXBL = 'U'
           IMXBL = IBL
```

```
           ISMXBL = IS
          ENDIF
          IF(RDN4 .GT. DHI) RLX = DHI/DN4
          IF(RDN4 .LT. DLO) RLX = DLO/DN4
C
   40   CONTINUE
    4 CONTINUE
C
C---- set true rms change
      RMSBL = SQRT( RMSBL / (4.0*FLOAT( NBL(1)+NBL(2) )) ) !/1000 !14/04/2004
C
C
      IF(LALFA) THEN
C----- set underrelaxed change in Reynolds number from change in lift
       CL = CL + RLX*DAC
      ELSE
C----- set underrelaxed change in alpha
       ALFA = ALFA + RLX*DAC
       ADEG = ALFA/DTOR
      ENDIF
C
C---- update BL variables with underrelaxed changes
      DO 5 IS=1, 2
        DO 50 IBL=2, NBL(IS)
          IV = ISYS(IBL,IS)
C
          DCTAU = VDEL(1,1,IV) - DAC*VDEL(1,2,IV)
          DTHET = VDEL(2,1,IV) - DAC*VDEL(2,2,IV)
          DMASS = VDEL(3,1,IV) - DAC*VDEL(3,2,IV)
          DUEDG = UNEW(IBL,IS) + DAC*U_AC(IBL,IS)  -  UEDG(IBL,IS)
          DDSTR = (DMASS - DSTR(IBL,IS)*DUEDG)/UEDG(IBL,IS)
C
          CTAU(IBL,IS) = CTAU(IBL,IS) + RLX*DCTAU
          THET(IBL,IS) = THET(IBL,IS) + RLX*DTHET
          DSTR(IBL,IS) = DSTR(IBL,IS) + RLX*DDSTR
          UEDG(IBL,IS) = UEDG(IBL,IS) + RLX*DUEDG
C
          IF(IBL.GT.IBLTE(IS)) THEN
           IW = IBL - IBLTE(IS)
           DSWAKI = WGAP(IW)
          ELSE
           DSWAKI = 0.
          ENDIF
C
C-------- eliminate absurd transients
          IF(IBL.GE.ITRAN(IS))
     &      CTAU(IBL,IS) = MIN( CTAU(IBL,IS) , 0.25 )
C
          IF(IBL.LE.IBLTE(IS)) THEN
            HKLIM = 1.02
          ELSE
            HKLIM = 1.00005
          ENDIF
          MSQ = UEDG(IBL,IS)**2*HSTINV
     &        / (GAMM1*(1.0 - 0.5*UEDG(IBL,IS)**2*HSTINV))
          DSW = DSTR(IBL,IS) - DSWAKI
          CALL DSLIM(DSW,THET(IBL,IS),UEDG(IBL,IS),MSQ,HKLIM)
          DSTR(IBL,IS) = DSW + DSWAKI
C
C-------- set new mass defect (nonlinear update)
          MASS(IBL,IS) = DSTR(IBL,IS) * UEDG(IBL,IS)
C
   50   CONTINUE
    5 CONTINUE
C
C
C---- equate upper wake arrays to lower wake arrays
      DO 6 KBL=1, NBL(2)-IBLTE(2)
        CTAU(IBLTE(1)+KBL,1) = CTAU(IBLTE(2)+KBL,2)
        THET(IBLTE(1)+KBL,1) = THET(IBLTE(2)+KBL,2)
        DSTR(IBLTE(1)+KBL,1) = DSTR(IBLTE(2)+KBL,2)
        UEDG(IBLTE(1)+KBL,1) = UEDG(IBLTE(2)+KBL,2)
```

```
         TAU(IBLTE(1)+KBL,1) =  TAU(IBLTE(2)+KBL,2)
         DIS(IBLTE(1)+KBL,1) =  DIS(IBLTE(2)+KBL,2)
         CTQ(IBLTE(1)+KBL,1) =  CTQ(IBLTE(2)+KBL,2)
 6    CONTINUE
C
      RETURN
      END



      SUBROUTINE DSLIM(DSTR,THET,UEDG,MSQ,HKLIM)
      IMPLICIT REAL (A-H,M,O-Z)
C
      H = DSTR/THET
      CALL HKIN(H,MSQ,HK,HK_H,HK_M)
C
      DH = MAX( 0.0 , HKLIM-HK ) / HK_H
      DSTR = DSTR + DH*THET
C
      RETURN
      END
```

VIX Subroutine Stagpoint

```
      !This subroutine finds the stagnation point of
      ! a interpolated section. If it is a membrane,
      ! fsharp = true, program will set the stagnation
      ! point at the very leading edge
      !***********************************************
      ! Created by: Augusto Veiga,
      ! FSIG, University of Southampton 2003
      !***********************************************

      subroutine stagpoint(Up,s,gamma,x,N,Nw,Ist,SST,SSt_GO,SST_GP,
     &   fsharp)
     integer:: N,Nw,IST,IS
     real:: cpmax,h
     real:: dcp1,dcp2
     real,dimension(N+Nw):: Up,s,x,gamma,cp
      logical:: fsharp

     !Calculating Cp over the wing
     do i = 1,N
        cp(i) = 1.-Up(i)**2
     enddo
      if (fsharp) then
       ist = int(N/2)+1
       sst = s(ist)
       is = ist
     else
       !Getting biggest Cp and position
       cpmax = 0
       is = int(N/2)+1
       k  = is
        i  = is
       lp1: do
        i = i+1
        if (cp(i)>cpmax) then
           cpmax = cp(i)
           is = i       !finding a possible point
           h = s(i)-s(i-1)
           dcp1 = (cpmax-cp(i-1))/h
               dcp2 = (cp(i+1)-cpmax)/h
               if ((dcp1>0 .and. dcp2<0)) then
                  ist = is  !testing derivatives
              sst = s(is)
              exit lp1
           endif
         else if (i>(k+int(N/2))) then
           is = int(N/2)+1
           sst = s(is)
           exit lp1
```

```
          endif
        enddo lp1
      endif
      !signal for gamma
      gamma(ist) = 0
      do i = 1,IST-1  !IST+1,N
          gamma(i) = Up(i)
      enddo
       !upper part
      do i = IST+1,N
         gamma(i) = -Up(i)
      enddo
      !wake
      do i = N+1,N+Nw
         gamma(i) = Up(i)
      enddo
       DGAM = GAMMA(IST+1) - GAMMA(IST)
      SST_GO = (SST - S(Ist+1))/DGAM
      SST_GP = (S(Ist+1) - SST)/DGAM
       return
      end subroutine
```

VIX Subroutine AIJCALC

```
!*************************************************************************************
      SUBROUTINE AIJCALC
C---------------------------------------------------------------
C     Calculates two surface vorticity (gamma) distributions
C     for alpha = 0, 90  degrees.  These are superimposed
C     in SPECAL or SPECCL for specified alpha or CL.
C     This subroutine was adapted from XFOIL by Augusto Veiga
C---------------------------------------------------------------
      INCLUDE 'XFOIL.INC'
C
C---- distance of internal control point ahead of sharp TE
C-    (fraction of smaller panel length adjacent to TE)
      BWT = 0.1
C
      WRITE(*,*) 'Calculating unit vorticity distributions ...'
C
      DO 10 I=1, N
!        GAM(I) = 0.
        GAMU(I,1) = 0.
        GAMU(I,2) = 0.
   10 CONTINUE
      PSIO = 0.
C
C---- Set up matrix system for  Psi = Psio  on airfoil surface.
C-    The unknowns are (dGamma)i and dPsio.
      DO 20 I=1, N
C
C------ calculate Psi and dPsi/dGamma array for current node
        CALL PSILIN(I,X(I),Y(I),NX(I),NY(I),PSI,PSI_N,.FALSE.,.TRUE.)
C
        PSIINF = QINF*(COS(ALFA)*Y(I) - SIN(ALFA)*X(I))
C
C------ RES1 = PSI( 0) - PSIO
C------ RES2 = PSI(90) - PSIO
        RES1 =  QINF*Y(I)
        RES2 = -QINF*X(I)
C
C------ dRes/dGamma
        DO 201 J=1, N
          AIJ(I,J) = DZDG(J)
  201   CONTINUE
C
        DO 202 J=1, N
          BIJ(I,J) = -DZDM(J)
  202   CONTINUE
C
C------ dRes/dPsio
```

```
         AIJ(I,N+1) = -1.0
C
         GAMU(I,1) = -RES1
         GAMU(I,2) = -RES2
C
   20 CONTINUE
C
C---- set Kutta condition
C-    RES = GAM(1) + GAM(N)
      RES = 0.
C
      DO 30 J=1, N+1
        AIJ(N+1,J) = 0.0
   30 CONTINUE
C
      AIJ(N+1,1) = 1.0
      AIJ(N+1,N) = 1.0
C
      GAMU(N+1,1) = -RES
      GAMU(N+1,2) = -RES
C
C---- set up Kutta condition (no direct source influence)
      DO 32 J=1, N
        BIJ(N+1,J) = 0.
   32 CONTINUE
C
      IF(SHARP) THEN
C----- set zero internal velocity in TE corner
C
C----- set TE bisector angle
       AG1 = ATAN2(-YP(1),-XP(1)    )
       AG2 = ATANC( YP(N), XP(N),AG1)
       ABIS = 0.5*(AG1+AG2)
       CBIS = COS(ABIS)
       SBIS = SIN(ABIS)
C
C----- minimum panel length adjacent to TE
       DS1 = SQRT( (X(1)-X(2)  )**2 + (Y(1)-Y(2)  )**2 )
       DS2 = SQRT( (X(N)-X(N-1))**2 + (Y(N)-Y(N-1))**2 )
       DSMIN = MIN( DS1 , DS2 )
C
C----- control point on bisector just ahead of TE point
       XBIS = XTE - BWT*DSMIN*CBIS
       YBIS = YTE - BWT*DSMIN*SBIS
ccc       write(*,*) xbis, ybis
C
C----- set velocity component along bisector line
       CALL PSILIN(0,XBIS,YBIS,-SBIS,CBIS,PSI,QBIS,.FALSE.,.TRUE.)
C
CCC--- RES = DQDGj*Gammaj + DQDMj*Massj + QINF*(COSA*CBIS + SINA*SBIS)
       RES = QBIS
C
C----- dRes/dGamma
       DO J=1, N
         AIJ(N,J) = DQDG(J)
       ENDDO
C
C----- -dRes/dMass
       DO J=1, N
         BIJ(N,J) = -DQDM(J)
       ENDDO
C
C----- dRes/dPsio
       AIJ(N,N+1) = 0.
C
C----- -dRes/dUinf
       GAMU(N,1) = -CBIS
C
C----- -dRes/dVinf
       GAMU(N,2) = -SBIS
C
      ENDIF
```

```
C
C---- LU-factor coefficient matrix AIJ
      CALL LUDCMP(IQX,N+1,AIJ,AIJPIV)
      LQAIJ = .TRUE.
C
C---- solve system for the two vorticity distributions
      CALL BAKSUB(IQX,N+1,AIJ,AIJPIV,GAMU(1,1))
      CALL BAKSUB(IQX,N+1,AIJ,AIJPIV,GAMU(1,2))
C
C---- set inviscid alpha=0,90 surface speeds for this geometry
      DO 50 I=1, N
        QINVU(I,1) = GAMU(I,1)
        QINVU(I,2) = GAMU(I,2)
   50 CONTINUE
C
      LGAMU = .TRUE.
C
      RETURN
      END
```

Mesh_Sail: Program for Creating Sail Mesh

```
      Program Mesh_sail
      include 'section.inc'

      real:: length, height, aflow, p(3),dt
      real:: org(3),x(3),y(3) !origin
     real,dimension(3,3):: vr
      type(section):: csec(7),tesec
     real:: intquad

      pi = 3.1415
      open(1,file = 'dados.txt')

      !reading height,footleng and flow incidence
         read(1,*) height,length,aflow
      !reading sections
     ! length, entry angle, te angle
        j = 1
         do i = 1,4
           read(1,*) csec(j).leng,csec(j).th1,csec(j).th2
          j = j+2
         enddo
      !reading trailing edge (te) section
         read(1,*) trv,abat,tesec.cpos,tesec.camber
     !reading foot and top section angles
        read(1,*) alfa1,beta1,alfa2,beta2
      close(1)
      dt = 1./3
     t = 0
     do i=1,7,2
        csec(i).t = t
       t=t+dt
     enddo
      aflow = aflow*pi/180
      abat = abat*pi/180
     alfa1 = alfa1*pi/180
      alfa2 = alfa2*pi/180
     beta1 = beta1*pi/180
      beta2 = beta2*pi/180

     !calculation of intermediary section angles

     do i = 2,6,2
       csec(i).th1 =(csec(i+1).th1+csec(i-1).th1)/2
        csec(i).th2 =(csec(i+1).th2+csec(i-1).th2)/2
       csec(i).t = (csec(i+1).t+csec(i-1).t)/2
       if (i==2) then
         k = 1
         do j=1,3
           x(j) = csec(k).t
               y(j) = csec(k).leng
```

```
      k = k+2
    enddo
  else
     k=i-3
    do j=1,3
      x(j) = csec(k).t
          !t = t+tr
          y(j) = csec(k).leng
      k = k+2
    enddo
   endif
  csec(i).leng = intquad(x,y,csec(i).t) !quadratic interpolation
enddo

 !generation of transversal sections
 do k =1,7
   !xm = csec(k).cpos
   !ym = csec(k).camber
  !call solve_foil(xm,ym,a,b,c)
  ds = 1.0/10
   s = 0
   csec(k).p1 = 0
  call set_sec(csec(k))
   do i = 2,11
     s=s+ds
     csec(k).p1(2,i)= csec(k).p1(2,i)*csec(k).leng
        !(a*s**3+b*s**2+c*s)*csec(k).leng
     csec(k).p1(1,i)= csec(k).p1(1,i)*csec(k).leng
        !s*csec(k).leng
   enddo
 enddo
 !foot section height
 zf = tan(beta1)
b  = tan(alfa1)
a  = zf-b
s = 0
csec(1).p1(3,1) = 0
do i = 2,11
  s = s+ds
  csec(1).p1(3,i) = (a*s**2+b*s)*csec(1).leng
enddo

 !top section height
 zf = tan(beta2)
b  = tan(alfa2)
a  = zf-b
s = 0
csec(7).p1(3,1) = height
do i = 2,11
  s = s+ds
  csec(7).p1(3,i) = height+(a*s**2+b*s)*csec(1).leng
enddo

!intermediate sections height
dt = 1.0/6
t=0
do i = 2,6
  t = t+dt
  do j = 1,11
     !df = csec(4).p1(3,j)+csec(1).p1(3,j)
     csec(i).p1(3,j) = t*height   !df
  enddo
enddo


 !Generation of te section
 xm = tesec.cpos
 ym = tesec.camber
 call solve_foil(xm,ym,a,b,c)
 t = 0
 !tesec.p1 = 0
 do i = 1,7
```

```
          tesec.p1(2,i)= (a*t**3+b*t**2+c*t)*height
          tesec.p1(3,i)= csec(i).p1(3,10)
          tesec.p1(1,i)= csec(i).p1(1,10)
         t=t+dt
        enddo


        !rotation of te section (just y coordinate)
        a1 = aflow-abat
       xp = csec(4).p1(1,10)
       yp = csec(4).p1(2,10)
        p(1) = xp*cos(a1)-yp*sin(a1)
       p(2) = xp*sin(a1)+Yp*cos(a1)
       p(3) = csec(4).p1(3,10)
       a2 = atan(p(2)/p(3))
        do i = 1,7
           zp = tesec.p1(3,i)
           yp = tesec.p1(2,i)
           !tesec.p1(3,i) = zp*cos(a1)-yp*sin(a1)
           tesec.p1(2,i) = zp*sin(a2)+Yp*cos(a2)
        enddo

        !translating te section
        yt = trv*length
        do i = 1,7
           tesec.p1(2,i) = tesec.p1(2,i) +yt
        enddo

        !Calculating central and top sections twist angles
        do k = 1,7
          csec(k).asec = atan(tesec.p1(2,k)/
       &                (tesec.p1(1,k)-csec(k).p1(1,1)) )
        enddo

        ! rotating sections
        do k = 1,7
          a1 = csec(k).asec
          xo = csec(k).p1(1,1)
          yo = 0
          do i = 2,11
             xp = csec(k).p1(1,i)
             yp = csec(k).p1(2,i)
             csec(k).p1(1,i) =xo+ xp*cos(a1)-yp*sin(a1)
             csec(k).p1(2,i) =yo+ xp*sin(a1)+Yp*cos(a1)
          enddo
        enddo
        !writting msh file
       M = 11  !chordwise
       N = 7   !spanwise
        open(2,file = 'c:\codigos\mshuns\sail.msh')
         do i = 1,12
           write(2,10)
         enddo
         write(2,20) M,N
         t = 0
         do i = 1,N
           s = 0
           do j = 1,M
             write(2,30) csec(i).p1(1,j),csec(i).p1(2,j),csec(i).p1(3,j),
       &                  s,t
             s = s+ds
           enddo
           t = t+dt
          enddo
       close(2)

10      format('%',1x)
20      format(1x,i4,1x,i4)
30      format(1x,f8.5,1x,f8.5,1x,f8.5,1x,f8.5,1x,f8.5)
        end program

!**********************************************************************
```

```
 subroutine set_sec(sec)
include 'section.inc'
!This program generates the section using the Jackson Polynomial
!Ref: P.S. Jackson, "A Simple Model for 2D Sails
! AIAA Technical notes 1983
  ! Author: Augusto Veiga

type(section):: sec
real:: A,B, pi
real:: delta, a1,b1,c1

pi = 4. * atan(1.)
sec.th1 = sec.th1*pi/180
sec.th2 = sec.th2*pi/180
A = sec.th1+sec.th2
B = sec.th1-sec.th2

!seeking maximum camber position
a1 = -0.75*B*2
b1 = -0.5*A
c1 = 0.25*B

delta = b1**2-4*a1*c1
if (delta>=0) then
  r = (b1-sqrt(delta))/(2*a1)
  sec.cpos = (1+r)/2.
  tm = 0.25*(1-r**2)*(A+B*r)
  sec.camber = tm/2.
endif

!Generating sections
 ds = 2.0/10
dx = 1.0/10
s = -1
x = 0
do i=1,11
  t = 0.25*(1-s**2)*(A+B*s)
  sec.p1(2,i) = t/2
  sec.p1(1,i) = x
  s = s+ds
  x = x+dx
enddo
 return
end subroutine

!*******************************************************
      real function intquad(x,y,x1)
      real,dimension(3):: x,y
      real:: x1,sum

      sum = 0
       sum = sum+((x1-x(2))*(x1-x(3)))/((x(1)-x(2))*(x(1)-x(3)))*y(1)
      sum = sum+((x1-x(1))*(x1-x(3)))/((x(2)-x(1))*(x(2)-x(3)))*y(2)
      sum = sum+((x1-x(1))*(x1-x(2)))/((x(3)-x(1))*(x(3)-x(2)))*y(3)

      intquad = sum
      end function
       subroutine solve_foil(xm,ym,a,b,c)
      !makes the foil using a 3rd order polynomial
      real:: xm,ym,a,b,c
      real:: vr(3,3), v(3)

      do i=1,3
        k = 4
        if (i==2) then
          v(i) = ym/xm
        else
          v(i) = 0
        endif
        do j=1,3
          k = k-j
          if (i==1) then
```

```
          vr(i,j) = k*xm**(k-1)
        else if (i==2) then
          vr(i,j) = xm**(k-1)
        else
          vr(i,j) = 1.0
        endif
      enddo
     enddo
     !solve system using Gauss elimination
     call gauss(3,3,Vr,v,1)
     a = v(1)
    b = v(2)
    c = v(3)

    return
    end subroutine


      SUBROUTINE GAUSS(NSIZ,NN,Z,R,NRHS)
C     *****************************************************
C     *                                                   *
C     *   Solves general NxN system in NN unknowns        *
C     *    with arbitrary number (NRHS) of righthand sides. *
C     *   Assumes system is invertible...                 *
C     *    ...if it isn't, a divide by zero will result.  *
C     *                                                   *
C     *   Z is the coefficient matrix...                  *
C     *     ...destroyed during solution process.         *
C     *   R is the righthand side(s)...                   *
C     *     ...replaced by the solution vector(s).        *
C     *                                                   *
C     *                                                   *
C     *****************************************************
C
      DIMENSION Z(NSIZ,NSIZ), R(NSIZ,NRHS)
C
      DO 1 NP=1, NN-1
        NP1 = NP+1
C
C------ find max pivot index NX
        NX = NP
        DO 11 N=NP1, NN
          IF(ABS(Z(N,NP))-ABS(Z(NX,NP))) 11,11,111
  111      NX = N
   11   CONTINUE
C
        PIVOT = 1.0/Z(NX,NP)
C
C------ switch pivots
        Z(NX,NP) = Z(NP,NP)
C
C------ switch rows & normalize pivot row
        DO 12 L=NP1, NN
          TEMP = Z(NX,L)*PIVOT
          Z(NX,L) = Z(NP,L)
          Z(NP,L) = TEMP
   12   CONTINUE
C
        DO 13 L=1, NRHS
          TEMP = R(NX,L)*PIVOT
          R(NX,L) = R(NP,L)
          R(NP,L) = TEMP
   13   CONTINUE
C
C------ forward eliminate everything
        DO 15 K=NP1, NN
          ZTMP = Z(K,NP)
C
C          IF(ZTMP.EQ.0.0) GO TO 15
C
          DO 151 L=NP1, NN
            Z(K,L) = Z(K,L) - ZTMP*Z(NP,L)
```

```
  151     CONTINUE
          DO 152 L=1, NRHS
            R(K,L) = R(K,L) - ZTMP*R(NP,L)
  152     CONTINUE
   15   CONTINUE
C
    1 CONTINUE
C
C---- solve for last row
      DO 2 L=1, NRHS
        R(NN,L) = R(NN,L)/Z(NN,NN)
    2 CONTINUE
C
C---- back substitute everything
      DO 3 NP=NN-1, 1, -1
        NP1 = NP+1
        DO 31 L=1, NRHS
          DO 310 K=NP1, NN
            R(NP,L) = R(NP,L) - Z(NP,K)*R(K,L)
  310     CONTINUE
   31   CONTINUE
    3 CONTINUE
C
      RETURN
      END ! GAUSS
```