รายการอ้างอิง

1. ปราโมทย์ เดชะอำไพ. ระเบียบวิธีไฟไนต์เอลิเมนต์ในงานวิศวกรรม. พิมพ์ครั้งที่ 2. จุฬาลงกรณ์มหาวิทยาลัย, 2542.
2. White, F. M. Viscous fluid flow . 2nd ed. New York: McGraw-Hill, 1991.
3. จิตติน ตรีพุทธรัตน์. การศึกษาการไหลผ่านวัตถุด้วยวิธีการไฟไนต์เอลิเมนต์. วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมเครื่องกล บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย, 2539.
4. Potter M. C., Wiggert, D. C. Mechanics of fluids . Prentice Hall, 1991.
5. Fox, R. W., and McDonald, A. T. Introduction to fluid mechanics. John Wiley&Son, 1994.
6. Dechaumphai, P.; Triputtarat, J.; and Sikkhabandit, S. A finite element method for viscous incompressible flow analysis. Thammasat International Journal of Science and Technology Vol. 3. No. 2 (1998): 60-68.
7. Reddy, J.N. Finite element method. 2nd ed. New York: McGraw-Hill, 1993.
8. Zienkiewicz, O. C. ,and Taylor, R. L. The finite element method. 4th ed. New York: McGraw-Hill, 1991.
9. Hestenes, Magnus R., and Stiefel, E. Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards Vol. 49. No. 6 (December, 1952): 409-436.
10. Shewchuk, J. R. An introduction to the conjugate gradient method without the agonizing pain: School of Computer Science, Carnegie Mellon University Pittsburgh, 1 — ed., 1994.
11. Press W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. Numerical Recipes in Fortran 77. Vol. 1. 2nd ed., The Art of Science Computing, Cambridge University Press, 1996.
12. Kelley, C. T., Iterative methods for linear and nonlinear equations: Society for Industrial and Applied Mathematics, Vol. 16, Philadelphia, P. A., 1995.
13. Barrett, R., and other.Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Society for Industrial and Applied Mathematics. Philadelphia. PA., 1994.

14. Golub, G. H. Van Loan, C. F. Metrix Computations. Baltimore, Maryland : Johns Hopkins University Press, 1983.

15. Dechaumphai, P. "Adaptive Finite Element Technique for Heat Transfer Problems." Energy, Heat and Mass Transfer Vol. 17 (1995): 87-94.

16. Dechaumphai, P., and Janphaisaeng, P. Adaptive finite element technique for high-speed compressible flows. Thammasat International Journal of Science and Technology Vol. 3. No. 1, 1998.

17. Gartling D. K. A test problem for outflow boundary conditions – flow over a backward facing step. International of Numerical Methods in Fluids Vol. 11 (1990) : 953-967.

18. Lewis, R. W., Huang, H. C., Usmani, A. S., and Cross, J. T. Finite element analysis of heat transfer and flow problems using adaptive remeshing including application to solidification problems. International Journal for Numerical Methods in Engineering Vol. 32 (1991) : 767-781.

19. Schlichting, H., Boundary layer theory. New York : McGraw-Hill, 1978.

20. Wirz, H. J., and Smolderen, J. J. Numerical methods in fluid dynamics. New York : McGraw-Hill, 1978.

ภาคผนวก

ภาคผนวก ก

รายละเอียดโปรแกรม NV

รายละเอียดโปรแกรม NV จะมีรายละเอียดเริ่มจากโปรแกรมหลัก และตามด้วยโปรแกรมย่อยต่าง ๆ ดังนี้

```
C     PROGRAM NV
C
C     A FINITE ELEMENT COMPUTER PROGRAM FOR SOLVING NAVIER-STOKES
C     EQUATION FOR TWO-DIMENTIONAL VISCOUS INCOMPRESSIBLE FLOWS.
C
C              USE PRECONDITIONED CONJUGATE GRADIENT
C           TO SOLVE LINEAR SYSTEM OF NORMAL EQUATIONS
C
C     THE VALUES DECLARED IN THE PARAMETER STATEMENT BELOW SHOULD
C     BE ADJUSTED ACCORDING TO THE SIZE OF THE PROBLEMS AND TYPES
C     OF COMPUTERS:
C         MXPOIV = MAXIMUM NUMBER OF VELOCITY NODES IN THE MODEL
C         MXPOIP = MAXIMUM NUMBER OF PRESSURE NODES IN THE MODEL
C         MXELE  = MAXIMUM NUMBER OF ELEMENTS IN THE MODEL
C         ITMAX  = MAXIMUM ITERATION SETTING (INTERNAL LOOP)
C         TOL1   = SPECIFIED TOLERANCE (INTERNAL LOOP)
C
      PARAMETER (MXPOIV=25,MXPOIP=9, MXELE=8, MXFREE=1)
      PARAMETER (NMAX=1000000, ITMAX=100000)
      PARAMETER (MXLINK=10)
      PARAMETER (MXNEQ=2*MXPOIV+MXPOIP)
C
      IMPLICIT  REAL*4 (A-H,O-Z)
C
       PARAMETER (TOL1=1.E-06)
C
C        TOL1  = SPECIFIED TOLERANCE OF EACH CONJUGATE GRADIENT
C                ITERATION
C
      DIMENSION  COORD(MXPOIV,2), TEXT(20)
      DIMENSION  UVEL(MXPOIV), VVEL(MXPOIV), PRES(MXPOIV)
      DIMENSION  B(MXNEQ), BN(MXNEQ)
      DIMENSION  AKELE(15,15,MXELE), RELE(15,MXELE)
      DIMENSION  AAS(MXNEQ)
      DIMENSION  ICOUNT(MXPOIV), IASSEM(MXPOIV,MXLINK)
      DIMENSION  SOL(MXNEQ), DSOL(MXNEQ)
      DIMENSION  P(MXNEQ), R(MXNEQ), Z(MXNEQ), Q(MXNEQ)
      DIMENSION  SA(NMAX), SB(NMAX), SC(NMAX)
      DIMENSION  IJA(NMAX), IJB(NMAX), IJC(NMAX)
      DIMENSION  INTMAT(MXELE,6), INTMATF(MXFREE,4)
      DIMENSION  IBCU(MXPOIV), IBCV(MXPOIV), IBCP(MXPOIV)
      CHARACTER*20  NAME1, NAME2, NAME3, NAME4, NAME5
      CHARACTER*20  NAME6, NAME7, NAME8
C
   10 WRITE(6,20)
   20 FORMAT(/,' PLEASE ENTER THE INPUT FILE NAME:', /)
      READ(5,'(A)',ERR=10) NAME1
      OPEN(UNIT=7, FILE=NAME1, STATUS='OLD', ERR=10)
      OPEN(UNIT=9, FILE='CHECK.OUT', STATUS='NEW')
C
C     READ TITLE OF COMPUTATION:
C
      READ(7,*)  NLINES
      DO 100  ILINE=1,NLINES
      READ(7,1) TEXT
```

```
    1 FORMAT(20A4)
  100 CONTINUE
C
C     READ INPUT DATA:
C
      READ(7,1)  TEXT
      WRITE(9,104)
  104 FORMAT('   NPOIV   NPOIP   NELEM   NFREE   NITER    TOL')
      READ(7,*)    NPOIV, NPOIP, NELEM, NFREE, NITER, TOL
      WRITE(9,105) NPOIV, NPOIP, NELEM, NFREE, NITER, TOL
  105 FORMAT(5I8, F8.2)
      IF(NPOIV.GT.MXPOIV)  WRITE(6,110) NPOIV
  110 FORMAT(/,' PLEASE INCREASE THE PARAMETER MXPOIV TO',I5)
      IF(NPOIV.GT.MXPOIV)  STOP
      IF(NPOIP.GT.MXPOIP)  WRITE(6,120) NPOIP
  120 FORMAT(/,' PLEASE INCREASE THE PARAMETER MXPOIP TO',I5)
      IF(NPOIP.GT.MXPOIP)  STOP
      IF(NELEM.GT.MXELE)  WRITE(6,130) NELEM
  130 FORMAT(/,' PLEASE INCREASE THE PARAMETER MXELE TO',I5)
      IF(NELEM.GT.MXELE)  STOP
      IF(NFREE.GT.MXFREE)  WRITE(6,140) NFREE
  140 FORMAT(/,' PLEASE INCREASE THE PARAMETER MXFREE TO',I5)
      IF(NFREE.GT.MXFREE)  STOP
C
C     READ FLUID PROPERTIES:
C
      READ(7,1)  TEXT
      WRITE(9,134)
  134 FORMAT('     DENSITY   VISCOSITY')
      READ(7,*)  DEN, VIS
      WRITE(9,135)  DEN, VIS
  135 FORMAT(2E12.4)
C
C     READ NODAL COORDINATES, BOUNDARY CONDITIONS, THEIR VALUES:
C     REQUIREMENT:  MAIN NODES MUST BE NUMBERED FIRST
C
      READ(7,1)  TEXT
      DO 150  IP=1,NPOIV
      READ(7,*)    I, IBCU(I), IBCV(I), IBCP(I),
     *             (COORD(I,K), K=1,2), UVEL(I), VVEL(I), PRES(I)
      IF(I.NE.IP)  WRITE(6,155) IP
  155 FORMAT(/, ' NODE NO.', I5, ' IN DATA FILE IS MISSING')
      IF(I.NE.IP) STOP
  150 CONTINUE
C
C     READ ELEMENT NODAL CONNECTIONS:
C
      READ(7,1)  TEXT
      DO 160  IE=1,NELEM
      READ(7,*)    I, (INTMAT(I,J), J=1,6)
  162 FORMAT(7I8)
      IF(I.NE.IE)  WRITE(6,165) IE
  165 FORMAT(/, ' ELEMENT NO.', I5, ' IN DATA FILE IS MISSING')
      IF(I.NE.IE) STOP
  160 CONTINUE
C
C     READ FREE BOUNDARY (FLOW EXIT) INFORMATION:
C
      READ(7,1)  TEXT
      WRITE(9,168)  NFREE
  168 FORMAT(' OUTFLOW INFORMATION (ELE NO., 3 NODE NO.): [',
     *       I4, ']')
      DO 170  IB=1,NFREE
      READ(7,*)    (INTMATF(IB,J), J=1,4)
  170 CONTINUE
C
      WRITE(6,200)  NPOIV, NPOIP, NELEM, NFREE, NITER, TOL
  200 FORMAT('    THE FINITE ELEMENT MODEL CONSISTS OF:', /,
     *       '         NUMBER OF VELOCITY NODES       =', I6, /,
     *       '         NUMBER OF PRESSURE NODES       =', I6, /,
     *       '         NUMBER OF ELEMENTS             =', I6, /,
     *       '         NUMBER OF OUTFLOW BOUNDARY     =', I6, /,
     *       '    WITH NUMBER OF ITERATIONS REQUIRED  =', I6, /,
     *       '    OR SPECIFIED STOPPING TOLERANCE     =', F6.2 )
C
      DO 400  I=1,NPOIV
      SOL(I      ) = UVEL(I)
```

```
      SOL(I+NPOIV) = VVEL(I)
  400 CONTINUE
      DO 410  I=1,NPOIP
      SOL(I+NPOIV+NPOIV) = PRES(I)
  410 CONTINUE
C
      NEQ = 2*NPOIV + NPOIP
C
      CALL COUNT(INTMAT, ICOUNT, IASSEM, MXPOIV,  MXELE, MXLINK)
C
C     ENTER ITERATION LOOP:
C
      DO 500  ITER=1,NITER
C
C     RESET THE SYSTEM EQUATIONS
C
      DO 510 I=1,15
      DO 510 J=1,15
      DO 510 K=1,MXELE
         AKELE(I,J,K) = 0.
  510 CONTINUE
      DO 520 I=1,15
      DO 520 J=1,MXELE
         RELE(I,J) = 0.
  520 CONTINUE
      DO 521 I=1,NMAX
      SA(I) =0.
      SB(I) =0.
      SC(I) =0.
      IJA(I)=0.
      IJB(I)=0.
      IJC(I)=0.
  521 CONTINUE
C
      WRITE(6,530)  ITER
  530 FORMAT(/, 3X, ' * PERFORMING COMPUTATION AT ITERATION NUMBER',
     *       I3, ':')
C
C     ESTABLISH ELEMENT MATRICES AND ASSEMBLE ELEMENT EQUATIONS
C
      WRITE(6,540)
  540 FORMAT(8X, ' ESTABLISHING ELEMENT MATRICES AND',
     *            ' ASSEMBLING ELEMENT EQS.'     )
C
      CALL TRI( MXPOIV, MXELE, MXNEQ,SOL, INTMAT, COORD,
     *          AKELE, RELE,IBCU, IBCV, IBCP, DEN, VIS )
C
C     ASSEMBLE THE SYSTEM STIFFNESS MATRIX
C     AND CHANGE TO SPARSE STORAGE FORMAT
*
C
C     FIND STIFFNESS MATRIX
C
*     FIND TRANSPOSE OF STIFFNESS MATRIX
      ijb(1)=neq+2
      K=NEQ+1
      DO 542 IJ=1,MXNEQ
      CALL ASSEMA(INTMAT, AKELE, AAS, 1, MXPOIV, MXNEQ, MXELE,
     *           ICOUNT, IASSEM, IJ,MXLINK)
C
C     CREAT ROW SPARSE STORAGE OF A TRANSPOSE
C
        CALL SPRSIN(AAS,NEQ,NMAX,K,IJ,SB,IJB)
C
  542 CONTINUE
C
C     ASSEMBLE THE RESIDUALS MATRICES TO FORM SYSTEM RESIDUALS:
C
      WRITE(6,545)
  545 FORMAT(8X, ' ASSEMBLING THE SYSTEM RESIDUALS VECTOR')
C
      CALL ASSEMR( MXPOIV, MXELE, MXNEQ,
     *             INTMAT,  RELE,     B )
C
C     SOLVE A SET OF SIMULTANEOUS EQUATIONS FOR NODAL INCREMENTS:
C
      WRITE(6,560)
```

```
  560 FORMAT(8X, ' SOLVING SET OF SIMULTANEOUS EQS. FOR',
     *          ' NODAL INCREMENTS'                    )
      DO 580 I=1,NEQ
       BN(I)=0.0
  580 CONTINUE
      DO 600 I=1,NEQ
      P(I)=0.0
      R(I)=0.0
      Z(I)=0.0
      Q(I)=0.0
  600 CONTINUE
C
C     CHANGE A*X=B TO AT*A*X=AT*B (NORMAL EQUATIONS)
C
C     FIND A TRANSPOSE * A
C
      write(*,*) '    FIND A TRANSPOSE * A '
      CALL SPRSTM(sb,ijb,sb,ijb,nmax,sc,ijc)
C
C     FIND A TRANSPOSE * B
C
      WRITE(*,*) '    FIND A TRANSPOSE * B '
      CALL SPRSAX(sb,ijb,b,bn,neq,nmax)
C
C     APPLY CONJUGATE GRADIENT METHOD TO NORMAL EQUATION
C
      WRITE(6,570)  MXNEQ
  570 FORMAT(8X, ' ( TOTAL OF', I5,' EQUATIONS TO BE SOLVED )')
      WRITE(*,*) '   APPLYING CONJUGATE GRADIENT METHOD '
      CALL PCGNR(NEQ,SC,IJC,BN,NEQ,P,R,Z,Q,DSOL,ITMAX,NMAX,TOL1)
C
C     CHECK FOR CONVERGENCE:
C
      UP   = 0.
      DOWN = 0.
      DO 700  I=1,NEQ
      ERROR = DSOL(I)
      UP    = UP + ABS(ERROR)
      VALUE = SOL(I)
      DOWN  = DOWN + ABS(VALUE)
  700 CONTINUE
      RATIO = UP*100./DOWN
      WRITE(6,585)  RATIO
  585 FORMAT(6X, 'CURRENT SOLUTION HAS GLOBAL ERROR OF',
     *       F8.2, ' %'                                 )
      WRITE(9,595)  ITER, RATIO
  595 FORMAT(6X, 'ITERATION NO.', I5, ' HAS GLOBAL ERROR OF',
     *       F8.2, ' %'                                    )
      IF(RATIO.GT.TOL)  GO TO 710
C
C     SOLUTION CONVERGED WITHIN THE SPECIFIED TOLERANCE
C
      WRITE(6,605)
  605 FORMAT(/, 3X, ' *** SOLUTION CONVERGED WITHIN SPECIFIED',
     *       ' TOLERANCE ***', //                          )
      GO TO 800
  710 CONTINUE
C
C     UPDATE NODAL SOLUTIONS:
C
      DO 720  I=1,NEQ
      SOL(I) = SOL(I) + DSOL(I)
  720 CONTINUE
  500 CONTINUE
C
C     SOLUTION NOT CONVERGED WITHIN THE SPECIFIED TOLERANCE
C
      WRITE(6,615)
  615 FORMAT(/, 3X, ' ??? SOLUTION NOT CONVERGED WITHIN',
     *       ' SPECIFIED TOLERANCE ???', //             )
C
  800 CONTINUE
C
C     PRINT OUT SOLUTIONS OF NODAL VELOCITIES AND PRESSURES:
C
  810 WRITE(6,625)
  625 FORMAT(' PLEASE ENTER FILE NAME FOR VELOCITY & PRESSURE',
```

```
     *       ' SOLUTIONS:', /                                        )
      READ(5, '(A)', ERR=810)  NAME2
      OPEN(UNIT=8, FILE=NAME2, STATUS='NEW', ERR=810)
      WRITE(8,635)  NPOIV
  635 FORMAT('  NODAL VELOCITY AND PRESSURE SOLUTIONS [', I5,']:',
     *       //, 2X, 'NODE', 6X, 'U-VELOCITY', 6X, 'V-VELOCITY',
     *       8X, 'PRESSURE', /                                       )
C
C     ROUND-OFF SOLUTION VALUES FOR NEAT OUTPUT:
C
      ROFF = 1.E-6
      DO 820  IEQ=1,NEQ
      VALUE = SOL(IEQ)
      IF(ABS(VALUE).LT.ROFF)  SOL(IEQ) = 0.
  820 CONTINUE
C
      DO 830  IP=1,NPOIP
      IEQU = IP
      IEQV = NPOIV + IP
      IEQP = 2*NPOIV + IP
      WRITE(8,645)  IP, SOL(IEQU), SOL(IEQV), SOL(IEQP)
  645 FORMAT(I6, 3E16.6)
  830 CONTINUE
      DO 840  IP=NPOIP+1,NPOIV
      IEQU = IP
      IEQV = NPOIV + IP
      WRITE(8,655)  IP, SOL(IEQU), SOL(IEQV)
  655 FORMAT(I6, 2E16.6)
  840 CONTINUE
C
C     CREATE DATA FILE FOR GRAPHIC DISPLAY (FEPLOT):
C
  850 WRITE(6,665)
  665 FORMAT(' PLEASE ENTER FILE NAME FOR U-V-P DISPLAY:', /)
      READ(5,'(A)', ERR=850)  NAME3
      OPEN(UNIT=10, FILE=NAME3, STATUS='NEW', ERR=850)
      NVAR = 3
      WRITE(10,675)  NPOIP, NELEM, NVAR
  675 FORMAT('   NPOIP   NELEM    NVAR', /, 3I8)
      WRITE(10,685)  NPOIP
  685 FORMAT(' NODAL COORDINATES & U-V-P SOLUTIONS [', I5, ']:')
      DO 860  I=1,NPOIP
      IEQU = I
      IEQV = NPOIV + I
      IEQP = 2*NPOIV + I
      WRITE(10,695)  I, (COORD(I,J), J=1,2), SOL(IEQU), SOL(IEQV),
     *               SOL(IEQP)
  695 FORMAT(I8, 5E13.5)
  860 CONTINUE
      WRITE(10,705)  NELEM
  705 FORMAT(' ELEMENT NODAL CONNECTIONS [', I5, ']:')
      DO 870  IE=1,NELEM
      WRITE(10,880)  IE, (INTMAT(IE,J), J=1,3)
  880 FORMAT(4I8)
  870 CONTINUE
C
  900 WRITE(6,910)
  910 FORMAT(' PLEASE ENTER FILE NAME FOR U-V DISPLAY:', /)
      READ(5,'(A)', ERR=900)  NAME4
      OPEN(UNIT=11, FILE=NAME4, STATUS='NEW', ERR=900)
      NVAR = 2
      NELEM4 = 4*NELEM
      WRITE(11,920)  NPOIV, NELEM4, NVAR
  920 FORMAT('   NPOIV   NELEM    NVAR', /, 3I8)
      WRITE(11,930)  NPOIV
  930 FORMAT(' NODAL COORDINATES & U-V SOLUTIONS [', I5, ']:')
      DO 940  I=1,NPOIV
      IEQU = I
      IEQV = NPOIV + I
      WRITE(11,950)  I, (COORD(I,J), J=1,2), SOL(IEQU), SOL(IEQV)
  950 FORMAT(I8, 4E13.5)
  940 CONTINUE
      WRITE(11,960)  NELEM4
  960 FORMAT(' ELEMENT NODAL CONNECTIONS [', I5, ']:')
      ICE = 1
      DO 970  IE=1,NELEM
      II = INTMAT(IE,1)
```

```
      JJ = INTMAT(IE,2)
      KK = INTMAT(IE,3)
      LL = INTMAT(IE,4)
      MM = INTMAT(IE,5)
      NN = INTMAT(IE,6)
      WRITE(11,980)  ICE, II, NN, MM
      ICE = ICE + 1
      WRITE(11,980)  ICE, JJ, LL, NN
      ICE = ICE + 1
      WRITE(11,980)  ICE, KK, MM, LL
      ICE = ICE + 1
      WRITE(11,980)  ICE, LL, MM, NN
      ICE = ICE + 1
  980 FORMAT(4I8)
  970 CONTINUE
C
C     CREATE VELOCITY DATA FOR NASTRAN:
C
 1000 WRITE(6,1005)
 1005 FORMAT(' PLEASE ENTER FILE NAME VELOCITY DATA FOR NASTRAN:', /)
      READ(5,'(A)', ERR = 1000) NAME4
      OPEN(UNIT = 12,FILE=NAME4, STATUS='NEW',ERR=1000)
      WRITE(12,1015)
 1015 FORMAT('1MSC/NASTRAN PAGE')
      WRITE(12,1025)
 1025 FORMAT('0')
      WRITE(12,1035)
 1035 FORMAT(' D I S P L A C E M E N T')
      WRITE(12,1045)
 1045 FORMAT('      POINT ID. TYPE')
      DO 1060 I = 1, MXPOIV
      IEQU = I
      IEQV = MXPOIV+I
      WRITE(12,1055) I, SOL(IEQU), SOL(IEQV)
 1055 FORMAT(I12,4X,'G',2(1X,E12.6),10X,'0.0')
 1060 CONTINUE
      WRITE(12,1065)
 1065 FORMAT('1')
C
C     CREATE DATA FOR NASTRAN GRAPHIC FOR INPUT
C
 1070 WRITE (6,1075)
 1075 FORMAT (' PLEASE ENTER FILE NAME INPUT DATA FOR NASTRAN:' /)
      READ (5,'(A)', ERR = 1070) NAME5
      OPEN (UNIT=13,FILE=NAME5,STATUS='NEW',ERR=1070)
      DO 1080 I = 1 , NPOIV
      Z1 = 0.0
      IZ = 0
      WRITE(13,1085) I, IZ, (COORD(I,J),J=1,2), Z1, IZ
      WRITE(6,1085) I, IZ, (COORD(I,J),J=1,2), Z1, IZ
 1085 FORMAT('GRID',9X,I4,7X,I1,2X,F7.4,1X,F7.4,6X,F2.0,7X,I1)
 1080 CONTINUE
      DO 1090 I = 1, NELEM
      IO = 1
      WRITE (13,1095) I, IO,INTMAT(I,1),INTMAT(I,3),INTMAT(I,2),
     *                       INTMAT(I,5),INTMAT(I,4),INTMAT(I,6)
      WRITE (6,1095) I, IO,(INTMAT(I,J), J=1,6)
 1095 FORMAT('CTRIA6', 7X, I4, 7X, I1, 6I8)
 1090 CONTINUE
C
C     CREATE DATA NASTRAN GRAPHIC FOR OUTPUT:
C
 1100 WRITE (6,1105)
 1105 FORMAT(' PLEASE ENTER FILE NAME OUTPUT DATA FOR NASTRAN:', /)
      READ(5,'(A)', ERR = 1100) NAME6
      OPEN(UNIT = 14,FILE=NAME6, STATUS='NEW',ERR=1100)
      WRITE(14,1115)
 1115 FORMAT('1MSC/NASTRAN PAGE')
      WRITE(14,1125)
 1125 FORMAT('0')
      WRITE(14,1135)
 1135 FORMAT(' D I S P L A C E M E N T')
      WRITE(14,1145)
 1145 FORMAT('      POINT ID. TYPE')
      DO 1110 I = 1, NPOIP
      IEQU = I
      IEQV = NPOIV+I
```

```
      IEQP = 2*NPOIV+I
      WRITE(14,1155) I, SOL(IEQU), SOL(IEQV),  SOL(IEQP)
 1155 FORMAT(I12,4X,'G',1X,E12.5,1X,E12.5,10X,'0.0',1X,E12.5,9X,'0.0'
     *,9X,'0.0')
 1110 CONTINUE
      WRITE(14,1175)
 1175 FORMAT('1')
C
C     CREATE DATA FOR VELOCITY REMESH [SPACE.FOR] :
C
 1130 WRITE (6,1185)
 1185 FORMAT(' PLEASE ENTER FILE NAME FOR VELOCITY REMESHING FOR SPACE:'
     1       , /)
      READ(5,'(A)', ERR = 1130) NAME7
      OPEN(UNIT = 15,FILE=NAME7, STATUS='NEW',ERR=1130)
      DO 1140 I = 1,NPOIV
      SOL(I) = SQRT(SOL(I)*SOL(I)+SOL(I+NPOIV)*SOL(I+NPOIV))
 1140 CONTINUE
      WRITE(15,1195) NPOIP
 1195 FORMAT(I4)
      DO 1150 I = 1,NPOIP
      WRITE(15,2005) I, SOL(I)
 2005 FORMAT(I4,2X,E12.5)
 1150 CONTINUE
C
C     CREATE DATA FOR PRESSURE REMESH [SPACE.FOR] :
C
 1160 WRITE (6,2010)
 2010 FORMAT(' PLEASE ENTER FILE NAME FOR PRESSURE REMESHING FOR SPACE:'
     1       , /)
      READ(5,'(A)', ERR = 1160) NAME8
      OPEN(UNIT = 15,FILE=NAME8, STATUS='NEW',ERR=1160)
      WRITE(15,2015) NPOIP
 2015 FORMAT(I4)
      J=0
      DO 1180 I = 2*NPOIV+1, 2*NPOIV+NPOIP
      J=J+1
      WRITE(15,2025) J, SOL(I)
 2025 FORMAT(I4,2X,E12.5)
 1180 CONTINUE
      STOP
      END
C
C-------------------------------------------------------------------------
C
      SUBROUTINE COUNT(INTMAT, ICOUNT, IASSEM, MXPOIV,  MXELE, MXLINK)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION  INTMAT(MXELE,6), ICOUNT(MXPOIV), IASSEM(MXPOIV,MXLINK)
      DO 10 I=1,MXPOIV
         ICOUNT(I) = 0
   10 CONTINUE
      DO 100 I=1,MXELE
         DO 200 J=1,6
            N = INTMAT(I,J)
            ICOUNT(N) = ICOUNT(N) + 1
            IC = ICOUNT(N)
            IF(IC.GT.MXLINK)  WRITE(*,125) IC
  125       FORMAT(/,' PLEASE INCREASE THE PARAMETER MXLINK TO',I5)
            IF(IC.GT.MXLINK) STOP
            IASSEM(N,IC) = I
  200    CONTINUE
  100 CONTINUE
      RETURN
      END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE APPLYBC( MXPOIV, MXELE, AKELE,  RELE,
     *                    INTMAT,  IBCU,  IBCV,  IBCP,    IE )
C
C     APPLY BOUNDARY CONDITIONS BEFORE SOLVING FOR NODAL INCREMENTS
C     WITH CONDITION CODES OF:
C           0 = FREE TO CHANGE (INCREMENTS COMPUTED)
C           1 = FIXED AS SPECIFIED (INCREMENTS FIXED AS ZERO)
C
      DIMENSION  AKELE(15,15,MXELE), RELE(15,MXELE)
      DIMENSION  INTMAT(MXELE,6)
```

```
      DIMENSION  IBCU(MXPOIV), IBCV(MXPOIV), IBCP(MXPOIV)
C
C     APPLY BOUNDARY CONDITIONS FOR NODAL U-VELOCITIES:
C
      DO 100  IEQ=1,6
         IEQU = INTMAT(IE,IEQ)
         IF(IBCU(IEQU).EQ.0)  GO TO 100
C
         DO 110  IR=1,15
            IF(IR.EQ.IEQ)  GO TO 110
            AKELE(IR,IEQ,IE) = 0.
  110    CONTINUE
C
         DO 120  IC=1,15
            AKELE(IEQ,IC,IE) = 0.
  120    CONTINUE
         AKELE(IEQ,IEQ,IE) = 1.
         RELE(IEQ,IE) = 0.
C
  100 CONTINUE
C
C     APPLY BOUNDARY CONDITIONS FOR NODAL V-VELOCITIES:
C
      DO 200  IEQ=1,6
         IEQV = INTMAT(IE,IEQ)
         IF(IBCV(IEQV).EQ.0)  GO TO 200
C
         DO 210  IR=1,15
            IF(IR.EQ.IEQ+6)  GO TO 210
            AKELE(IR,IEQ+6,IE) = 0.
  210    CONTINUE
C
         DO 220  IC=1,15
            AKELE(IEQ+6,IC,IE) = 0.
  220    CONTINUE
         AKELE(IEQ+6,IEQ+6,IE) = 1.
         RELE(IEQ+6,IE) = 0.
C
  200 CONTINUE
C
C     APPLY BOUNDARY CONDITIONS FOR NODAL PRESSURES:
C
      DO 300  IEQ=1,3
         IEQP = INTMAT(IE,IEQ)
         IF(IBCP(IEQP).EQ.0)  GO TO 300
C
         DO 310  IR=1,15
            IF(IR.EQ.IEQ+12)  GO TO 310
            AKELE(IR,IEQ+12,IE) = 0.
  310    CONTINUE
C
         DO 320  IC=1,15
            AKELE(IEQ+12,IC,IE) = 0.
  320    CONTINUE
         AKELE(IEQ+12,IEQ+12,IE) = 1.
         RELE(IEQ+12,IE) = 0.
C
  300 CONTINUE
C
      RETURN
      END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE ASSEMA(INTMAT, A, B, ITYPE, MXPOIV, MXNEQ, MXELE,
     *                  ICOUNT, IASSEM, N, MXLINK)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION  A(15,15,MXELE), B(MXNEQ)
      DIMENSION  INTMAT(MXELE,6), ICOUNT(MXPOIV), IASSEM(MXPOIV,MXLINK)
C
C     RESET STIFFNESS MATRICES
C
      DO 10 I=1,MXNEQ
         B(I) = 0.
   10 CONTINUE
C
      IF (N.LE.MXPOIV) THEN
```

```
         N1 = N
         NN = 0
      ELSE
         IF ((N.GT.MXPOIV).AND.(N.LE.2*MXPOIV)) THEN
            N1 = N - MXPOIV
            NN = 1
         ELSE
            N1 = N - 2*MXPOIV
            NN = 2
         ENDIF
      ENDIF
      IC = ICOUNT(N1)
C
C     ASSEMBLE STIFFNESS MATRICES
C
      IF (ITYPE.EQ.0) THEN
C
C     ITYPE = 0; COMPUTE [A]
C
         DO 100 IE=1,IC
            IA = IASSEM(N1,IE)
            DO 30 I=1,6
               II = INTMAT(IA,I)
               IF (II.EQ.N1) THEN
C
                  DO 40 J=1,6
                     JJ = INTMAT(IA,J)
                     K  = J + 6
                     KK = MXPOIV + JJ
                     B(JJ) = B(JJ) + A(I+6*NN,J,IA)
                     B(KK) = B(KK) + A(I+6*NN,K,IA)
   40             CONTINUE
C
                  DO 50 J=1,3
                     JJ = INTMAT(IA,J)
                     K  = J + 12
                     KK = 2*MXPOIV + JJ
                     B(KK) = B(KK) + A(I+6*NN,K,IA)
   50             CONTINUE
C
               ENDIF
   30       CONTINUE
  100    CONTINUE
C
      ELSE
C
C     ITYPE = 1; COMPUTE TRANSPOSE OF [A]
C
         DO 200 IE=1,IC
            IA = IASSEM(N1,IE)
            DO 130 I=1,6
               II = INTMAT(IA,I)
               IF (II.EQ.N1) THEN
C
                  DO 140 J=1,6
                     JJ = INTMAT(IA,J)
                     K  = J + 6
                     KK = MXPOIV + JJ
                     B(JJ) = B(JJ) + A(J,I+6*NN,IA)
                     B(KK) = B(KK) + A(K,I+6*NN,IA)
  140             CONTINUE
C
                  DO 150 J=1,3
                     JJ = INTMAT(IA,J)
                     K  = J + 12
                     KK = 2*MXPOIV + JJ
                     B(KK) = B(KK) + A(K,I+6*NN,IA)
  150             CONTINUE
C
               ENDIF
  130       CONTINUE
  200    CONTINUE
C
      ENDIF
C
      RETURN
      END
```

```
C
C-----------------------------------------------------------------------
C
      SUBROUTINE ASSEMR( MXPOIV, MXELE, MXNEQ,
     *                   INTMAT,  RELE,     B )
C
C     ASSEMBLE ELEMENT EQUATIONS INTO SYSTEM EQUATIONS
C
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION  RELE(15,MXELE), B(MXNEQ)
      DIMENSION  INTMAT(MXELE,6)
C
C     RESET SYSTEM LOAD VECTOR
C
      DO 100 I=1,MXNEQ
         B(I) = 0.
  100 CONTINUE
C
C     ASSEMBLING SYSTEM LOAD VECTOR
C
C     CONTRIBUTION OF VALUES ASSOCIATED WITH U & V VELOCITIES:
C
      DO 500 IE=1,MXELE
C
         DO 200  I=1,6
            II = INTMAT(IE,I)
            K  = I + 6
            KK = MXPOIV + II
            B(II) = B(II) + RELE(I,IE)
            B(KK) = B(KK) + RELE(K,IE)
  200    CONTINUE
C
C     CONTRIBUTION OF VALUES ASSOCIATED WITH PRESSURE:
C
         DO 300  I=1,3
            II = INTMAT(IE,I)
            K  = I + 12
            KK = 2*MXPOIV + II
            B(KK) = B(KK) + RELE(K,IE)
  300    CONTINUE
C
  500 CONTINUE
C
      RETURN
      END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE SPRSIN(AAS,N,NMAX,K,IJ,SB,IJB)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION AAS(N), SB(NMAX), IJB(NMAX)
*
*     CHANGE A(NP,NP) -> SB(NMAX):NONZERO ELEMENTS,IJB(NMAX):INDEX
*
      SB(IJ)=AAS(IJ)
        DO 10 J=1,N
          IF(ABS(AAS(J)).GE.1E-16) THEN
            IF(J.NE.IJ) THEN
              K=K+1
              IF(K.GT.NMAX) PAUSE 'NMAX TOO SMALL IN SPRSIN'
              SB(K)=AAS(J)
              IJB(K)=J
            ENDIF
          ENDIF
   10   CONTINUE
       IJB(IJ+1)=K+1
      RETURN
      END
C
C -------------------------------------------------------------
C
      SUBROUTINE SPRSTM(SA,IJA,SB,IJB,NMAX,SC,IJC)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION SA(NMAX), SB(NMAX), SC(NMAX)
      DIMENSION IJA(NMAX),IJB(NMAX),IJC(NMAX)
C     SA,IJA FOR A TRANSPOSE
C     SB,IJB FOR A
```

```
C     SC,IJC FOR A TRANSPOSE * A
C
      K=0
      IF(IJA(1).NE.IJB(1)) PAUSE 'SPRSTM SIZES DO NOT MATCH'
      K=IJA(1)
      IJC(1)=K
      DO 14 I=1,IJA(1)-2
        DO 13 J=1,IJB(1)-2
          IF(I.EQ.J)THEN
            SUM=SA(I)*SB(J)
          ELSE
            SUM=0.D0
          ENDIF
          MB=IJB(J)
          DO 11 MA=IJA(I),IJA(I+1)-1
            IJMA=IJA(MA)
            IF(IJMA.EQ.J)THEN
              SUM=SUM+SA(MA)*SB(J)
            ELSE
    2         IF(MB.LT.IJB(J+1))THEN
                IJMB=IJB(MB)
                IF(IJMB.EQ.I)THEN
                  SUM=SUM+SA(I)*SB(MB)
                  MB=MB+1
                  GOTO 2
                ELSE IF(IJMB.LT.IJMA)THEN
                  MB=MB+1
                  GOTO 2
                ELSE IF(IJMB.EQ.IJMA)THEN
                  SUM=SUM+SA(MA)*SB(MB)
                  MB=MB+1
                  GOTO 2
                ENDIF
              ENDIF
            ENDIF
   11     CONTINUE
          DO 12 MBB=MB,IJB(J+1)-1
            IF(IJB(MBB).EQ.I)THEN
              SUM=SUM+SA(I)*SB(MBB)
            ENDIF
   12     CONTINUE
          IF(I.EQ.J)THEN
            SC(I)=SUM
          ELSE IF(ABS(SUM).GT.1E-16)THEN
            IF(K.GT.NMAX)PAUSE 'SPRSTM: NMAX TO SMALL'
            SC(K)=SUM
            IJC(K)=J
            K=K+1
          ENDIF
   13   CONTINUE
        IJC(I+1)=K
   14 CONTINUE
      WRITE(*,*) K
       WRITE(9,*) K
      return
      END
C
C -------------------------------------------------------------
C
      SUBROUTINE SPRSAX(SB,IJB,X,B,N,NMAX)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION b(n), sb(nmax), x(n), ijb(nmax)
      IF(IJB(1).NE.N+2) PAUSE 'MISMATCHED VECTOR AND MATRIX IN APRSAX'
      DO 11 I=1,N
        B(I)=SB(I)*X(I)
        DO 12 K=IJB(I),IJB(I+1)-1
          B(I)=B(I)+SB(K)*X(IJB(K))
   12   CONTINUE
   11 CONTINUE
      RETURN
      END
C
C -------------------------------------------------------------
C
      SUBROUTINE PCGNR(N,SA,IJA,BN,MXNEQ,P,R,Z,Q,X,ITMAX,NMAX,TOL1)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION SA(NMAX), IJA(NMAX), BN(MXNEQ)
```

```
      DIMENSION P(MXNEQ), R(MXNEQ), Q(MXNEQ), X(MXNEQ), Z(MXNEQ)
*
      ITER1 = 0
      DO 140 I=1,N
       X(I)=0.0
  140 CONTINUE
      CALL ATIMES(SA,IJA,X,R,N,NMAX)
      DO 150 J=1,N
       R(J)=BN(J)-R(J)
  150 CONTINUE
  500 IF(ITER1.LE.ITMAX) THEN
      ITER1= ITER1+1
      CALL ASOLVE(N,R,Z,SA,NMAX)
      BKNUM=0.0
      DO 160 J=1,N
      BKNUM=BKNUM+Z(J)*R(J)
  160 CONTINUE
      IF(ITER1.EQ.1) THEN
       DO 170 J=1,N
        P(J)=Z(J)
  170  CONTINUE
      ELSE
       BK=BKNUM/BKDEN
       DO 180 J=1,N
        P(J)=Z(J)+BK*P(J)
  180  CONTINUE
      ENDIF
      CALL ATIMES(SA,IJA,P,Q,N,NMAX)
      BKDEN=BKNUM
      AKDEN=0.0
      DO 190 J=1,N
       AKDEN=AKDEN+P(J)*Q(J)
  190 CONTINUE
      AK=BKNUM/AKDEN
      DO 200 J=1,N
       X(J)=X(J)+AK*P(J)
       R(J)=R(J)-AK*Q(J)
  200 CONTINUE
      BNRM=SNRM(N,BN)
      ERR1=SNRM(N,R)/BNRM
      WRITE(*,201) ITER1, ERR1
  201 FORMAT(' ITER= ',I5,' ERR= ', F15.10)
      IF(ERR1.GT.TOL1) GOTO 500
      ENDIF
      RETURN
      END
C
C -------------------------------------------------------
C
      SUBROUTINE ATIMES(SA,IJA,X,B,N,NMAX)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION B(N), X(N), SA(NMAX), IJA(NMAX)
      IF(IJA(1).NE.N+2) PAUSE 'MISMATCHED VECTOR AND MATRIX IN ATIMES'
      DO 10 I=1,N
       B(I)=0.0
   10 CONTINUE
      DO 11 I=1,N
       B(I)=SA(I)*X(I)
        DO 12 K=IJA(I),IJA(I+1)-1
         B(I)=B(I)+SA(K)*X(IJA(K))
   12 CONTINUE
   11 CONTINUE
      RETURN
      END
C
C -------------------------------------------------------
C
      SUBROUTINE ASOLVE(N,B,X,SA,NMAX)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION B(N), X(N), SA(NMAX)
      DO 10 I=1,N
       X(I)=B(I)/SA(I)
   10 CONTINUE
      RETURN
      END
C
C -------------------------------------------------------
```

```
C
      FUNCTION SNRM(N,SX)
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION SX(N)
      SNRM=0.0
      DO 10 I=1,N
       SNRM=SNRM+SX(I)*SX(I)
   10 CONTINUE
       SNRM=SQRT(SNRM)
      RETURN
      END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE TRI( MXPOIV, MXELE, MXNEQ,   SOL,
     *                INTMAT, COORD, AKELE,  RELE,
     *                  IBCU,  IBCV,  IBCP,
     *                   DEN,   VIS )
C
C     ESTABLISH ALL ELEMENT MATRICES AND ASSEMBLE THEM TO FORM
C     UP SYSTEM EQUATIONS
C
      IMPLICIT REAL*4 (A-H,O-Z)
      DIMENSION  COORD(MXPOIV,2)
      DIMENSION  SOL(MXNEQ)
      DIMENSION  A(6,6), B(6,3), C(6,3), G(3,3), F(6,6,3)
      DIMENSION  UELE(6), VELE(6), PELE(3)
      DIMENSION  SXX(6,6), SXY(6,6), SYX(6,6), SYY(6,6)
      DIMENSION  HX(3,6), HY(3,6), HXT(6,3), HYT(6,3)
      DIMENSION  ABGXUG(6,6), AGBXUG(6,6), AGBYVG(6,6)
      DIMENSION  ABGYVG(6,6), ABGXVG(6,6), ABGYUG(6,6)
      DIMENSION  GXX(6,6), GYY(6,6), ALX(6,6), ALY(6,6)
      DIMENSION  AKELE(15,15,MXELE), RELE(15,MXELE)
      DIMENSION  FX(6), FY(6), FP(3)
      DIMENSION  IBCU(MXPOIV), IBCV(MXPOIV), IBCP(MXPOIV)
      DIMENSION  INTMAT(MXELE,6)
C
C     SET UP [A] MATRIX BASED ON TENSOR NOTATIONS:
C
      DO 10  I=1,6
      DO 10  J=1,6
         A(I,J) = 0.
   10 CONTINUE
      A(1,1) =  1.
      A(2,2) =  1.
      A(3,3) =  1.
      A(4,4) =  4.
      A(5,5) =  4.
      A(6,6) =  4.
      A(1,5) = -1.
      A(1,6) = -1.
      A(2,4) = -1.
      A(2,6) = -1.
      A(3,4) = -1.
      A(3,5) = -1.
C
C     COMPUTE KINEMATIC VISCOSITY:
C
      ANEW = VIS/DEN
C
C     LOOP OVER THE NUMBER OF ELEMENTS:
C
      DO 500  IE=1,MXELE
C
C     FIND ELEMENT LOCAL COORDINATES:
C
      II = INTMAT(IE,1)
      JJ = INTMAT(IE,2)
      KK = INTMAT(IE,3)
      LL = INTMAT(IE,4)
      MM = INTMAT(IE,5)
      NN = INTMAT(IE,6)
C
      XG1 = COORD(II,1)
      XG2 = COORD(JJ,1)
      XG3 = COORD(KK,1)
      YG1 = COORD(II,2)
```

```
      YG2 = COORD(JJ,2)
      YG3 = COORD(KK,2)
      AREA= 0.5*(XG2*(YG3-YG1) + XG1*(YG2-YG3) + XG3*(YG1-YG2))
      IF(AREA.LE.0.)  WRITE(6,15)  IE
   15 FORMAT(/,'  !!! ERROR !!!  ELEMENT NO.', I5,
     *         ' HAS NEGATIVE OR ZERO AREA ', /,
     *         '  --- CHECK F.E. MODEL FOR NODAL COORDINATES',
     *         ' AND ELEMENT NODAL CONNECTIONS ---'          )
      IF(AREA.LE.0.)  STOP
C
      AREA2 = 2.*AREA
      B1 = (YG2 - YG3)/AREA2
      B2 = (YG3 - YG1)/AREA2
      B3 = (YG1 - YG2)/AREA2
      C1 = (XG3 - XG2)/AREA2
      C2 = (XG1 - XG3)/AREA2
      C3 = (XG2 - XG1)/AREA2
C
C     SET UP [B] AND [C] MATRICES BASED ON TENSOR NOTATIONS:
C
      DO 30  I=1,6
      DO 30  J=1,3
         B(I,J) = 0.
         C(I,J) = 0.
   30 CONTINUE
      B(1,1) = 2.*B1
      B(2,2) = 2.*B2
      B(3,3) = 2.*B3
      B(4,2) = B3
      B(4,3) = B2
      B(5,1) = B3
      B(5,3) = B1
      B(6,1) = B2
      B(6,2) = B1
      C(1,1) = 2.*C1
      C(2,2) = 2.*C2
      C(3,3) = 2.*C3
      C(4,2) = C3
      C(4,3) = C2
      C(5,1) = C3
      C(5,3) = C1
      C(6,1) = C2
      C(6,2) = C1
C
C     SET UP [G] MATRIX:
C
      FAC  = AREA/12.
      FAC2 = 2.*FAC
      G(1,1) = FAC2
      G(2,2) = FAC2
      G(3,3) = FAC2
      G(1,2) = FAC
      G(1,3) = FAC
      G(2,1) = FAC
      G(2,3) = FAC
      G(3,1) = FAC
      G(3,2) = FAC
C
C     SET UP [F] MATRIX BASED ON TENSOR NOTATIONS:
C
      FACTOR = 2.*AREA/5040.
      F4   = FACTOR*4.
      F6   = FACTOR*6.
      F12  = FACTOR*12.
      F24  = FACTOR*24.
      F120 = FACTOR*120.
C
      F(1,1,1) = F120
      F(1,2,1) = F12
      F(1,3,1) = F12
      F(1,4,1) = F6
      F(1,5,1) = F24
      F(1,6,1) = F24
      F(2,2,1) = F24
      F(2,3,1) = F4
      F(2,4,1) = F6
      F(2,5,1) = F4
```

```
      F(2,6,1) = F12
      F(3,3,1) = F24
      F(3,4,1) = F6
      F(3,5,1) = F12
      F(3,6,1) = F4
      F(4,4,1) = F4
      F(4,5,1) = F4
      F(4,6,1) = F4
      F(5,5,1) = F12
      F(5,6,1) = F6
      F(6,6,1) = F12
      DO 40  I=1,6
      DO 40  J=I,6
         F(J,I,1) = F(I,J,1)
   40 CONTINUE
C
      F(1,1,2) = F24
      F(1,2,2) = F12
      F(1,3,2) = F4
      F(1,4,2) = F4
      F(1,5,2) = F6
      F(1,6,2) = F12
      F(2,2,2) = F120
      F(2,3,2) = F12
      F(2,4,2) = F24
      F(2,5,2) = F6
      F(2,6,2) = F24
      F(3,3,2) = F24
      F(3,4,2) = F12
      F(3,5,2) = F6
      F(3,6,2) = F4
      F(4,4,2) = F12
      F(4,5,2) = F4
      F(4,6,2) = F6
      F(5,5,2) = F4
      F(5,6,2) = F4
      F(6,6,2) = F12
      DO 50  I=1,6
      DO 50  J=I,6
         F(J,I,2) = F(I,J,2)
   50 CONTINUE
C
      F(1,1,3) = F24
      F(1,2,3) = F4
      F(1,3,3) = F12
      F(1,4,3) = F4
      F(1,5,3) = F12
      F(1,6,3) = F6
      F(2,2,3) = F24
      F(2,3,3) = F12
      F(2,4,3) = F12
      F(2,5,3) = F4
      F(2,6,3) = F6
      F(3,3,3) = F120
      F(3,4,3) = F24
      F(3,5,3) = F24
      F(3,6,3) = F6
      F(4,4,3) = F12
      F(4,5,3) = F6
      F(4,6,3) = F4
      F(5,5,3) = F12
      F(5,6,3) = F4
      F(6,6,3) = F4
      DO 60  I=1,6
      DO 60  J=I,6
         F(J,I,3) = F(I,J,3)
   60 CONTINUE
C
C     EXTRACT ELEMENT NODAL  U, V, P:
C
      UELE(1) = SOL(II)
      UELE(2) = SOL(JJ)
      UELE(3) = SOL(KK)
      UELE(4) = SOL(LL)
      UELE(5) = SOL(MM)
      UELE(6) = SOL(NN)
      VELE(1) = SOL(II+MXPOIV)
```

```
      VELE(2) = SOL(JJ+MXPOIV)
      VELE(3) = SOL(KK+MXPOIV)
      VELE(4) = SOL(LL+MXPOIV)
      VELE(5) = SOL(MM+MXPOIV)
      VELE(6) = SOL(NN+MXPOIV)
      PELE(1) = SOL(II+MXPOIV+MXPOIV)
      PELE(2) = SOL(JJ+MXPOIV+MXPOIV)
      PELE(3) = SOL(KK+MXPOIV+MXPOIV)
C
C     COMPUTE  [SXX], [SXY], [SYX], [SYY]  MATRICES:
C
      DO 100  IA=1,6
      DO 100  IB=1,6
         CXX = 0.
         CYY = 0.
         CXY = 0.
         CYX = 0.
         DO 110  I=1,6
         DO 110  J=1,3
         DO 110  K=1,3
         DO 110  L=1,6
            CXX = CXX + A(IA,I)*B(I,J)*A(IB,L)*B(L,K)*G(J,K)
            CYY = CYY + A(IA,I)*C(I,J)*A(IB,L)*C(L,K)*G(J,K)
            CXY = CXY + A(IA,I)*C(I,J)*A(IB,L)*B(L,K)*G(J,K)
            CYX = CYX + A(IA,I)*B(I,J)*A(IB,L)*C(L,K)*G(J,K)
  110 CONTINUE
         SXX(IA,IB) = 2.*ANEW*CXX + ANEW*CYY
         SXY(IA,IB) = ANEW*CXY
         SYX(IA,IB) = ANEW*CYX
         SYY(IA,IB) = ANEW*CXX + 2.*ANEW*CYY
  100 CONTINUE
C
C     COMPUTE  [HX] AND [HY]  MATRICES:
C     (SAME AS MATRICES ON THE LOWER LEFT OF LINEAR EQS.)
C
      DO 150  IA=1,3
      DO 150  IB=1,6
         CX = 0.
         CY = 0.
         DO 160  I=1,6
         DO 160  J=1,3
            CX = CX + A(IB,I)*B(I,J)*G(J,IA)
            CY = CY + A(IB,I)*C(I,J)*G(J,IA)
  160    CONTINUE
         HX(IA,IB) = CX
         HY(IA,IB) = CY
  150 CONTINUE
C
C     THEN THE CORRESPONDING TWO MATRICES ON THE UPPER RIGHT ARE:
C
      DO 170  IA=1,3
      DO 170  IB=1,6
         HXT(IB,IA) = -HX(IA,IB)
         HYT(IB,IA) = -HY(IA,IB)
  170 CONTINUE
C
C     COMPUTE ALL MATRICES ASSOCIATED WITH THE INERTIA TERMS:
C     (SEE DERIVATION IN NOTE FOR BETTER UNDERSTANDING)
C
      DO 200  IA=1,6
      DO 200  IB=1,6
         CABGXUG = 0.
         CAGBXUG = 0.
         CAGBYVG = 0.
         CABGYVG = 0.
         CABGXVG = 0.
         CABGYUG = 0.
         DO 210  I=1,6
         DO 210  J=1,6
         DO 210  K=1,6
         DO 210  L=1,6
         DO 210  M=1,3
            CABGXUG = CABGXUG
     *              + A(IA,I)*A(IB,J)*A(K,L)*B(L,M)*F(I,J,M)*UELE(K)
            CAGBXUG = CAGBXUG
     *              + A(IA,I)*A(K,J)*A(IB,L)*B(L,M)*F(I,J,M)*UELE(K)
            CAGBYVG = CAGBYVG
```

```
     *              + A(IA,I)*A(K,J)*A(IB,L)*C(L,M)*F(I,J,M)*VELE(K)
            CABGYVG = CABGYVG
     *              + A(IA,I)*A(IB,J)*A(K,L)*C(L,M)*F(I,J,M)*VELE(K)
            CABGXVG = CABGXVG
     *              + A(IA,I)*A(IB,J)*A(K,L)*B(L,M)*F(I,J,M)*VELE(K)
            CABGYUG = CABGYUG
     *              + A(IA,I)*A(IB,J)*A(K,L)*C(L,M)*F(I,J,M)*UELE(K)
  210    CONTINUE
         ABGXUG(IA,IB) = CABGXUG
         AGBXUG(IA,IB) = CAGBXUG
         AGBYVG(IA,IB) = CAGBYVG
         ABGYVG(IA,IB) = CABGYVG
         ABGXVG(IA,IB) = CABGXVG
         ABGYUG(IA,IB) = CABGYUG
  200 CONTINUE
C
      DO 220  I=1,6
      DO 220  J=1,6
         GXX(I,J) = ABGXUG(I,J) + AGBXUG(I,J) + AGBYVG(I,J) + SXX(I,J)
         GYY(I,J) = ABGYVG(I,J) + AGBYVG(I,J) + AGBXUG(I,J) + SYY(I,J)
         ALX(I,J) = ABGXVG(I,J) + SYX(I,J)
         ALY(I,J) = ABGYUG(I,J) + SXY(I,J)
  220 CONTINUE
C
C     THEN THE MATRIX (15X15) ON LHS OF THE ELEMENT EQS. IS:
C
C
      DO 240  I=1,6
         DO 250  J=1,6
            AKELE(I  ,J  ,IE) = GXX(I,J)
            AKELE(I+6,J+6,IE) = GYY(I,J)
            AKELE(I  ,J+6,IE) = ALY(I,J)
            AKELE(I+6,J  ,IE) = ALX(I,J)
  250    CONTINUE
         DO 260  J=1,3
            AKELE(I  ,J+12,IE) = HXT(I,J)
            AKELE(I+6,J+12,IE) = HYT(I,J)
  260    CONTINUE
  240 CONTINUE
      DO 270  I=1,3
      DO 270  J=1,6
         AKELE(I+12,J  ,IE) = HX(I,J)
         AKELE(I+12,J+6,IE) = HY(I,J)
  270 CONTINUE
C
C     BEGIN COMPUTING THE RESIDUALS ON RHS OF ELEMENT EQS.:
C
      DO 300  I=1,6
         TERM1 = 0.
         TERM2 = 0.
         TERM3 = 0.
         TERM4 = 0.
         TERM5 = 0.
         DO 310  J=1,6
            TERM1 = TERM1 + ABGXUG(I,J)*UELE(J)
            TERM2 = TERM2 + ABGYUG(I,J)*VELE(J)
            TERM4 = TERM4 + SXX(I,J)*UELE(J)
            TERM5 = TERM5 + SXY(I,J)*VELE(J)
  310    CONTINUE
         DO 320  J=1,3
            TERM3 = TERM3 + (1/DEN)*HXT(I,J)*PELE(J)
  320    CONTINUE
         FX(I) = TERM1 + TERM2 + TERM3 + TERM4 + TERM5
  300 CONTINUE
C
      DO 350  I=1,6
         TERM1 = 0.
         TERM2 = 0.
         TERM3 = 0.
         TERM4 = 0.
         TERM5 = 0.
         DO 360  J=1,6
            TERM1 = TERM1 + ABGXVG(I,J)*UELE(J)
            TERM2 = TERM2 + ABGYVG(I,J)*VELE(J)
            TERM4 = TERM4 + SYX(I,J)*UELE(J)
            TERM5 = TERM5 + SYY(I,J)*VELE(J)
  360    CONTINUE
```

```
         DO 370  J=1,3
            TERM3 = TERM3 + (1/DEN)*HYT(I,J)*PELE(J)
  370    CONTINUE
         FY(I) = TERM1 + TERM2 + TERM3 + TERM4 + TERM5
  350 CONTINUE
C
      DO 400  I=1,3
         TERM1 = 0.
         TERM2 = 0.
         DO 410  J=1,6
            TERM1 = TERM1 + HX(I,J)*UELE(J)
            TERM2 = TERM2 + HY(I,J)*VELE(J)
  410    CONTINUE
         FP(I) = TERM1 + TERM2
  400 CONTINUE
C
C     THUS THE RESIDUAL VECTOR ON RHS OF ELEMENT EQS. IS:
C
      DO 420  I=1,6
         RELE(I  ,IE) = -FX(I)
         RELE(I+6,IE) = -FY(I)
  420 CONTINUE
      DO 430  I=1,3
         RELE(I+12,IE) = -FP(I)
  430 CONTINUE
C
C     APPLY BOUNDARY CONDITIONS FOR ELEMENT MATRICES:
C
      CALL APPLYBC( MXPOIV, MXELE, AKELE,  RELE,
     *              INTMAT,  IBCU,  IBCV,  IBCP,    IE )
C
  500 CONTINUE
C
      RETURN
      END
```

ภาคผนวก ข.

รายละเอียดโปรแกรม BUILT

รายละเอียดโปรแกรม BUILT จะมีรายละเอียดเริ่มจากโปรแกรมหลัก และตามด้วยโปรแกรมย่อยต่าง ๆ ดังนี้

```
c*-----------------------------------------------------------------*
c*                                                                 *
c*                  program  B U I L T                             *
c*                                                                 *
c*    Surface Triangulation Program for Built-Up Structures        *
c*                     (version 1.2)                               *
c*    [modified to be used with 3-d built-up structures            *
c*     consisting of plate and membrane elements,                  *
c*                                Yupa's birthday, 10/28/91        *
c*                                                                 *
c*    Version 1.0 ......... July 1990                              *
c*            1.1 ......... December 1990                          *
c*            1.2 ......... October 1991                           *
c*                                                                 *
c*    Written by .......... Joaquim Peiro                          *
c*    Contact Address ..... Department of Aeronautics              *
c*                          Imperial College                       *
c*                          Prince Consort Road                    *
c*                          London SW7 2BY                         *
c*                          (071) 5895111 Ext. 4041                *
c*                                                                 *
c*    This program generates a surface triangulation given         *
c*    the definition of the surface in terms of edges and          *
c*    faces represented by means of composite curves and           *
c*    surfaces with curvature continuity.                          *
c*                                                                 *
c*    Note: the normal to the support surface is defined by        *
c*    u x v. this normal should agree with the orientation         *
c*    defined by the boundary segments of the surface to be        *
c*    generated.                                                   *
c*                                                                 *
c*    numbering of the support points of the surfaces              *
c*                                                                 *
c*                 1    4    7   10                               *
c*                  +----+----+----+ ----> v                       *
c*                 2|   5|   8|  11|                              *
c*                  +----+----+----+                               *
c*                 3|   6|   9|  12|                              *
c*                  +----+----+----+                               *
c*                  |                                              *
c*                  v                                              *
c*                  u           npu=3 , npv=4                      *
c*                                                                 *
c*                                                                 *
c*    input/output files                                           *
c*                                                                 *
c*    The convention for naming the input and output files         *
c*    is as follows. A run is assigned a problem name, say         *
c*    TEST, and a version, for instance 2; then the names          *
c*    assigned to the input and ouput files are:                   *
c*                                                                 *
c*    inp1 ..... geometrical definition of the surface             *
c*                This file does no change during the              *
c*                remeshing procedure                              *
c*                * file name > TEST.dat                           *
c*    inp2 ..... previous triangulation of the surface             *
```

```
c*                * file name > TEST.tri1                              *
c*    inp3 ...... local length coordinates of the points on          *
c*                the edges of the previous triangulation             *
c*                * file name > TEST.edg1                             *
c*    inp4 ...... local coordinates in the parameter plane           *
c*                of the points on the faces of the previous          *
c*                triangulation                                       *
c*                * file name > TEST.fac1                             *
c*                                                                    *
c*    inp5 ...... nodal values on the previous mesh of the           *
c*                spacings for the new triangulation                  *
c*                * file name > TEST.ref1                             *
c*                                                                    *
c*    Note : inp2 to inp5 are required only when remeshing.           *
c*           The files inp2 to inp3 have been produced by             *
c*           the program in a previous run. The file inp5             *
c*           could be obtained from the estimation of the             *
c*           errors after analysis or defined by the user.            *
c*           This second option would allow the user, after           *
c*           generating an initial coarse mesh, to define a           *
c*           new distribution of spacings which will enhance          *
c*           the resolution in chosen areas of the surface.           *
c*                                                                    *
c*    iou1 ...... current triangulation of the surface                *
c*                * file name > TEST.tri2                             *
c*    iou2 ...... local length coordinates of the points on           *
c*                the edges of the current triangulation              *
c*                * file name > TEST.edg2                             *
c*    iou3 ...... local coordinates in the parameter plane            *
c*                of the points on the faces of the current           *
c*                triangulation                                       *
c*                * file name > TEST.fac2                             *
c*                                                                    *
c*    Note: all these files are produced by the program.              *
c*                                                                    *
c*--------------------------------------------------------------------*
      parameter(mxnip=4000,mxnsp=10000,mxnsf=200,mxnbs=1000)
      parameter(mxnnp=4000,mxpbg=4000,mxebg=12000,mxnis=600)
      parameter(mxpst=4000,mxest=12000)
      parameter(mxpsf=6000,mxesf=20000)
c
      dimension lpip(mxnis),lpsp(mxnsf),lpbs(mxnsf),lpnp(mxnnp)
      dimension kpnp(mxnsf),kpne(mxnsf),lkel(mxnsf)
      dimension lkip(mxnis),igip(mxnip),alip(mxnip)
      dimension lknp(mxnsf),ignp(mxnnp),clnp(2,mxnnp)
      dimension nspl(mxnsf),nsp2(mxnsf),isbs(mxnbs)
      dimension nwed(mxnnp),nwfa(mxnnp)
      dimension cpip(3,mxnip),cpsp(3,mxnsp),spnp(mxnnp)
      dimension cosf(3,mxpsf),lmsf(3,mxesf),conp(3,mxnnp)
      dimension cobg(3,mxpbg),lmbg(3,mxebg),spbg(mxpbg)
      dimension cost(3,mxpst),lmst(4,mxest)
      dimension nib(mxnis)
      common /spa/ spac,spmin
c
      character filnam*12,cv*4,pv*4
c
c *** assign input and output channels
c
      inp1 = 21
      inp2 = 22
      inp3 = 23
      inp4 = 24
      inp5 = 25
      iou1 = 41
      iou2 = 42
      iou3 = 43
      iou4 =  8
      iou5 =  9
      iou6 = 90
      iou8 = 91
      iou9 = 92
      iou10= 94
C     iou11= 95
c
c *** mesh generation options:
c              0 ...... initial mesh (constant size)
c              else ... remeshing
```

```
c
      write(*,10)
      write(*,20)
      read(*,*) irem
c
c *** opens input/ouput files
c
      write(*,'(/,a,$)') ' Enter problem name: '
      read(*,'(a)') filnam
      write(*,'(a,$)') ' Enter current version number: '
      read(*,'(a)') cv
      l = namlen(filnam)
      if (l .eq. 0) go to 40
      open(inp1,file=filnam(1:l)//'.dat',status='old',err=40)
      open(iou1, file=filnam(1:l)//'.t'//cv ,status='unknown',err=40)
      open(iou2, file=filnam(1:l)//'.e'//cv ,status='unknown',err=40)
      open(iou3, file=filnam(1:l)//'.c'//cv ,status='unknown',err=40)
      open(iou4, file=filnam(1:l)//'.n'//cv ,status='unknown',err=40)
      open(iou5, file=filnam(1:l)//'.l'//cv ,status='unknown',err=40)
      open(iou6, file=filnam(1:l)//'.dim'   ,status='unknown',err=40)
      open(iou8, file=filnam(1:l)//'.f'//cv ,status='unknown',err=40)
      open(iou9, file=filnam(1:l)//'.d'//cv ,status='unknown',err=40)
      open(iou10,file=filnam(1:l)//'.i'//cv ,status='unknown',err=40)
      rewind(inp1)
c
c *** reads spacing for the first mesh or information for remeshing
c
      spmin = -1.
c
      if(irem.eq.0) then
        write(*,30)
        read(*,*) spac
        spmin = spac
      else
        write(*,'(a,$)') ' Enter previous version number: '
        read(*,'(a)') pv
        if(pv.eq.cv ) then
          write(*,'(a,$)') '...and so everything wil go lost !?'
          goto 40
        end if
        open(inp2,file=filnam(1:l)//'.t'//pv,status='old',err=40)
        open(inp3,file=filnam(1:l)//'.e'//pv,status='old',err=40)
        open(inp4,file=filnam(1:l)//'.c'//pv,status='old',err=40)
        open(inp5,file=filnam(1:l)//'.r'//pv,status='old',err=40)
        rewind(inp2)
        rewind(inp3)
        rewind(inp4)
        rewind(inp5)
      endif
c
c *** reads the geometrical definition of the surface
c
      call input(inp1,nis,nsf,lpip,cpip,
     *           lpsp,nspl,nsp2,cpsp,lpbs,isbs,nib)
c
c *** make dimension file of problem
c
      call size(iou6,nsf,cpsp)
c
c *** if remeshing reads the information of the previous mesh
c
      if(irem.ne.0) then
        call inpbg(inp2,npbg,nebg,cobg,lmbg,lkel)
        call inpsg(inp3,nis ,lkip,igip,alip)
        call inpsf(inp4,nsf ,lknp,ignp,clnp)
        call inpsp(inp5,npbg,spbg)
      endif
c
c *** generates points on the edges of the surface
c
      call genis(irem,nis ,lpip,cpip,lpnp,spnp,conp,
     .           lkip,igip,alip,spbg)
c
c *** generates points on the faces of the surface
c
      call gensf(irem,nsf ,lpbs,isbs,nspl,nsp2,lpnp,conp,
     .           lpsp,cpsp,kpne,kpnp,cosf,lmsf,npbg,lmbg,
```

```
     .           spbg,lkel,ignp,clnp,nwed,lknp)
c
c *** produces the global numbering
c
      call glnum(nis ,nsf ,npst,nest,lpnp,conp,kpnp,kpne,
     .           cosf,lmsf,lpbs,isbs,lpsp,cpsp,nspl,nsp2,
     .           cost,lmst,nwed,nwfa)
c
c *** outputs the necessary information for analysis and remeshing
c
      call outbg(iou1,npst,nest,cost,lmst)
      call outsg(iou2,nis ,lpnp,spnp,nwed)
      call outsf(iou3,nsf ,kpnp,cosf,nwfa)
      call datagraphic(iou8,iou9,npst,nest,cost,lmst)
c
   10 format(10(/),
     .       '          ************************************',/,
     .       '          ***        B  U  I  L  T         ***',/,
     .       '          ***     surface triangulator     ***',/,
     .       '          ***   for built-up structures    ***',/,
     .       '          ************************************',/)
   20 format(/,'  *** mesh generation ***',/,
     .         '      0.- initial mesh   ',/,
     .         '      1.- remeshing      ',//,
     .         '         Option ?: ',$)
   30 format(/,'  *** initial mesh ***',/,
     .         '      element size ?: ',$)
   40 stop
      end
c*--------------------------------------------------------------------*
c*    [input] reads the geometrical definition of the surface         *
c*--------------------------------------------------------------------*
      subroutine input(inp ,nis ,nsf,lpip,cpip,lpsp,nspl,nsp2,
     .                 cpsp,lpbs,isbs,nib)
c
      dimension lpip(*),lpsp(*),lpbs(*)
      dimension nspl(*),nsp2(*),isbs(*)
      dimension cpip(3,*),cpsp(3,*)
      dimension nib(*)
      character*80 text
c
      write(*,10)
c
c *** reads the number of edges and faces
c
      read(inp,'(a)') text
      read(inp,*) nis,nsf
c
c *** for each edge read the global coordinates of the points
c     in both faces (the points in the segment are ordered).
c
c     js   : number of the edge.
c     lpip : pointer of the position of the edge points in cpip.
c     cpip : 3D coordinates of the points in the edge.
c
      read(inp,'(a)') text
      kp = 0
      do 200 is=1,nis
      lpip(is) = kp+1
      read(inp,*) js,nip,nib(is)
      do 100 in=1,nip
      kp = kp+1
      read(inp,*) (cpip(il,kp),il=1,3)
  100 continue
  200 continue
      lpip(nis+1) = kp+1
c
c *** for each surface reads the coordinates of the support points.
c
c     js        : number of the face
c     lpsp      : pointer of the position of the support points in cpsp.
c     nspl,nsp2 : number of points in the u and v direction,respectively.
c     cpsp      : 3D coordinates of the support points.
c
      read(inp,'(a)') text
      kp = 0
      do 400 is=1,nsf
```

```
      read(inp,*) js,nsp1(js),nsp2(js)
      nn = nsp1(js)*nsp2(js)
      lpsp(is) = kp+1
      do 300 in=1,nn
      kp = kp+1
      read(inp,*) (cpsp(il,kp),il=1,3)
  300 continue
  400 continue
      lpsp(nsf+1) = kp+1
c
c *** reads in the right order the number of the edges that define
c     the boundary of the face to mesh ( a minus sign in the number
c     of the edge indicates that the correct orientation for this edge
c     is the opposite to its defining orientation).
c
c     nn    : number of edges defining the boundary of the face.
c     lpbs  : pointer of the position of the edges in isbs.
c     isbs  : number of the edges bounding the face to be triangulated.
c
      read(inp,'(a)') text
      kp1 = 1
      do 500 is=1,nsf
      lpbs(is) = kp1
      read(inp,*) js,nn
      kp2 = kp1+nn-1
      read(inp,*) (isbs(ik),ik=kp1,kp2)
      kp1 = kp2+1
  500 continue
      lpbs(nsf+1) = kp1
c
      close(inp)
c ...
   10 format(/,'  facet > reading surface definition data')
c
      return
      end
c*--------------------------------------------------------------------*
c*    [inpbg] reads the triangulation of the previous mesh            *
c*--------------------------------------------------------------------*
      subroutine inpbg(inp,npbg,nebg,cobg,lmbg,lkel)
      dimension cobg(3,*),lmbg(3,*),lkel(*)
c
c *** notation
c     npbg ..... number of nodes in the previous triangulation
c     nebg ..... number of elements in the previous triangulation
c     cobg ..... 3D coordinates of the nodes (stored but not used)
c     lmbg ..... connectivity array
c     lkel ..... pointer of elements in lmbg in each face
c
      write(*,10)
c
      read(inp,*) npbg,nebg
c
      do 100 ip=1,npbg
      read(inp,*) jp,(cobg(in,jp),in=1,3)
  100 continue
c
      isf0 = 1
      lkel(1) = 1
c
      do 200 ie=1,nebg
      read(inp,*) je,(lmbg(in,je),in=1,3),isf
      if(isf.ne.isf0) then
        isf0 = isf
        lkel(isf) = ie
      endif
  200 continue
c
      lkel(isf+1) = nebg+1
c
      close(inp)
c ...
   10 format(/,'  facet > reading previous triangulation')
c
      return
      end
c*--------------------------------------------------------------------*
```

```
c*    [inpsg] reads the arc lengths of points generated in the         *
c*    boundary edges of the previous mesh                              *
c*-----------------------------------------------------------------------*
      subroutine inpsg(inp,nis,lkip,igip,alip)
      dimension lkip(*),igip(*),alip(*)
c
      write(*,10)
c
c *** the information about the length and global number of the
c     points generated in the previous mesh along the edges of
c     the surface is stored in the vectors:
c
c     igip :  global number of the points generated along the edge is
c             (number in the surface definition) in the previous mesh
c     alip :  local length coordinate of these points
c     lkip :  pointer for the vectors igip, alip
c
      kp = 0
      do 200 is=1,nis
      lkip(is) = kp+1
      read(inp,*) nn
      do 100 ip=1,nn
      kp = kp+1
      read(inp,*) igip(kp),alip(kp)
  100 continue
  200 continue
      lkip(nis+1) = kp+1
c
      close(inp)
c ...
   10 format(/,'  facet > reading previous edge data')
c
      return
      end
c*-----------------------------------------------------------------------*
c*    [inpsf] reads the local coordinates of the points generated in   *
c*    the boundary faces of the previous mesh                          *
c*-----------------------------------------------------------------------*
      subroutine inpsf(inp,nsf,lknp,ignp,clnp)
      dimension lknp(*),ignp(*),clnp(2,*)
c
      write(*,10)
c
c *** the information about the local coordinates in the parameter
c     plane and global number of the points generated in the previous
c     mesh on the faces of the surface is stored in the vectors:
c
c     ignp :  global number of the points generated along the edge is
c             (number in the surface definition) in the previous mesh
c     clnp :  local coordinates of these points
c     lknp :  pointer for the vectors ignp, clnp
c
      kp = 0
      do 200 is=1,nsf
      lknp(is) = kp+1
      read(inp,*) nn
      do 100 ip=1,nn
      kp = kp+1
      read(inp,*) ignp(kp),(clnp(in,kp),in=1,2)
  100 continue
  200 continue
      lknp(nsf+1) = kp+1
c
      close(inp)
c ...
   10 format(/,'  facet > reading previous face data')
c
      return
      end
c*-----------------------------------------------------------------------*
c*    [inpsp] reads the spacings in the points of the previous mesh    *
c*-----------------------------------------------------------------------*
      subroutine inpsp(inp,npbg,spbg)
      dimension spbg(*)
      common /spa/ spac,spmin
c
      write(*,10)
```

```
c
c *** the array spbg contains the spacings for the new mesh
c     to be generated as nodal values of the previous mesh
c
      read(inp,*) np
      if(np.ne.npbg) then
        write(*,20)
        stop
      endif
      do 100 ip=1,npbg
      read(inp,*) jp,spbg(jp)
      spmin = min(spmin,spbg(jp))
  100 continue
c
      close(inp)
c ...
   10 format(/,'  facet > reading new spacings data')
   20 format(/,'  err-facet > subroutine inpsp',/,
     .        '      incompatible spacing file')
c
      return
      end
c*--------------------------------------------------------------------*
c*    [namlen] counts the number of characters in filnam              *
c*--------------------------------------------------------------------*
      integer function namlen(filnam)
c
      character*12 filnam
c
      namlen = 0
      do 10 i = 12,1,-1
         if (filnam(i:i) .eq. ' ') go to 10
         namlen = i
         go to 20
10    continue
20    return
      end
c*-------------------------------------------------------------*
c*    [split] generates points on a ferguson spline defined    *
c*    by nps points according to the spacing spi in the        *
c*    points of the previous discretization of the edge        *
c*-------------------------------------------------------------*
      subroutine split(irem,nps,npi,npn,xsp,tsp,xip,sip,xnp,snp)
c
      parameter (naux=2000)
      dimension xsp(3,*),tsp(3,*),xnp(3,*),xip(*),sip(*),snp(*)
      dimension al(naux),xne(naux),xl(naux),coef(5,naux)
      dimension r1(6),r2(6),r(6)
      common /spa/ spac,spmin
c
c *** notation:
c     nps ..... number of points defining the spline curve
c     npi ..... number of points in the previous discretization
c               of the edge (sampling points). unused if initial mesh.
c     npn ..... number of newly generated points
c     xsp ..... 3D coordinates of the points defining the spline
c     tsp ..... tangents vectors for the spline
c     xip ..... length coordinates of the sampling points
c     sip ..... element size at the sampling points
c     xnp ..... 3D coordinates of the newly generated points
c
      ndim = 3
      eps  = 1.e-05
c
      if(nps.gt.naux) then
        write(*,10) nps
        stop
      endif
c
c *** interpolates splines for the support points.
c
      do 300 id=1,ndim
      do 100 ip=1,nps
      xl(ip) = xsp(id,ip)
  100 continue
      call spline(2,nps,xl,xne,al)
      do 200 ip=1,nps
```

```
      tsp(id,ip) = xne(ip)
  200 continue
  300 continue
c
c *** calculates the length of each segment.
c
      u1  = 0.0
      u2  = 1.0
      ssp = 0.0
      rumin = 1.e+30
      ns = nps-1
c
      do 500 is=1,ns
      do 400 id=1,ndim
      id1 = id+ndim
      r1(id ) = xsp(id,is)
      r1(id1) = tsp(id,is)
      is1 = is+1
      r2(id ) = xsp(id,is1)
      r2(id1) = tsp(id,is1)
  400 continue
      call coeff(ndim,r1,r2,a1,a2,a3,a4,a5,rum)
      coef(1,is) = a1
      coef(2,is) = a2
      coef(3,is) = a3
      coef(4,is) = a4
      coef(5,is) = a5
      rumin = min(rumin,rum)
      call lengt(0,a1,a2,a3,a4,a5,u1,u2,s,eps)
      al(is) = s
      ssp = ssp+s
  500 continue
c
c *** calculates the number of elements and the resulting spacings.
c
      if(irem.eq.0) then
        ane = ssp/spac
        nel = max(int(ane+0.5),2)
        sdt = ssp/real(nel)
        npn = nel+1
        if(npn.gt.naux) then
          write(*,10) npn
          stop
        endif
        do 550 ip=1,npn
        ap = ip-1
        snp(ip) = ap*sdt
  550   continue
      else
        xne(1) = 0.0
        ane = 0.0
        do 600 is=1,npi-1
        d1 = sip(is)
        d2 = sip(is+1)
        xr = xip(is+1)-xip(is)
        av = (d2-d1)/xr
        if(av.le.eps) then
         anel = 2.*xr/(d1+d2)
        else
         anel = log(d2/d1)/av
        endif
        ane = ane+anel
        xne(is+1) = ane
  600   continue
        nel = max(int(ane+0.5),2)
        scl = ane/real(nel)
        npn = nel+1
        if(npn.gt.naux) then
          write(*,10) npn
          stop
        endif
        ik = 0
        x1 = 0.0
        ik = ik+1
        snp(ik) = 0.0
        an = real(ik)*scl
        do 900 is=1,npi-1
```

```
        an1 = xne(is)
        an2 = xne(is+1)
        xr  = xip(is+1)-xip(is)
        x2  = x1+xr
        if(an.gt.an2) goto 800
        d1 = sip(is)
        d2 = sip(is+1)
        av = (d2-d1)/xr
  700   continue
        ane = an-an1
        if(av.le.1.e-05) then
         xin = ane*0.5*(d1+d2)
        else
         xin = d1*(exp(av*ane)-1.)/av
        endif
        ik = ik+1
        snp(ik) = x1+xin
        an = real(ik)*sc1
        if(an.le.an2) goto 700
  800   continue
        x1 = x2
  900   continue
        snp(npn) = ssp
      endif
c
c *** places the points.
c
      ik = 1
      s1 = snp(ik+1)-snp(ik)
      np = 1
      do 1000 id = 1,ndim
      xnp(id,np) = xsp(id,1)
 1000 continue
c
      do 1400 is=1,ns
      u1 = 0.0
      u0 = is-1
      a1 = coef(1,is)
      a2 = coef(2,is)
      a3 = coef(3,is)
      a4 = coef(4,is)
      a5 = coef(5,is)
      do 1100 id=1,ndim
      id1 = id+ndim
      r1(id ) = xsp(id,is)
      r1(id1) = tsp(id,is)
      is1 = is+1
      r2(id ) = xsp(id,is1)
      r2(id1) = tsp(id,is1)
 1100 continue
      sis = a1(is)
 1200 continue
c
      if(s1.gt.sis) then
       s1 = s1-sis
       goto 1400
      else
       call markp(a1,a2,a3,a4,a5,rumin,u1,u2,s1)
       call fgcurv(0,ndim,r1,r2,u2,r)
       np = np+1
       do 1300 id=1,ndim
       xnp(id,np) = r(id)
 1300  continue
       if(np.eq.npn-1) go to 1450
       ik = ik+1
       s1 = s1+snp(ik+1)-snp(ik)
       if(s1.gt.eps) goto 1200
      endif
 1400 continue
 1450 continue
      do 1500 id=1,ndim
      xnp(id,npn) = xsp(id,nps)
 1500 continue
c ...
   10 format(/,'  err-facet > subroutine split',/,
     .        '     increase naux to: ',i7)
      return
```

```
      end
c*-------------------------------------------------------------*
c*    [genis] generates points in the edges of the surface     *
c*-------------------------------------------------------------*
      subroutine genis(irem,nis ,lpip,cpip,lpnp,spnp,conp,
     .                 lkip,igip,alip,spbg)
c
      parameter (naux=4000)
      dimension lpip(*),lpnp(*)
      dimension spnp(*),conp(3,*),cpip(3,*)
      dimension lkip(*),igip(*),alip(*),spbg(*)
      dimension xsp(3,naux),xnp(3,naux),tsp(3,naux)
      dimension xip(naux),sip(naux),snp(naux)
c
c *** notation
c     conp ..... vector containing the 3D coordinates of the points
c                generated in the edges of the surface.
c     spnp ..... vector containing the length coordinate of the
c                generated points (remeshing).
c     lpnp ..... is a pointer for the vectors conp and spnp.
c
      kp0 = 0
      lpnp(1) = 1
c
      do 2000 is=1,nis
      write(*,10) is
      is1  = is+1
      nip1 = lpip(is)
      nip2 = lpip(is1)-1
c
      nk = 0
      do 100 ip=nip1,nip2
      nk = nk+1
      xsp(1,nk) = cpip(1,ip)
      xsp(2,nk) = cpip(2,ip)
      xsp(3,nk) = cpip(3,ip)
  100 continue
      if(nk.gt.naux) then
        write(*,20) nk
        stop
      endif
c
c *** if remeshing fetch the spacings
c
      npi = 0
      if(irem.ne.0) then
        kp1 = lkip(is)
        kp2 = lkip(is+1)-1
        do 200 ip=kp1,kp2
        npi = npi+1
        xip(npi) = alip(ip)
        jp       = igip(ip)
        sip(npi) = spbg(jp)
  200   continue
      endif
      if(npi.gt.naux) then
        write(*,20) npi
        stop
      endif
c
c *** generates points along the edge
c
      call split(irem,nk,npi,npn,xsp,tsp,xip,sip,xnp,snp)
      if(npn.gt.naux) then
        write(*,20) npn
        stop
      endif
c
c *** stores the computed coordinates
c
      do 300 ip=1,npn
      kn = kp0+ip
      conp(1,kn) = xnp(1,ip)
      conp(2,kn) = xnp(2,ip)
      conp(3,kn) = xnp(3,ip)
  300 continue
c
```

```
c *** stores the length coordinate of the generated points
c
      do 1000 ip=1,npn
      kn = kp0+ip
      spnp(kn) = snp(ip)
 1000 continue
c
c *** updates the pointer
c
      write(*,30) npn
      kp0 = kp0+npn
      lpnp(is+1) = kp0+1
c
 2000 continue
c ...
   10 format(/,'  *** generating edge no: ',i3)
   20 format(/,'  err-facet > subroutine genis',/,
     .         '     increase naux to: ',i7)
   30 format(/,'      generated points: ',i7)
c
      return
      end
c*--------------------------------------------------------------------*
c*    [gensf] discretizes the faces of the surface given the edges    *
c*    that form their boundary and the distribution of element        *
c*    sizes in the previous mesh.                                     *
c     note: it is assumed that the boundary segments delimiting the   *
c     surface have been introduced in the right order of contiguity   *
c*--------------------------------------------------------------------*
      subroutine gensf(irem,nsf ,lpbs,isbs,nspl,nsp2,lpnp,conp,
     .                 lpsp,cpsp,kpne,kpnp,cosf,lmsf,npbg,lmbg,
     .                 spbg,lkel,ignp,clnp,lnew,lknp)
c
      parameter (mxpl=5000,mxel=10000,mxsl=16000)
c
      dimension lpbs(*),isbs(*),nspl(*),nsp2(*)
      dimension lpnp(*),conp(3,*),lpsp(*),cpsp(3,*)
      dimension cosf(2,*),lmsf(3,*),kpnp(*),kpne(*),lknp(*)
      dimension ignp(*),clnp(2,*),lnew(*),lkel(*),lmbg(3,*),spbg(*)
      dimension rl(12,mxpl),col(2,mxpl),lml(3,mxel)
      dimension cbg(2,mxpl),lbg(3,mxel),dbg(mxpl)
      dimension ipfr(mxsl),iqfr(mxsl),cog(3,mxsl)
      common /spa/ spac,spmin
c
      logical store
c
c *** tolerance for determining common points in terms of the
c     distance in 3D when forming the initial generation front.
c
      eps = 1.e-05
c
      np      = 0
      np0     = np
      ne      = 0
      kpnp(1) = 1
      kpne(1) = 1
c
c *** loop over the faces
c
      do 1000 is=1,nsf
      write(*,10) is
      is1  = is+1
      npu  = nspl(is)
      npv  = nsp2(is)
      n1   = lpsp(is)
      n2   = lpsp(is1)-1
      nks1 = lpbs(is)
      nks2 = lpbs(is1)-1
c
      if(npu*npv.gt.mxpl) then
        write(*,20) 'mxpl',npu*npv
        stop
      endif
c
c *** interpolates the composite spline surface for the face
c
      kp = 0
```

```
      do 100 ip = n1,n2
      kp = kp+1
      rl(1,kp) = cpsp(1,ip)
      rl(2,kp) = cpsp(2,ip)
      rl(3,kp) = cpsp(3,ip)
  100 continue
      call sfgeo(npu,npv,rl)
c
c *** forms a background mesh in the parameter plane
c
      if(irem.eq.0) then
        npg = 4
        neg = 2
        anu = npu-1
        anv = npv-1
        cbg(1,1) = 0.
        cbg(2,1) = 0.
        cbg(1,2) = anu
        cbg(2,2) = 0.
        cbg(1,3) = anu
        cbg(2,3) = anv
        cbg(1,4) = 0.
        cbg(2,4) = anv
        dbg(1) = spac
        dbg(2) = spac
        dbg(3) = spac
        dbg(4) = spac
        lbg(1,1) = 1
        lbg(2,1) = 2
        lbg(3,1) = 3
        lbg(1,2) = 1
        lbg(2,2) = 3
        lbg(3,2) = 4
      else
        do 200 ip=1,npbg
        lnew(ip) = 0
  200   continue
        ip1 = lknp(is)
        ip2 = lknp(is+1)-1
        npg = 0
        do 300 ip=ip1,ip2
        npg = npg+1
        cbg(1,npg) = clnp(1,ip)
        cbg(2,npg) = clnp(2,ip)
        jp       = ignp(ip)
        lnew(jp) = npg
        dbg(npg) = spbg(jp)
  300   continue
        ie1 = lkel(is)
        ie2 = lkel(is+1)-1
        neg = 0
        do 400 ie=ie1,ie2
        neg = neg+1
        i1 = lmbg(1,ie)
        i2 = lmbg(2,ie)
        i3 = lmbg(3,ie)
        lbg(1,neg) = lnew(i1)
        lbg(2,neg) = lnew(i2)
        lbg(3,neg) = lnew(i3)
  400   continue
      endif
c
c *** stores the boundary points
c
      npl = 0
      nel = 0
      store = .true.
c
      do 600 iks=nks1,nks2
      ks1 = isbs(iks)
      ks = abs(ks1)
c
c *** checks orientation of the segment
c
      if(ks1.gt.0) then
       np1 = lpnp(ks)
       np2 = lpnp(ks+1)-1
```

```
       jk = 1
      else
       npl = lpnp(ks+1)-1
       np2 = lpnp(ks)
       jk = -1
      endif
c
c *** stores value of first point for checking
c
      if(store) then
       ikpt  = npl+1
       xkpt  = conp(1,npl)
       ykpt  = conp(2,npl)
       zkpt  = conp(3,npl)
       store = .false.
      endif
c
c *** stores coordinates and forms the front
c
      do 500 ip=npl,np2-jk,jk
      npl        = npl+1
      cog(1,npl) = conp(1,ip)
      cog(2,npl) = conp(2,ip)
      cog(3,npl) = conp(3,ip)
      ipfr(npl)  = npl
      iqfr(npl)  = npl+1
  500 continue
c
      if(npl.gt.mxsl) then
        write(*,20) 'mxsl',npl
        stop
      endif
c
c *** checks against the first point.
c
      xdst = conp(1,np2)-xkpt
      ydst = conp(2,np2)-ykpt
      zdst = conp(3,np2)-zkpt
      dist = sqrt(xdst**2+ydst**2+zdst**2)
      if(dist.lt.eps) then
        iqfr(npl) = ikpt
        store = .true.
      endif
  600 continue
c
c *** computes the local coordinates
c
      call locuv(npl,cog,col,npu,npv,rl)
c
c *** generates the mesh in the parameter plane
c
      call msh2d(rl,npu,npv,neg,cbg,lbg,dbg,ipfr,iqfr,
     .           npl,nel,col,lml)
      if(npl.gt.mxpl) then
        write(*,20) 'mxpl',npl
        stop
      endif
      if(nel.gt.mxel) then
        write(*,20) 'mxel',nel
        stop
      endif
c
c *** stores the local coordinates and the element
c     connectivities in global vectors
c
      do 700 ip=1,npl
      np = np+1
      cosf(1,np) = col(1,ip)
      cosf(2,np) = col(2,ip)
  700 continue
c
      do 800 iel=1,nel
      ne = ne+1
      lmsf(1,ne) = lml(1,iel)+np0
      lmsf(2,ne) = lml(2,iel)+np0
      lmsf(3,ne) = lml(3,iel)+np0
  800 continue
```

```
c
      np0 = np
      kpnp(is1) = np+1
      kpne(is1) = ne+1
c
 1000 continue
c ...
   10 format(/,'  *** generating face no: ',i7)
   20 format(/,'  err-facet > subroutine gensf',/,
     .         '     increase ',a,' to: ',i7)
c
      return
      end
c*-------------------------------------------------------*
c*    [mesh2d] generates the mesh in the parameter plane   *
c*-------------------------------------------------------*
      subroutine msh2d(r,nu,nv,neg,cbg,lbg,dbg,
     .                 ipfr,iqfr,node,nelem,coor,iel)
c
      parameter (mxp = 8000, mxe = 16000)
      parameter (mxs = 2*mxe)
c                                                  ... previous mesh.
      dimension cbg(2,*),lbg(3,*),dbg(*)
c                                                  ... front vectors.
      dimension ipfr(*),iqfr(*)
c                                                  ... generated mesh.
      dimension coor(2,*),iel(3,*)
c                                                  ... surfaces.
      dimension r(12,*)
c                                                  ... help vectors.
      dimension coor0(2,mxp)
      dimension lcoor(mxp),lcoid(mxp) ,lwher(mxp)
      dimension lcore(mxp),icone(3*mxe),lposi(mxp)
      dimension nregi(mxp),strec(4,mxp),iside(4,mxs)
      dimension lhowm(mxp)
c
c *** initialize.
c
      call rfiliv(lcoor,mxp,0)
      call rfiliv(lcore,mxp,0)
      call rfiliv(icone,3*mxe,0)
      call rfiliv(lwher,mxp,0)
      call rfiliv(lhowm,mxp,0)
      call rfiliv(lposi,mxp,0)
      call rfiliv(lcoid,mxp,0)
c
      nonf=node
      nonr=node
      do 100 in=1,node
      nregi(in)=ipfr(in)
  100 continue
c
c *** triangulate regions
c
      call trian(r,nu,nv,nonf,ipfr,iqfr,nregi,iel,nonr,
     .           coor,nelem,node,neg,lbg,cbg,dbg)
c
c *** find out boundary points
c
      call bound(node,nelem,lcoor,iel)
c
c *** fill in iside
c
      call sides(nelem,node,nside,iel,iside,lwher,lhowm,icone)
c
c *** find out real and ideal number of connectivities
c
      call conre(node,nelem,iel,lcore)
      call conid(node,nside,iside,lcoor,lcoid,coor,strec,
     .           r,nu,nv,neg,lbg,cbg,dbg)
c
c *** cosmetics sequence.
c
      call eattr(node,nelem,nside,iel,iside,lcoor,lposi,
     .           lwher,lhowm,lcore,lcoid,icone,coor)
      do 111 ii=1,3
      call swapd(nside,iside,iel,
```

```
     .           lcore,lcoid,coor,nu,nv,r)
      nsmoo = 2
      call smoot(2,3,nelem,node,nsmoo,lcoor,lcore,iel,coor,coor0)
 111  continue
      call areach(node,nelem,iel,coor)
c
      return
      end
c*--------------------------------------------------------------*
c*    [triang] generates points and triangulates the region     *
c*--------------------------------------------------------------*
      subroutine trian(r,nu,nv,nonf,ipfr,iqfr,nregi,iel,nonr,
     .                 coor,nelem,node,neg,lbg,cbg,dbg)
c
      parameter (mxn=2000)
      dimension nregi(*),ipfr(*),iqfr(*)
      dimension iel(3,*),coor(2,*)
      dimension dv2(4),r(12,*)
      dimension cbg(2,*),dbg(*),lbg(3,*)
      dimension near(mxn),near1(mxn),howf(mxn),howf1(mxn)
      dimension ncheck(mxn)
c
c *** transformations stretched-unstretched
c
      xtr(xq,yq,ax,ay,alph) = alph*ax*xq-ay*yq
      ytr(xq,yq,ax,ay,alph) = alph*ay*xq+ax*yq
      xba(xq,yq,ax,ay,alph) = (ax*xq+ay*yq)/alph
      yba(xq,yq,ax,ay,alph) = -ay*xq+ax*yq
c
      aw = 1.1
      kount = 0
      fac = 1.25
c
      do 570 ik=1,node
      ncheck(ik)=-100
  570 continue
c
c *** set up ncheck values for region
c
      nk=0
      do 580 ik=1,nonf
      k1=ipfr(ik)
      ncheck(k1)=2
  580 continue
c
      disw = 0.0
c
  160 continue
      nl = nonf
  161 continue
      kn   = iqfr(nl)
      kn1  = ipfr(nl)
      xn   = coor(1,kn)
      yn   = coor(2,kn)
      xn1  = coor(1,kn1)
      yn1  = coor(2,kn1)
      xmp  = 0.5*(xn+xn1)
      ymp  = 0.5*(yn+yn1)
      call getsp(xmp,ymp,r,nu,nv,neg,cbg,lbg,dbg,dv2)
      spa  = dv2(1)
      alph = dv2(2)
      anx  = dv2(3)
      any  = dv2(4)
  170 continue
      xnf  = xba(xn,yn,anx,any,alph)
      ynf  = yba(xn,yn,anx,any,alph)
      xn1f = xba(xn1,yn1,anx,any,alph)
      yn1f = yba(xn1,yn1,anx,any,alph)
      x12f = xnf-xn1f
      y12f = ynf-yn1f
      alen1f=sqrt(x12f*x12f+y12f*y12f)
      xd1 = xn1-xn
      yd1 = yn1-yn
      al1 = sqrt(xd1*xd1+yd1*yd1)
      if(al1.lt.aw*disw) goto 162
      disw1=0.0
      call order(nl,ipfr,iqfr,coor,disw,disw1)
```

```
      goto 161
  162 continue
c
      tole1=0.00001*alen1f
c
c *** find out average spacing
c
      aver = spa
      x12=xn-xn1
      y12=yn-yn1
      alen1=sqrt(x12*x12+y12*y12)
c
c *** create a new node
c ***                              .... safety factor
      csafe=1.0
      dside=csafe*aver
      if(dside.gt.2.0*alen1f) dside=2.0*alen1f
      if(dside.lt.0.55*alen1f) dside=0.55*alen1f
      twod2=2.0*dside*dside
      xbarf=0.5*(xnf+xn1f)
      ybarf=0.5*(ynf+yn1f)
      xdiff=xnf-xn1f
      ydiff=ynf-yn1f
      dkarg=xdiff*xdiff+ydiff*ydiff
      d12=sqrt(dkarg)
      hkarg=0.5*twod2-0.25*d12*d12
      hk=sqrt(hkarg)
      d12=1./d12
      xcbf=-hk*ydiff*d12
      ycbf= hk*xdiff*d12
      xtempf=xbarf+xcbf
      ytempf=ybarf+ycbf
      xtemp=xtr(xtempf,ytempf,anx,any,alph)
      ytemp=ytr(xtempf,ytempf,anx,any,alph)
c
c *** loop over possible nodes - find closest neighbours
c
      a=yn1-yn
      b=xn-xn1
      c=(xn1-xn)*yn1+(yn-yn1)*xn1
      radius=dside
      h1=0.8*radius
      inum=0
      do 110 kp=1,nonr
      ken=nregi(kp)
      if(ken.eq.kn.or.ken.eq.kn1) go to 110
      xken=coor(1,ken)
      yken=coor(2,ken)
      if(a*xken+b*yken+c.le.0.0) goto 110
      xkenf=xba(xken,yken,anx,any,alph)
      ykenf=yba(xken,yken,anx,any,alph)
      xdiff1=xkenf-xtempf
      ydiff1=ykenf-ytempf
      distf=sqrt(xdiff1*xdiff1+ydiff1*ydiff1)
      if(distf.gt.h1) go to 110
c
      inum=inum+1
      howf1(inum)=distf
      near1(inum)=ken
  110 continue
c
c *** decide which of the nodes is chosen
c
      if(inum.gt.mxn) stop ' triang: increase mxn'
      if(inum.eq.0) then
       inum=1
       near(1)=0
       howf(1)=0.0
      else
c ***                                .... order them
       do 599 i=1,inum
       comp=1.e+6
       do 598 j=1,inum
       if(near1(j).eq.0) goto 598
       if(howf1(j).gt.comp) goto 598
       is=j
       comp=howf1(j)
```

```
  598  continue
       near(i)=near1(is)
       howf(i)=howf1(is)
       near1(is)=0
  599  continue
c ***                                  .... add the new point to the list
       inum=inum+1
       near(inum)=0
       howf(inum)=0.0
      endif
c
c *** select ---> start by the closest
c
      do 601 i=1,inum
      kp=near(i)
      if(kp.eq.0) then
        xp=xtemp
        yp=ytemp
      else
        xp=coor(1,kp)
        yp=coor(2,kp)
      endif
c
c *** see if this connection is possible
c
      call possib(kn1,kn,kp,xn1,yn1,xn,yn,xp,yp,nonf,
     .            nonr,ipfr,iqfr,nregi,coor,iyon)
c
      if(iyon.eq.0) goto 601
      goto 603
  601 continue
c
c *** we are in trouble !!!!!
c *** find the 30 existing nodes that give maximum angle
c
      inum=0
      angl=0.0
      do 210 kp=1,nonr
      ken=nregi(kp)
      if(ken.eq.kn.or.ken.eq.kn1) go to 210
      xken=coor(1,ken)
      yken=coor(2,ken)
      if(a*xken+b*yken+c.le.0.0) goto 210
c
c *** see if this connection is possible
c
      call possib(kn1,kn,ken,xn1,yn1,xn,yn,xken,yken,
     .            nonf,nonr,ipfr,iqfr,nregi,coor,iyon)
c
      if(iyon.eq.0) goto 210
      xkenf=xba(xken,yken,anx,any,alph)
      ykenf=yba(xken,yken,anx,any,alph)
      xdiff1=xkenf-xn1f
      ydiff1=ykenf-yn1f
      xdiff2=xkenf-xnf
      ydiff2=ykenf-ynf
      distf1=sqrt(xdiff1*xdiff1+ydiff1*ydiff1)
      distf2=sqrt(xdiff2*xdiff2+ydiff2*ydiff2)
      cosa=(xdiff1*xdiff2+ydiff1*ydiff2)/(distf1*distf2)
      if(cosa.gt.1.0) cosa=1.0
      if(cosa.lt.-1.0) cosa=-1.0
      angl=acos(cosa)
      if(angl.lt.angl) go to 210
c
      howf(inum+1)=angl
      near(inum+1)=ken
c
      if(inum.eq.0) goto 311
      do 310 i=1,inum
      k=i
      angi=howf(i)
      if(angi.gt.angl) goto 310
      do 410 j=1,inum-k
      l=inum-j
      near(l+1)=near(l)
      howf(l+1)=howf(l)
  410 continue
```

```
      near(k)=ken
      howf(k)=angl
      goto 311
  310 continue
  311 continue
      if(inum.lt.30) inum=inum+1
      if(inum.eq.30) angl=howf(30)
  210 continue
c
c *** select ---> start by the closest
c
      if(inum.eq.0) goto 703
      wfar=1.e+6
      do 701 i=1,inum
      kp=near(i)
      angi=howf(i)
      xp=coor(1,kp)
      yp=coor(2,kp)
c
c *** check now the size
c
      xpf=xba(xp,yp,anx,any,alph)
      ypf=yba(xp,yp,anx,any,alph)
      d1=sqrt((xpf-xn1f)**2+(ypf-yn1f)**2)
      d2=sqrt((xpf-xnf)**2+(ypf-ynf)**2)
      di=max(d1,d2)
      if(di.lt.wfar) then
       kp1=kp
       wfar=di
      endif
  701 continue
      kp=kp1
      xp=coor(1,kp)
      yp=coor(2,kp)
      goto 603
  703 continue
c
      print *,' cannot find the connectivity'
c
c *** try another side
c
      disw1=1.01*alen1f
      call order(n1,ipfr,iqfr,coor,disw,disw1)
      goto 161
c
  603 continue
c
c *** check differences of spacing.
c
      xmx = 0.5*(xmp+xp)
      ymx = 0.5*(ymp+yp)
      sp1 = aver*alph
      call getsp(xmx,ymx,r,nu,nv,neg,cbg,lbg,dbg,dv2)
      spa  = dv2(1)
      alph = dv2(2)
      anx  = dv2(3)
      any  = dv2(4)
      sp2 = spa*alph
      if(sp1.gt.fac*sp2) then
       kount = kount+1
       if(kount.le.2) goto 170
      endif
c
c *** nothing wrong with it
c
      kount=0
      indic=1
      knear=kp
      if(knear.ne.0) indic=0
c
      if(indic.eq.0) go to 620
      node=node+1
      coor(1,node)=xp
      coor(2,node)=yp
c
      nonr=nonr+1
      nuno=nonr
```

```
      nregi(nuno)=node
      knear=node
      ncheck(knear)=-100
  620 continue
c
c *** form element
c
      nelem=nelem+1
      iel(1,nelem)=kn1
      iel(2,nelem)=kn
      iel(3,nelem)=knear
c
      nk=nk+1
      if(nk/20.eq.1) then
       nk=0
       write(*,11) nelem,node,nl
   11  format('  nelem = ',i5,' npoin = ',i5,' nsfr = ',i5)
      endif
c
c ***  update front and active nodes
c
      nl2=nl+1
      iqfr(nl2)=kn
      iqfr(nl)=knear
      ipfr(nl2)=knear
      if(ncheck(knear).lt.0) ncheck(knear)=0
      ncheck(knear)=ncheck(knear)+2
      nonf=nl2
c
c *** delete sides from active list
c
      ncht=0
      npass=1
      ntop=kn1
      nbot=knear
      marker=0
      knon=nonf-2
  360 do 300 kp=1,knon
      ktop=ipfr(kp)
      kbot=iqfr(kp)
      if(marker.gt.0) go to 330
      if((ktop.eq.nbot).and.(kbot.eq.ntop)) go to 320
      go to 300
  320 marker=1
      nonf=nonf-2
      ncheck(ktop)=ncheck(ktop)-2
      ncheck(kbot)=ncheck(kbot)-2
      ncht=1
      go to 300
  330 kp1=kp-1
      ipfr(kp1)=ktop
      iqfr(kp1)=kbot
  300 continue
c
      npass=npass+1
      if(npass.gt.2) go to 340
      if(marker.eq.0) go to 350
      ipfr(nonf)=knear
      iqfr(nonf)=kn
      marker=0
      knon=knon-1
      go to 400
  350 knon=knon+1
  400 ntop=knear
      nbot=kn
      go to 360
  340 continue
c
c *** remove nodes from active list
c
      if(ncht.eq.0) go to 370
      ired=0
      knon=nonr
      do 380 kp=1,knon
      kkn=nregi(kp)
      kch=ncheck(kkn)
      if(kch.ne.0) go to 390
```

```
      ired=ired+1
      go to 380
  390 kp1=kp-ired
      nregi(kp1)=nregi(kp)
  380 continue
      nonr=knon-ired
  370 continue
      if(nonf.gt.0) go to 160
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine conid(npoin,nside,iside,lcoor,lcoid,coord,strec,
     .                 r,nu,nv,neg,lbg,cbg,dbg)
c
      parameter(pi=3.14159265358979328462)
c
      dimension iside(4,*),lcoor(*),lcoid(*)
      dimension coord(2,*),strec(4,*)
      dimension r(12,*),cbg(2,*),lbg(3,*),dbg(*),d2(4)
c
      xba(xq,yq,ax,ay,alph)=(ax*xq+ay*yq)/alph
      yba(xq,yq,ax,ay,alph)=-ay*xq+ax*yq
c
c *** interpolate the mesh parameters.
c
      do 500 i=1,npoin
      x=coord(1,i)
      y=coord(2,i)
      call getsp(x,y,r,nu,nv,neg,cbg,lbg,dbg,d2)
      strec(1,i)=d2(1)
      strec(2,i)=d2(2)
      strec(3,i)=d2(3)
      strec(4,i)=d2(4)
  500 continue
c
c *** finds out the optimal number of connectivities for each node.
c
      do 1000 ip=1,npoin
      lcoid(ip)=6
 1000 continue
c
c *** search for a side in the boundary to start.
c
      do 2000 is=1,nside
      ie = iside(4,is)
      if(ie.eq.0) then
        ib = iside(1,is)
        ic = iside(2,is)
        goto 2001
      endif
 2000 continue
 2001 continue
c
      imemo = ic
 4000 ia=ib-lcoor(ic)
      alph=strec(2,ic)
      anx=strec(3,ic)
      any=strec(4,ic)
      x1r=coord(1,ib)-coord(1,ic)
      y1r=coord(2,ib)-coord(2,ic)
      x2r=coord(1,ia)-coord(1,ic)
      y2r=coord(2,ia)-coord(2,ic)
      x1=xba(x1r,y1r,anx,any,alph)
      y1=yba(x1r,y1r,anx,any,alph)
      x2=xba(x2r,y2r,anx,any,alph)
      y2=yba(x2r,y2r,anx,any,alph)
      cosa=(x1*x2+y1*y2)/(sqrt(x1*x1+y1*y1)*sqrt(x2*x2+y2*y2))
      if(cosa.gt.1.0) cosa=1.0
      if(cosa.lt.-1.0) cosa=-1.0
      theta=acos(cosa)
      if((x2*y1-x1*y2).lt.0.0) theta=2.*pi-theta
      divi=(3.*theta/pi)+0.5
      lcoid(ic)=divi
      if(lcoid(ic).lt.1) lcoid(ic)=1
```

```
      ib=ic
      ic=ia
      if(ic.ne.imemo) goto 4000
c
      return
      end
c*--------------------------------------------------------------------*
c*    [getsp] gets the mesh parameters at a point in the parameter    *
c*    plane by interpolating from the previous triangulation and      *
c*    considering the usrface mapping                                 *
c*--------------------------------------------------------------------*
      subroutine getsp(u,v,r,nu,nv,neg,cbg,lbg,dbg,d2)
c
      dimension r(12,*),r1(12),r2(12),r3(12),r4(12),r5(9)
      dimension cbg(2,*),lbg(3,*),dbg(*),d2(4)
c
c *** determinant function
c
      deter(p1,q1,p2,q2,p3,q3)=p2*q3-p3*q2-p1*q3+p3*q1+p1*q2-p2*q1
c
c *** computes the tangent vectors to the surface at (u,v)
c
      iu = int(u)
      iv = int(v)
      if(iu.eq.(nu-1)) iu = iu-1
      if(iv.eq.(nv-1)) iv = iv-1
      ul = u-iu
      vl = v-iv
      ip1 = iu+iv*nu+1
      ip2 = ip1+1
      ip3 = ip1+nu
      ip4 = ip3+1
      do 100 id=1,12
      r1(id) = r(id,ip1)
      r2(id) = r(id,ip2)
      r3(id) = r(id,ip3)
      r4(id) = r(id,ip4)
  100 continue
      call fgsurf(2,r1,r2,r3,r4,ul,vl,r5)
      rp1 = r5(1)
      rp2 = r5(2)
      rp3 = r5(3)
      ru1 = r5(4)
      ru2 = r5(5)
      ru3 = r5(6)
      rv1 = r5(7)
      rv2 = r5(8)
      rv3 = r5(9)
c
c *** searches over the elements of the background mesh
c
      ac = -1.e+30
c
      do 200 ie=1,neg
      k1  = lbg(1,ie)
      k2  = lbg(2,ie)
      k3  = lbg(3,ie)
      x1  = cbg(1,k1)
      y1  = cbg(2,k1)
      x2  = cbg(1,k2)
      y2  = cbg(2,k2)
      x3  = cbg(1,k3)
      y3  = cbg(2,k3)
c
c *** performs the box-test
c
      bx1 = min(x1,x2,x3)
      bx2 = min(y1,y2,y3)
      bx3 = max(x1,x2,x3)
      bx4 = max(y1,y2,y3)
      if(bx1.gt.u) goto 200
      if(bx2.gt.v) goto 200
      if(bx3.lt.u) goto 200
      if(bx4.lt.v) goto 200
c
c *** computes the shape functions
c
```

```
      ar = deter(x1,y1,x2,y2,x3,y3)
      ar = 1./ar
      a1 = deter(u ,v ,x2,y2,x3,y3)*ar
      a2 = deter(x1,y1,u ,v ,x3,y3)*ar
      a3 = 1.-a1-a2
      am = min(a1,a2,a3)
      if(am.gt.ac) then
       j1 = k1
       j2 = k2
       j3 = k3
       ac = am
       b1 = a1
       b2 = a2
       b3 = a3
      endif
      if(ac.ge.0.) goto 300
c
  200 continue
c
  300 continue
c
c *** interpolates the spacings
c
      spc = b1*dbg(j1)+b2*dbg(j2)+b3*dbg(j3)
c
c *** calculates first fundamental form of the surface
c
      e  = ru1*ru1+ru2*ru2+ru3*ru3
      f  = ru1*rv1+ru2*rv2+ru3*rv3
      g  = rv1*rv1+rv2*rv2+rv3*rv3
c
c *** eigenvectors and eigenvalues.
c
      f2   = f*f
      emg  = e-g
      a    = 0.5*(e+g)
      b    = sqrt(0.25*emg*emg+f2)
      eig1 = a+b
      eig2 = a-b
      emax = max(e,g)
      epsi = abs(eig1-emax)
      if(epsi.lt.1.e-03) then
       if(e.ge.g) then
        eig1 = e
        eig2 = g
        v21  = 0.0
        v22  = 1.0
       else
        eig1 = g
        eig2 = e
        v21  = 1.0
        v22  = 0.0
       endif
      else
       em2 = e-eig2
       b2  = 1./sqrt(f2+em2*em2)
       v21 = -f*b2
       v22 = em2*b2
      endif
c
c *** 2d mesh parameters
c
      sp1   = spc/sqrt(eig1)
      sp2   = spc/sqrt(eig2)
      strec = sqrt(eig1/eig2)
      d2(1) = sp1
      d2(2) = strec
      d2(3) = v21
      d2(4) = v22
c
      return
      end
c*--------------------------------------------------------------------*
c*    [spline] finds the tangents in the points defining a ferguson   *
c*    splines. ib is an indicator of the end constraints.             *
c*                                                                    *
c*            ib  =  1 ..... specified tangents: t(1),t(n).           *
```

```
c*            ib  =  2 ..... zero second derivatives.           *
c*                                                              *
c*    note: the number of points is n. the tridiagonal system of n-2   *
c*    equations is solved by gauss elimination & backsubstitution.     *
c*--------------------------------------------------------------------*
      subroutine spline(ib,n,r,t,aux)
c
      dimension aux(*),r(*),t(*)
c
c *** first row i = 2.   ib = 1  -> [ 4, 1] ;  ib = 2  -> [ 3.5, 1]
c
      if(n.le.1) then
       pause ' spline: n=1 wrong number of points'
      else if(n.eq.2) then
       t(1) = r(2)-r(1)
       t(2) = t(1)
      else if(n.eq.3) then
       if(ib.eq.1) then
        t(2) = 0.25*(3.*(r(3)-r(1))-t(1)-t(3))
       else
        t(1) = -1.25*r(1)+1.50*r(2)-0.25*r(3)
        t(2) = -0.50*r(1)          +0.50*r(3)
        t(3) =  0.25*r(1)-1.50*r(2)+1.25*r(3)
       endif
      else
       rv = 3.*(r(3)-r(1))
       if(ib.eq.1) then
        bet = 4.0
        rv = rv-t(1)
       else
        bet = 3.5
        rv = rv-1.5*(r(2)-r(1))
       endif
       t(2) = rv/bet
c
c *** rows of the type  [ 1,  4,  1 ]
c
       do 100 j=3,n-2
       aux(j) = 1./bet
       bet = 4.-aux(j)
       if(bet.eq.0.) pause ' error in spline: zero pivot'
       rv = 3.*(r(j+1)-r(j-1))
       t(j) = (rv-t(j-1))/bet
  100  continue
c
c *** last row i = n-1   ib = 1  -> [ 1, 4] ;  ib = 2  -> [ 1, 3.5 ].
c
       aux(n-1) = 1./bet
       rv = 3.*(r(n)-r(n-2))
       if(ib.eq.1) then
        bet = 4.
        rv = rv-t(n)
       else
        bet = 3.5
        rv = rv-1.5*(r(n)-r(n-1))
       endif
       bet = bet-aux(n-1)
       if(bet.eq.0.) stop ' error in spline: zero pivot'
       t(n-1) = (rv-t(n-2))/bet
c
c *** backsubstitution.
c
       do 200 j=n-2,2,-1
       t(j) = t(j)-aux(j+1)*t(j+1)
  200  continue
c
c *** end values when ib = 2.
c
       if(ib.eq.2) then
        t(1) = 1.5*(r(2)-r(1  ))-0.5*t(2)
        t(n) = 1.5*(r(n)-r(n-1))-0.5*t(n-1)
       endif
      endif
c
      return
      end
c
```

```
c ----------------------------------------------------------------------
c
      subroutine coeff(ndim,r1,r2,a1,a2,a3,a4,a5,rumin)
c
c *** this sub. computes the coeficients of the polynomial |r'|**2
c
      dimension r1(2*ndim),r2(2*ndim)
c
      a1 = 0.0
      a2 = 0.0
      a3 = 0.0
      a4 = 0.0
      a5 = 0.0
      am1 = 0.
      am2 = 0.
c
      do 1000 id=1,ndim
      id1 = id+ndim
      r12 = r1(id)-r2(id)
      p = r1(id1)
      q = r2(id1)
      pp = p*p
      qq = q*q
      s = 3.*( 2.*r12+   p+q)
      t = 2.*(-3.*r12-2.*p-q)
      a1 = a1+pp
      a2 = a2+2.*p*t
      a3 = a3+t*t+2.*p*s
      a4 = a4+2.*t*s
      a5 = a5+s*s
      am1 = am1+pp
      am2 = am2+qq
 1000 continue
c
      rumin = min(am1,am2)
      rumin = sqrt(rumin)
c
      return
      end
c
c ----------------------------------------------------------------------
c
      subroutine lengt(in,a1,a2,a3,a4,a5,u1,u2,s,eps)
c
c *** this sub. computes the length of a segment of a cubic.
c
      parameter(nit=20)
c
      eps1 = eps
      os = -1.e+30
c
      f1 = sqrt(a1+u1*(a2+u1*(a3+u1*(a4+u1*a5))))
      f2 = sqrt(a1+u2*(a2+u2*(a3+u2*(a4+u2*a5))))
c
      u21 = u2-u1
      st = 0.5*u21*(f1+f2)
      ost = st
      kt = 1
      do 200 it=1,nit
      tnm = kt
      del = u21/tnm
      x = u1+0.5*del
      sum = 0.0
      do 100 jt=1,kt
      sum = sum+sqrt(a1+x*(a2+x*(a3+x*(a4+x*a5))))
      x = x+del
  100 continue
      st = 0.5*(st+u21*sum/tnm)
      kt = kt*2
      s = (4.*st-ost)/3.
      if(in.eq.0) eps1 = eps*abs(os)
      if(abs(s-os).le.eps1) goto 300
      os = s
      ost = st
  200 continue
      stop ' lengt: number of iterations exceeded'
  300 return
```

```
      end
c
c ---------------------------------------------------------------------
c
      subroutine markp(a1,a2,a3,a4,a5,rumin,u1,u2,sl)
c
c *** this sub. calculates the position u2 of a point on a f.s. such that
c     the length of the cubic segment u1,u2 is sl.
c
      parameter(nit=20)
c
      eps1 = 1.e-03
      eps2 = rumin*eps1*1.e-02
      ux = u1
      u2 = u1
      ss = sl
      do 100 it=1,nit
      ff = sqrt(a1+u2*(a2+u2*(a3+u2*(a4+u2*a5))))
      u2 = u2+ss/ff
      if(abs(u2-ux).le.eps1) goto 200
      call lengt(1,a1,a2,a3,a4,a5,ux,u2,vi,eps2)
      ss = ss-vi
      ux = u2
  100 continue
      stop ' markp: number of iterations exceeded'
  200 return
      end
c
c ---------------------------------------------------------------------
c
      subroutine sfgeo(npg,ngs,r)
c
c *** given the coordinates of the generated points on the surface,
c     calculates the parameters defining the ferguson splines.
c
      parameter(naux=4000 ,ndim=3)
c
      dimension r(12,1)
      dimension v1(naux),v2(naux),v3(naux)
c
      if(npg.gt.naux.or.ngs.gt.naux) pause ' sfgeo: increase naux'
c
c *** interpolates ru in the u direction.
c
      do 400 id=1,ndim
      id1 = id+ndim
      do 300 ig=1,ngs
      n0 = (ig-1)*npg
      do 100 ip=1,npg
      ik = ip+n0
      v1(ip) = r(id,ik)
  100 continue
      call spline(2,npg,v1,v2,v3)
      do 200 ip=1,npg
      ik = ip+n0
      r(id1,ik) = v2(ip)
  200 continue
  300 continue
  400 continue
c
c *** interpolates rv in the v direction.
c
      do 800 id=1,ndim
      id2 = id+2*ndim
      do 700 ip=1,npg
      do 500 ig=1,ngs
      ik = ip+(ig-1)*npg
      v1(ig) = r(id,ik)
  500 continue
      call spline(2,ngs,v1,v2,v3)
      do 600 ig=1,ngs
      ik = ip+(ig-1)*npg
      r(id2,ik) = v2(ig)
  600 continue
  700 continue
  800 continue
c
```

```
c *** interpolates ruv.
c
      do 2000 id=1,ndim
      id1 = id +ndim
      id2 = id1+ndim
      id3 = id2+ndim
c ***                              ! interpolates in the v direction.
      do 900 ig=1,ngs
      ik = (ig-1)*npg+1
      v1(ig) = r(id1,ik)
  900 continue
      call spline(2,ngs,v1,v2,v3)
      do 1000 ig=1,ngs
      ik = (ig-1)*npg+1
      r(id3,ik) = v2(ig)
 1000 continue
      do 1100 ig=1,ngs
      ik = ig*npg
      v1(ig) = r(id1,ik)
 1100 continue
      call spline(2,ngs,v1,v2,v3)
      do 1200 ig=1,ngs
      ik = ig*npg
      r(id3,ik) = v2(ig)
 1200 continue
c ***                              ! interpolates in the u direction.
      do 1500 ig=1,ngs
      n0 = (ig-1)*npg
      do 1300 ip=1,npg
      ik = n0+ip
      v1(ip) = r(id2,ik)
 1300 continue
      v2(1)   = r(id3,n0+1)
      v2(npg)= r(id3,n0+npg)
      call spline(1,npg,v1,v2,v3)
      do 1400 ip=2,npg-1
      ik = n0+ip
      r(id3,ik) = v2(ip)
 1400 continue
 1500 continue
 2000 continue
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine fgsurf(ider,r1,r2,r3,r4,u,v,r)
c
c *** expression of a ferguson surface patch and its derivatives.
c     in the vector r are stored the values:  r, ru, rv, ruv, ruu, rvv
c
c                    (0,1)            (1,1)
c                      +----------------+
c                      | 3            4 |        ^ v
c                      |                |        |
c                      | 1            2 |        |
c                      +----------------+        +----> u
c                    (0,0)            (1,0)
c
      parameter(ndim=3 ,nd4=4*ndim, nd6=6*ndim)
c
      dimension r1(nd4),r2(nd4),r3(nd4),r4(nd4),r(nd6)
c
      do 1000 id=1,ndim
c
      id1 = id +ndim
      id2 = id1+ndim
      id3 = id2+ndim
      id4 = id3+ndim
      id5 = id4+ndim
c
      q11 = r1(id)
      q12 = r3(id)
      q13 = r1(id2)
      q14 = r3(id2)
      q21 = r2(id)
```

```
      q22 = r4(id)
      q23 = r2(id2)
      q24 = r4(id2)
      q31 = r1(id1)
      q32 = r3(id1)
      q33 = r1(id3)
      q34 = r3(id3)
      q41 = r2(id1)
      q42 = r4(id1)
      q43 = r2(id3)
      q44 = r4(id3)
c
      s1 = -3.*q11+3.*q12-2.*q13-q14
      s2 = -3.*q21+3.*q22-2.*q23-q24
      s3 = -3.*q31+3.*q32-2.*q33-q34
      s4 = -3.*q41+3.*q42-2.*q43-q44
      t1 = 2.*q11-2.*q12+q13+q14
      t2 = 2.*q21-2.*q22+q23+q24
      t3 = 2.*q31-2.*q32+q33+q34
      t4 = 2.*q41-2.*q42+q43+q44
c
      a11 = q11
      a12 = q13
      a13 = s1
      a14 = t1
      a21 = q31
      a22 = q33
      a23 = s3
      a24 = t3
      a31 = -3.*q11+3.*q21-2.*q31-q41
      a32 = -3.*q13+3.*q23-2.*q33-q43
      a33 = -3.*s1 +3.*s2 -2.*s3 -s4
      a34 = -3.*t1 +3.*t2 -2.*t3 -t4
      a41 = 2.*q11-2.*q21+q31+q41
      a42 = 2.*q13-2.*q23+q33+q43
      a43 = 2.*s1 -2.*s2 +s3 +s4
      a44 = 2.*t1 -2.*t2 +t3 +t4
c
      s10 = a11+v*(a12+v*(a13+v*a14))
      s20 = a21+v*(a22+v*(a23+v*a24))
      s30 = a31+v*(a32+v*(a33+v*a34))
      s40 = a41+v*(a42+v*(a43+v*a44))
c
      r(id) = s10+u*(s20+u*(s30+u*s40))
c
      if(ider.eq.0) goto 1000
c
      r(id1) = s20+u*(2.*s30+u*3.*s40)
c
      s1 = a12+v*(2.*a13+v*3.*a14)
      s2 = a22+v*(2.*a23+v*3.*a24)
      s3 = a32+v*(2.*a33+v*3.*a34)
      s4 = a42+v*(2.*a43+v*3.*a44)
c
      r(id2) = s1+u*(s2+u*(s3+u*s4))
      r(id3) = s2+u*(2.*s3+u*3.*s4)
c
      if(ider.eq.1) goto 1000
c
      r(id4) = 2.*s30+u*6.*s40
c
      s1 = 2.*a13+v*6.*a14
      s2 = 2.*a23+v*6.*a24
      s3 = 2.*a33+v*6.*a34
      s4 = 2.*a43+v*6.*a44
c
      r(id5) = s1+u*(s2+u*(s3+u*s4))
c
 1000 continue
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine fgcurv(ider,ndim,r1,r2,u,r)
c
```

```
c *** expression of a ferguson curve segment.
c
      dimension r1(3*ndim),r2(3*ndim),r(3*ndim)
c
      do 1000 id=1,ndim
c
      id1 = id+ndim
      id2 = id1+ndim
      r12 = r2(id)-r1(id)
      a1 = r1(id)
      a2 = r1(id1)
      a3 =  3.*r12-2.*r1(id1)-r2(id1)
      a4 = -2.*r12+   r1(id1)+r2(id1)
      r(id) = a1+u*(a2+u*(a3+u*a4))
      if(ider.eq.0) goto 1000
      r(id1) = a2+u*(2.*a3+3.*u*a4)
      if(ider.eq.1) goto 1000
      r(id2) = 2.*a3+6.*u*a4
c
 1000 continue
c
      return
      end
c*--------------------------------------------------------------------*
c*    [locuv] calculates the coordinates u,v in the parameter plane   *
c*    of a point on a Ferguson surface given by its 3D coordinates    *
c*--------------------------------------------------------------------*
      subroutine locuv(np,xp,xl,nu,nv,r)
c
      parameter (nd4=12,nit=150)
      dimension r(nd4,*),r1(nd4),r2(nd4),r3(nd4),r4(nd4),r5(nd4)
      dimension xp(3,*),xl(2,*)
      common /spa/ spac,spmin
c
      eps1 = 1.e-10
      eps2 = 1.e-05
      eps3 = 1.e-15
c
      anu  = nu-1
      anv  = nv-1
      sc   = 0.2
c
      do 3000 ip=1,np
      x = xp(1,ip)
      y = xp(2,ip)
      z = xp(3,ip)
c
c *** initial guess:(u,v) of the closest support point
c
      dmn = 1.e+27
      do 22 ip1=1,nu*nv
      xm = r(1,ip1)-x
      ym = r(2,ip1)-y
      zm = r(3,ip1)-z
      dt = sqrt(xm*xm+ym*ym+zm*zm)
      if(dt.lt.dmn) then
        dmn = dt
        ipk = ip1
      endif
   22 continue
      iv = (ipk-1)/nu+1
      iu = ipk-(iv-1)*nu
      un = iu-1
      vn = iv-1
c
c *** iteration for finding the local coordinates
c
      do 1000 it=1,nit
      iu  = int(un)
      iv  = int(vn)
      if(iu.eq.(nu-1)) iu = iu-1
      if(iv.eq.(nv-1)) iv = iv-1
      ul  = un-iu
      vl  = vn-iv
      ip1 = iu+iv*nu+1
      ip2 = ip1+1
      ip3 = ip1+nu
```

```
      ip4 = ip3+1
      do 100 id=1,nd4
      r1(id) = r(id,ip1)
      r2(id) = r(id,ip2)
      r3(id) = r(id,ip3)
      r4(id) = r(id,ip4)
  100 continue
      call fgsurf(1,r1,r2,r3,r4,ul,vl,r5)
      rxn = r5(1)
      ryn = r5(2)
      rzn = r5(3)
      rux = r5(4)
      ruy = r5(5)
      ruz = r5(6)
      rvx = r5(7)
      rvy = r5(8)
      rvz = r5(9)
c
c *** distance between current point and target point
c
      drx = x-rxn
      dry = y-ryn
      drz = z-rzn
      drm = sqrt(drx*drx+dry*dry+drz*drz)
c
c *** end of iteration check in global coordinates
c
      if(drm.lt.0.0001*spmin) goto 2000
c
c *** calculates the projected vector in the tangent plane
c     and the direction of advance in the parameter plane
c
      e = rux*rux+ruy*ruy+ruz*ruz
      g = rvx*rvx+rvy*rvy+rvz*rvz
      if(e.le.eps2) then
        dun = 0.
        dvn = (drx*rvx+dry*rvy+drz*rvz)/g
      else if(g.le.eps2) then
        dun = (drx*rux+dry*ruy+drz*ruz)/e
        dvn = 0.
      else if(e.gt.eps2.and.g.gt.eps2) then
        dun = (drx*rux+dry*ruy+drz*ruz)/e
        dvn = (drx*rvx+dry*rvy+drz*rvz)/g
      else
        write(*,30)
        stop
      endif
c
c *** end of iteration check in local coordinates
c
      tol = sqrt(dun*dun+dvn*dvn)
      if(tol.lt.eps2) goto 2000
c
c *** new position
c
      un = un+sc*dun
      vn = vn+sc*dvn
      un = max(un,0.0)
      vn = max(vn,0.0)
      un = min(un,anu)
      vn = min(vn,anv)
 1000 continue
      write(*,20)
 2000 continue
      if(drm.gt.0.01*spmin) write(*,10) drm,ip
      xl(1,ip) = un
      xl(2,ip) = vn
 3000 continue
c ...
   10 format('  facet-war > distance = ',f8.5,' for point ',i5)
   20 format(//,'  facet-war > subroutine locuv',/,
     .       '       number of iterations exceeded')
   30 format(//,'  facet-err > subroutine locuv',/,
     .       '       null tangent vectors')
c
      return
      end
```

```
c
c ---------------------------------------------------------------------
c
      subroutine inter(iin,x1,y1,x2,y2,x3,y3,x,y)
c
c ***  determinant function
c
      deter(p1,q1,p2,q2,p3,q3)=p2*q3-p3*q2-p1*q3+p3*q1+p1*q2-p2*q1
c
      iin=1
      area2=deter(x1,y1,x2,y2,x3,y3)
      if(area2.lt.1.e-12) return
      area2=1./area2
      a1=deter(x ,y ,x2,y2,x3,y3)*area2
      a2=deter(x1,y1,x ,y ,x3,y3)*area2
      a3=1.-a1-a2
c
      wcomp=min(a1,a2,a3)
      if(wcomp.lt.-0.001) iin=0
c
      return
      end
c
c ---------------------------------------------------------------------
c
     subroutine possib(kn1,kn,kp,xn1,yn1,xn,yn,xp,yp,nonf,
     *                  nonr,ipfr,iqfr,nregi,coor,iyon)
c
c ***  this subroutine finds out whether connection whith point kp
c ***  is possible iyon=1 or not iyon=0
c
      dimension nregi(1),ipfr(1),iqfr(1),coor(2,1)
c
      iyon=1
c
c ***  loop over the front nodes
c
      do 1000 it=1,nonr
      kj=nregi(it)
      if(kj.eq.kn1.or.kj.eq.kn.or.kj.eq.kp) goto 1000
      xt=coor(1,kj)
      yt=coor(2,kj)
c
c ***  check if the point is interior
c
      call inter(iin,xn1,yn1,xn,yn,xp,yp,xt,yt)
      if(iin.eq.0) goto 1000
      iyon=0
      return
 1000 continue
c
c ***  equation of the mid-base : kp line
c
      xmb=0.5*(xn1+xn)
      ymb=0.5*(yn1+yn)
      as=ymb-yp
      bs=xp-xmb
      cs=(xmb-xp)*ymb+(yp-ymb)*xmb
c
c ***  loop over the front sides : check for intersection
c
      do 2000 ir=1,nonf
      kntl=ipfr(ir)
      if(kntl.eq.0) goto 2000
      knt=iqfr(ir)
c
      if(kntl.eq.kn1.and.knt.eq.kn) goto 2000
      if(kntl.eq.kp.or.knt.eq.kp) goto 2000
c
      xntl=coor(1,kntl)
      yntl=coor(2,kntl)
      xnt=coor(1,knt)
      ynt=coor(2,knt)
c
      at=yntl-ynt
      bt=xnt-xntl
      ct=(xntl-xnt)*yntl+(ynt-yntl)*xntl
```

```
c
      s1=at*xmb+bt*ymb+ct
      s2=at*xp+bt*yp+ct
      s3=as*xnt1+bs*ynt1+cs
      s4=as*xnt+bs*ynt+cs
c
      sig1=s1*s2
      sig2=s3*s4
c
      if(sig1.gt.0.0.or.sig2.gt.0.0) goto 2000
      iyon=0
      return
c
 2000 continue
c
      return
      end
c
c -----------------------------------------------------------------------
c
      subroutine smoot(ndimn,nnode,nelem,npoin,nsmoo,
     *                 lcoor,lcore,intmat,coord,coor0)
c
      dimension coord(2,*),coor0(2,*)
      dimension rdivn(30),x(6,2)
      dimension lcoor(*),lcore(*)
      dimension intmat(3,*),node(6)
c
      ndivn=30
c
      do 500 idivn=1,ndivn
      rdivn(idivn)=1./real(idivn)
  500 continue
c
c *** smooth out the grid in nsmoo steps
c
      if(nsmoo.eq.0) goto 10001
c
      do 10000 ismoo=1,nsmoo
c
c *** set coor0=0
c
      do 1200 ip=1,npoin
      do 1201 id=1,2
      coor0(id,ip)=0.0
1201  continue
1200  continue
c
c *** loop over the elements
c
      do 2000 ielem=1,nelem
      do 2100 ic=1,nnode
      in=intmat(ic,ielem)
      do 2101 id=1,ndimn
      x(      ic,id)=coord(id,in)
      x(nnode+ic,id)=coord(id,in)
 2101 continue
      node(ic)=in
 2100 continue
c
      do 2200 ic=1,nnode
      in=node(ic)
      if(lcoor(in).ne.0) goto 2199
      do 2300 jc=1,nnode-1
      do 2301 id=1,ndimn
      coor0(id,in)=coor0(id,in)+x(ic+jc,id)
 2301 continue
 2300 continue
 2199 continue
 2200 continue
c
 2000 continue
c
      do 3000 icoor=1,npoin
      if(lcoor(icoor).ne.0) goto 3000
      is=2*lcore(icoor)
      cn=rdivn(is)
```

```
      do 3100 id=1,ndimn
      coord(id,icoor)=cn*coor0(id,icoor)
 3100 continue
 3000 continue
c
10000 continue
10001 continue
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine order(nl,ipfr,iqfr,coor,disw,disw1)
c
      dimension ipfr(1),iqfr(1),coor(2,1)
c
      disw=1.e+6
c
      do 1000 il=1,nl
      kn=iqfr(il)
      kn1=ipfr(il)
      xn=coor(1,kn)
      yn=coor(2,kn)
      xn1=coor(1,kn1)
      yn1=coor(2,kn1)
      x12 = xn1-xn
      y12 = yn1-yn
      dis=sqrt(x12*x12+y12*y12)
      if(dis.lt.disw1) goto 1000
      if(dis.gt.disw) goto 1000
      iwh=il
      disw=dis
      iq=kn
      ip=kn1
 1000 continue
c
c *** swap values
c
      ipfr(iwh)=ipfr(nl)
      iqfr(iwh)=iqfr(nl)
      ipfr(nl)=ip
      iqfr(nl)=iq
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine bound(npoin,nelem,lcoor,intmat)
c
      dimension lcoor(npoin),intmat(3,nelem)
c
c *** this subroutine finds out the boundary points
c             lcoor(ipoin).eq.0 --> interior
c             lcoor(ipoin).ne.0 --> boundary
c
      do 1000 ip=1,npoin
      lcoor(ip)=0
 1000 continue
c
c *** loop over the elements
c
      do 2000 ie=1,nelem
      do 2001 in=1,3
      in1=in+1
      if(in1.gt.3) in1=in1-3
      in2=in+2
      if(in2.gt.3) in2=in2-3
      ip=intmat(in,ie)
      lcoor(ip)=lcoor(ip)+intmat(in2,ie)-intmat(in1,ie)
 2001 continue
 2000 continue
c
      return
      end
c
```

```
c ------------------------------------------------------------------------
c
      subroutine conre(npoin,nelem,intmat,lcore)
c
      dimension intmat(3,nelem),lcore(npoin)
c
      do 1000 ip=1,npoin
      lcore(ip)=0
 1000 continue
c
      do 2000 ie=1,nelem
      do 2001 in=1,3
      ip=intmat(in,ie)
      lcore(ip)=lcore(ip)+1
 2001 continue
 2000 continue
c
      return
      end
c
c ------------------------------------------------------------------------
c
      subroutine sides(nelem,npoin,iloca,intmat,iside,lwher,lhowm,
     *                 icone)
c
      dimension intmat(3,1),iside(4,1)
      dimension lwher(1),lhowm(1),icone(1)
c
c *** fill in lhowm : nr. of elements per node
c
      do 1490 ip=1,npoin
      lhowm(ip)=0
 1490 continue
      do 1500 ie=1,nelem
      do 1500 in=1,3
      ip=intmat(in,ie)
      lhowm(ip)=lhowm(ip)+1
 1500 continue
c
c ***  fill in lwher : location of each node inside icone
c
      lwher(1)=0
      do 1600 ip=2,npoin
      lwher(ip)=lwher(ip-1)+lhowm(ip-1)
 1600 continue
c
c ***  fill in icone : elements in each node
c
      do 1690 ip=1,npoin
      lhowm(ip)=0
 1690 continue
      do 1700 ie=1,nelem
      do 1701 in=1,3
      ip=intmat(in,ie)
      lhowm(ip)=lhowm(ip)+1
      jloca=lwher(ip)+lhowm(ip)
      icone(jloca)=ie
 1701 continue
 1700 continue
c
c *** loop over the nodes
c
      iloca=0
c
      do 3000 ip=1,npoin
      iloc1=iloca
      iele=lhowm(ip)
      if(iele.eq.0) goto 3000
c
c *** initialize iside ----> important for boundary sides
c
      do 3001 is=1,iele+2
      iside(3,is+iloc1)=0
      iside(4,is+iloc1)=0
 3001 continue
c
      iwher=lwher(ip)
```

```
c
c *** loop over elements surrounding the point ip
c
      ip1=ip
      do 3090 iel=1,iele
      ie=icone(iwher+iel)
c
c *** find out position of ip in the connectivity matrix
c
      do 3091 in=1,3
      in1=in
      ipt=intmat(in,ie)
      if(ipt.eq.ip) goto 3092
 3091 continue
 3092 continue
c
      do 3100 j=1,2
      in2=in1+j
      if(in2.gt.3) in2=in2-3
      ip2=intmat(in2,ie)

      if(ip2.lt.ip1) goto 3100
c
c *** check the side ----->  new or old
c
      if(iloca.eq.iloc1) goto 7304
      do 5600 is=iloc1+1,iloca
      jloca=is
      if(iside(2,is).eq.ip2) goto 7303
 5600 continue
 7304 continue
c
c *** new side
c
      iloca=iloca+1
      iside(1,iloca)=ip1
      iside(2,iloca)=ip2
      iside(2+j,iloca)=ie
      goto 3012
c
c *** old side
c
 7303 continue
      iside(2+j,jloca)=ie
 3012 continue
c
 3100 continue
c
 3090 continue
c
      do 8000 is=iloc1+1,iloca
      if(iside(3,is).ne.0) goto 8000
      iside(3,is)=iside(4,is)
      iside(4,is)=0
      iside(1,is)=iside(2,is)
      iside(2,is)=ip1
 8000 continue
c
 3000 continue
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine swapd(nside,iside,intmat,lcore,
     *                 lcoid,coord,nu,nv,r)
c
      dimension iside(4,*),intmat(3,*)
      dimension lcore(*),lcoid(*),coord(2,*)
      dimension r(12,*)
c
c *** this subroutine swaps the diagonals to obtain a more
c *** even distribution of elements per node
c
      deter(p1,q1,p2,q2,p3,q3)=p2*q3-p3*q2-p1*q3+p3*q1+p1*q2-p2*q1
c
```

```
 1000 ichan=0
c
      do 2000 is=1,nside
      i1=iside(1,is)
      i2=iside(2,is)
      ie1=iside(3,is)
      ie2=iside(4,is)
c
c ***  check for boundary sides
c
      if(ie2.eq.0) goto 2000
c
c *** determine i3 & i4
c
      do 5000 in=1,3
      in1=in+1
      if(in1.gt.3) in1=in1-3
      in2=in+2
      if(in2.gt.3) in2=in2-3
      ip11=intmat(in,ie1)
      ip12=intmat(in1,ie1)
      ip13=intmat(in2,ie1)
      if(ip11.eq.i1.and.ip12.eq.i2) i3=ip13
      ip21=intmat(in,ie2)
      ip22=intmat(in1,ie2)
      ip23=intmat(in2,ie2)
      if(ip21.eq.i2.and.ip22.eq.i1) i4=ip23
 5000 continue
c
c *** find 'deficit' or 'superhavit' of connectivities
c
      ih1=abs(lcore(i1)-lcoid(i1))
      ih2=abs(lcore(i2)-lcoid(i2))
      ih3=abs(lcore(i3)-lcoid(i3))
      ih4=abs(lcore(i4)-lcoid(i4))
      ihf1=abs(lcore(i1)-1-lcoid(i1))
      ihf2=abs(lcore(i2)-1-lcoid(i2))
      ihf3=abs(lcore(i3)+1-lcoid(i3))
      ihf4=abs(lcore(i4)+1-lcoid(i4))
c
c *** check if it is worth to swap
c
      iswap=0
      iactu=ih1+ih2+ih3+ih4
      ifutu=ihf1+ihf2+ihf3+ihf4
      if(iactu.gt.ifutu) iswap=1
      iam=max(ih1,ih2,ih3,ih4)
      ifm=max(ihf1,ihf2,ihf3,ihf4)
      if(iactu.eq.ifutu.and.iam.gt.(ifm+1)) iswap=1
      if(iswap.eq.0) goto 2000
c
c *** check area
c
      u1=coord(1,i1)
      v1=coord(2,i1)
      u3=coord(1,i3)
      v3=coord(2,i3)
      u4=coord(1,i4)
      v4=coord(2,i4)
      if(deter(u1,v1,u4,v4,u3,v3).le.0.0) goto 2000
      u2=coord(1,i2)
      v2=coord(2,i2)
      if(deter(u3,v3,u4,v4,u2,v2).le.0.0) goto 2000
c
c *** curved surfaces: an additional checking
c
      call getpt(u1,v1,nu,nv,r,x1,y1,z1)
      call getpt(u2,v2,nu,nv,r,x2,y2,z2)
      call getpt(u3,v3,nu,nv,r,x3,y3,z3)
      call getpt(u4,v4,nu,nv,r,x4,y4,z4)
      a11 = u2-u1
      a12 = u3-u4
      a21 = v2-v1
      a22 = v3-v4
      b1 = u3-u1
      b2 = v3-v1
      dt = 1.0/(a11*a22-a12*a21)
```

```
      if(abs(dt).lt.1.e-06) print *,' error: wrong dt'
      f1 = dt*( a22*b1-a12*b2)
      f2 = dt*(-a21*b1+a11*b2)
      if(f1.lt.0.0.or.f1.gt.1.0) print *,' error: wrong f1'
      if(f2.lt.0.0.or.f2.gt.1.0) print *,' error: wrong f2'
      u = u1+f1*a11
      v = v1+f1*a21
      call getpt(u,v,nu,nv,r,xp,yp,zp)
      x12 = x1+f1*(x2-x1)
      y12 = y1+f1*(y2-y1)
      z12 = z1+f1*(z2-z1)
      x34 = x3+f2*(x4-x3)
      y34 = y3+f2*(y4-y3)
      z34 = z3+f2*(z4-z3)
      d12 = (x12-xp)**2+(y12-yp)**2+(z12-zp)**2
      d34 = (x34-xp)**2+(y34-yp)**2+(z34-zp)**2
      if(d34.gt.d12) goto 2000
c
c *** swap
c
      ichan=ichan+1
      intmat(1,ie1)=i1
      intmat(2,ie1)=i4
      intmat(3,ie1)=i3
      intmat(1,ie2)=i3
      intmat(2,ie2)=i4
      intmat(3,ie2)=i2
      lcore(i1)=lcore(i1)-1
      lcore(i2)=lcore(i2)-1
      lcore(i3)=lcore(i3)+1
      lcore(i4)=lcore(i4)+1
c
c *** detect the sides i1-i4 and i3-i2
c
      ist1=0
      ist2=0
      do 4000 ist=1,nside
      i1t=iside(1,ist)
      i2t=iside(2,ist)
      if((i1t.eq.i1.and.i2t.eq.i4).or.(i1t.eq.i4.and.i2t.eq.i1))
     *            ist1=ist
      if((i1t.eq.i3.and.i2t.eq.i2).or.(i1t.eq.i2.and.i2t.eq.i3))
     *            ist2=ist
      if(ist1.ne.0.and.ist2.ne.0) goto 4001
 4000 continue
      print *,' error in swapdi '
      stop
 4001 continue
      if(iside(3,ist1).eq.ie2) iside(3,ist1)=ie1
      if(iside(4,ist1).eq.ie2) iside(4,ist1)=ie1
      if(iside(3,ist2).eq.ie1) iside(3,ist2)=ie2
      if(iside(4,ist2).eq.ie1) iside(4,ist2)=ie2
c
c *** update iside
c
      iside(1,is)=i4
      iside(2,is)=i3
c
 2000 continue
c
      print 1002,ichan
 1002 format(i6,' sides have been swapped')
c
      if(ichan.ge.1) goto 1000
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine getpt(u,v,nu,nv,r,x,y,z)
c
      parameter (nd4=12)
      dimension r(nd4,1),r1(nd4),r2(nd4),r3(nd4),r4(nd4),r5(nd4)
c
      iu = int(u)
      iv = int(v)
```

```
      if(iu.eq.(nu-1)) iu = iu-1
      if(iv.eq.(nv-1)) iv = iv-1
      ul = u-iu
      vl = v-iv
      ip1 = iu+iv*nu+1
      ip2 = ip1+1
      ip3 = ip1+nu
      ip4 = ip3+1
      do 100 id=1,nd4
      r1(id) = r(id,ip1)
      r2(id) = r(id,ip2)
      r3(id) = r(id,ip3)
      r4(id) = r(id,ip4)
  100 continue
      call fgsurf(1,r1,r2,r3,r4,ul,vl,r5)
      x = r5(1)
      y = r5(2)
      z = r5(3)
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine eattr(npoin,nelem,nside,intmat,iside,lcoor,lposi,
     *                 lwher,lhowm,lcore,lcoid,icone,coord)
c
      parameter(mxpoi=40000)
      dimension intmat(3,1),iside(4,1),lcoor(1)
      dimension lposi(1),lwher(1),lhowm(1)
      dimension lcore(1),lcoid(1),icone(1)
      dimension coord(2,1),leat(mxpoi)
c
c *** this subroutine removes the points where only three
c *** elements coincide
c
      if(mxpoi.le.npoin) stop ' increase dimensions in eat 3s '
      do 1988 i=1,npoin
1988  leat(i) =0
c
      ntres=0
      kpoin=0
c
c *** loop over number of nodes
c
      do 1000 ip=1,npoin
      kpoin=kpoin+1
      lposi(ip)=kpoin
c
c *** if it's already been eaten leave it
c
      if(leat(ip).ne.0) goto 1000
c
c *** if boundary point leave it
c
      if(lcoor(ip).ne.0) goto 1000
c
c *** check number of elements
c
      if(lcore(ip).ne.3) goto 1000
c
c *** a point with only three elements
c
      ntres=ntres+1
      kpoin=kpoin-1
      lposi(ip)=0
c
c *** get the elements from icone
c
      iloca=lwher(ip)
      ie1=icone(iloca+1)
      ie2=icone(iloca+2)
      ie3=icone(iloca+3)
c
c *** get new connectivity point for ie1 from ie2
c
      ip1=intmat(1,ie1)
      ip2=intmat(2,ie1)
```

```
      ip3=intmat(3,ie1)
c
      do 1002 in=1,3
      ipt=intmat(in,ie2)
      if(ipt.ne.ip1.and.ipt.ne.ip2.and.ipt.ne.ip3) ino=ipt
 1002 continue
c
c *** replace connectivity
c
      do 1003 in=1,3
      if(intmat(in,ie1).eq.ip) intmat(in,ie1)=ino
      lcore(intmat(in,ie1))=lcore(intmat(in,ie1))-1
      leat(intmat(in,ie1))=1
 1003 continue
c
      do 1004 in=1,3
      intmat(in,ie2)=0
      intmat(in,ie3)=0
 1004 continue
c
 1000 continue
c
c *** transfer
c
      do 2000 ip=1,npoin
      il=lposi(ip)
      if(il.eq.0) goto 2000
      lcoor(il)=lcoor(ip)
      lcore(il)=lcore(ip)
      lcoid(il)=lcoid(ip)
      do 2001 id=1,2
      coord(id,il)=coord(id,ip)
 2001 continue
 2000 continue
c
      je=0
      do 3000 ie=1,nelem
      if(intmat(1,ie).eq.0) goto 3000
      je=je+1
      do 3001 in=1,3
      iold=intmat(in,ie)
      inew=lposi(iold)
      intmat(in,je)=inew
 3001 continue
 3000 continue
c
c *** get npoin and nelem
c
      npoin=npoin-ntres
      nelem=nelem-2*ntres
c
c *** output nr. of eaten points
c
      print *,' *** nr. of 3"s removed = ',ntres
c
c *** fill in iside
c
      call sides(nelem,npoin,nside,intmat,iside,lwher,lhowm,icone)
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine areach(npoin,nelem,intmat,coord)
c
      dimension coord(2,npoin),intmat(3,nelem)
c
c *** determinant function
c
      deter(p1,q1,p2,q2,p3,q3)=p2*q3-p3*q2-p1*q3+p3*q1+p1*q2-p2*q1
c
      isucc=0
      write(9,888)  nelem
  888 format(i12)
      do 1000 ie=1,nelem
      i1=intmat(1,ie)
```

```
      i2=intmat(2,ie)
      i3=intmat(3,ie)
      write(9,999)  ie, i1, i2, i3
  999 format(4i8)
      x1=coord(1,i1)
      x2=coord(1,i2)
      x3=coord(1,i3)
      y1=coord(2,i1)
      y2=coord(2,i2)
      y3=coord(2,i3)
c
      if(deter(x1,y1,x2,y2,x3,y3).gt.0.0) goto 1000
c
      print 1002,ie,i1,x1,y1,i2,x2,y2,i3,x3,y3
 1002 format('  area error in element ',i5,/,
     *       '  node 1 =',i5,'  x1 =',f10.5,'  y1 =',f10.5,/,
     *       '  node 2 =',i5,'  x2 =',f10.5,'  y2 =',f10.5,/,
     3       '  node 3 =',i5,'  x3 =',f10.5,'  y3 =',f10.5,/)
c
      isucc=1
c
 1000 continue
c
      if(isucc.eq.0) print *,'  the checking has been succesful !!!!'
c
c *** output number of nodes and elements
c
      print 228,npoin,nelem
  228 format(' total number of generated points :',i5,/
     *       ' total number of generated elements :',i5)
c
      return
      end
c
c -------------------------------------------------------------------------
c
      subroutine rfiliv(ia,n,ival)
c
      dimension ia(n)
c
      do 1000 i=1,n
      ia(i)=ival
 1000 continue
c
      return
      end
c*--------------------------------------------------------------------------*
c*    [glnum] generates the global mesh from individual mesh information    *
c*--------------------------------------------------------------------------*
      subroutine glnum(nis ,nsf ,np  ,ne  ,lpnp,conp,kpnp,kpne,
     .                 cosf,lmsf,lpbs,isbs,lpsp,cpsp,nsp1,nsp2,
     .                 cost,lmst,nwed,nwfa)
c
      parameter (naux=20000)
      dimension lpnp(*),conp(3,*),lpbs(*),isbs(*)
      dimension kpnp(*),kpne(*),cosf(2,*),lmsf(3,*)
      dimension cost(3,*),lmst(4,*),nwed(*),nwfa(*)
      dimension lpsp(*),cpsp(3,*),nsp1(*),nsp2(*)
      dimension r1(12,naux),r1(12),r2(12),r3(12),r4(12),r5(12)
      dimension kpip(naux)
c
c *** eps is the tolerance for identifying the corner points.
c     the check is performed in terms of the 3D coordinates of
c     the edges defined by user. therefore it depends on the
c     accuracy of the given definition of the edges.
c
      eps = 1.e-05
c
c *** first we deal with the vector containing the points
c     generated on the edges of the surface. the new numbering
c     is stored in the vector nwed.
c
      npge = kpnp(nis+1)-1
      do 100 ip=1,npge
      nwed(ip) = 0
  100 continue
c
```

```
c *** identifies the corner points by looping over the edges,
c     starts the numbering and stores the 3D coordinates
c
      np = 0
      do 300 is=1,nis
      ip1 = lpnp(is)
      ip2 = lpnp(is+1)-1
      xp1 = conp(1,ip1)
      yp1 = conp(2,ip1)
      zp1 = conp(3,ip1)
      xp2 = conp(1,ip2)
      yp2 = conp(2,ip2)
      zp2 = conp(3,ip2)
c
c *** checks if the end points have been included in the list
c
      inw1 = 1
      inw2 = 1
      if(np.ne.0) then
        do 200 ip=1,np
        xdt1 = xp1-cost(1,ip)
        ydt1 = yp1-cost(2,ip)
        zdt1 = zp1-cost(3,ip)
        dst1 = sqrt(xdt1*xdt1+ydt1*ydt1+zdt1*zdt1)
        if(dst1.lt.eps) then
          inw1 = 0
          nwed(ip1) = ip
        endif
        xdt2 = xp2-cost(1,ip)
        ydt2 = yp2-cost(2,ip)
        zdt2 = zp2-cost(3,ip)
        dst2 = sqrt(xdt2*xdt2+ydt2*ydt2+zdt2*zdt2)
        if(dst2.lt.eps) then
          inw2 = 0
          nwed(ip2) = ip
        endif
  200   continue
      endif
      if(inw1.eq.1) then
        np = np+1
        cost(1,np) = xp1
        cost(2,np) = yp1
        cost(3,np) = zp1
        nwed(ip1) = np
      endif
      if(inw2.eq.1) then
        np = np+1
        cost(1,np) = xp2
        cost(2,np) = yp2
        cost(3,np) = zp2
        nwed(ip2) = np
      endif
  300 continue
c
c *** numbers the rest of the points on the edges
c
      do 500 is=1,nis
      ip1 = lpnp(is)+1
      ip2 = lpnp(is+1)-2
      do 400 ip=ip1,ip2
      np = np+1
      cost(1,np) = conp(1,ip)
      cost(2,np) = conp(2,ip)
      cost(3,np) = conp(3,ip)
      nwed(ip)  = np
  400 continue
  500 continue
c
c *** now we deal with the vector containing the points
c     generated on the faces of the surface. the new numbering
c     is stored in the vector nwed.
c
      npgf = kpne(nsf+1)-1
      do 600 ip=1,npgf
      nwfa(ip) = 0
  600 continue
c
```

```
c *** finds the global number. this is based on the procedure
c     used to form the initial front for the faces.
c     the edges are ordered and the faces are closed, thus
c     the end points of the edge are not taken into account
c     notation:
c     kpip .... is a pointer of the interior points on a face
c
      do 900 is=1,nsf
      ks1 = lpbs(is)
      ks2 = lpbs(is+1)-1
      jpk = kpnp(is)-1
      do 800 ik=ks1,ks2
      ked = isbs(ik)
      kab = abs(ked)
      if(ked.gt.0) then
        ip1 = lpnp(kab)
        ip2 = lpnp(kab+1)-1
        ikn = 1
      else
        ip1 = lpnp(kab+1)-1
        ip2 = lpnp(kab)
        ikn = -1
      endif
      do 700 ip=ip1,ip2-ikn,ikn
      jpk = jpk+1
      nwfa(jpk) = nwed(ip)
  700 continue
  800 continue
      kpip(is) = jpk+1
  900 continue
c
c *** finally numbers the interior points on the faces
c     and computes their 3D coordinates.
c
      do 1300 is=1,nsf
      npu = nsp1(is)
      npv = nsp2(is)
      n1  = lpsp(is)
      n2  = lpsp(is+1)-1
      ip1 = kpip(is)
      ip2 = kpnp(is+1)
      if(ip1.eq.ip2) goto 1300
      kp  = 0
      do 1000 ip=n1,n2
      kp = kp+1
      rl(1,kp) = cpsp(1,ip)
      rl(2,kp) = cpsp(2,ip)
      rl(3,kp) = cpsp(3,ip)
 1000 continue
      call sfgeo(npu,npv,rl)
      do 1200 ip=ip1,ip2-1
      u  = cosf(1,ip)
      v  = cosf(2,ip)
      iu = int(u)
      iv = int(v)
      if(iu.eq.(npu-1)) iu = iu-1
      if(iv.eq.(npv-1)) iv = iv-1
      ul = u-iu
      vl = v-iv
      ip1 = iu+iv*npu+1
      ip2 = ip1+1
      ip3 = ip1+npu
      ip4 = ip3+1
      do 1100 id = 1,12
      r1(id) = rl(id,ip1)
      r2(id) = rl(id,ip2)
      r3(id) = rl(id,ip3)
      r4(id) = rl(id,ip4)
 1100 continue
      call fgsurf(0,r1,r2,r3,r4,ul,vl,r5)
      np = np+1
      nwfa(ip) = np
      cost(1,np) = r5(1)
      cost(2,np) = r5(2)
      cost(3,np) = r5(3)
 1200 continue
 1300 continue
```

```
c
c *** gets the connectivity array for the surface triangulation.
c     the local number of the points which appears in the array
c     lmsf ranges form 1 to the number of points in the face.
c
      ne = kpne(nsf+1)-1
c
      do 1500 is=1,nsf
      ie1 = kpne(is)
      ie2 = kpne(is+1)-1
      do 1400 ie=ie1,ie2
      ip1 = lmsf(1,ie)
      ip2 = lmsf(2,ie)
      ip3 = lmsf(3,ie)
      lmst(1,ie) = nwfa(ip1)
      lmst(2,ie) = nwfa(ip2)
      lmst(3,ie) = nwfa(ip3)
      lmst(4,ie) = is
 1400 continue
 1500 continue
c
      return
      end
c*--------------------------------------------*
c*    [outbg] writes the new triangulation    *
c*--------------------------------------------*
      subroutine outbg(iou,np,ne,cost,lmst)
      dimension cost(3,*),lmst(4,*)
c
      write(*,10)
c
      write(iou,*) np,ne
      do 100 ip=1,np
      write(iou,*) ip,(cost(in,ip),in=1,3)
  100 continue
      do 200 ie=1,ne
      write(iou,*) ie,(lmst(in,ie),in=1,4)
  200 continue
c ...
   10 format(/,'  facet > writing surface triangulation')
c
      return
      end
c*-----------------------------------------------------------*
c*    [outsg] writes the edge information on the new mesh    *
c*-----------------------------------------------------------*
      subroutine outsg(iou,nis,lpnp,spnp,nwed)
      dimension lpnp(*),spnp(*),nwed(*)
c
      write(*,10)
c
      do 200 is=1,nis
      ip1 = lpnp(is)
      ip2 = lpnp(is+1)-1
      np = ip2-ip1+1
      write(iou,*) np
      do 100 ip=ip1,ip2
      write(iou,*) nwed(ip),spnp(ip)
  100 continue
  200 continue
c ...
   10 format(/,'  facet > writing edge information')
c
      return
      end
c*-----------------------------------------------------------*
c*    [outsf] writes the face information on the new mesh    *
c*-----------------------------------------------------------*
      subroutine outsf(iou,nsf,kpnp,cosf,nwfa)
      dimension kpnp(*),cosf(2,*),nwfa(*)
c
      write(*,10)
c
      do 200 is=1,nsf
      ip1 = kpnp(is)
      ip2 = kpnp(is+1)-1
      np = ip2-ip1+1
```

```
      write(iou,*) np
      write(8,888) np
  888 format(i12)
      nodloc = 1
      do 100 ip=ip1,ip2
      write(iou,*)         nwfa(ip), (cosf(in,ip),in=1,2)
      write(8,889) nodloc, nwfa(ip), (cosf(in,ip),in=1,2)
  889 format(2i8, 2e30.14)
      nodloc = nodloc + 1
  100 continue
  200 continue
c ...
   10 format(/,'  facet > writing face information')
c
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine size(iou,nsf,cpsp)
c
c *** this make <filnam>.dim to create <filnam>.re_ in SPACE PROGRAM
c
      dimension cpsp(3,*)
c
      k = 1
      l = 4
      do 100 isf = 1,nsf
      write(iou,*) isf
      d1 = abs(cpsp(1,l)-cpsp(1,k))
      d2 = abs(cpsp(2,l)-cpsp(2,k))
      write(iou,*) d1,d2
      k = k+4
      l = l+4
  100 continue
      close(iou)
      return
      end
c
c ---------------------------------------------------------------------
c
      subroutine datagraphic(iou1,iou2,np,ne,cost,lmst)
c
c *** this make datafile to draw graphic in FEPLOT PROGRAM and NASTRAN
c
      dimension cost(3,*),lmst(4,*)
c
      nc = 0
      nd = 1
      nvar = 1
      write(iou1,10)
   10 format('         npoin     nelem     nvar ')
      write(iou1,*) np,ne,nvar
      write(iou1,20)
   20 format('  number of node ')
      do 100 in = 1,np
      write(iou1,30) in,(cost(i,in),i=1,3)
   30 format(1x,i3,3x,3(f10.6,3x))
      write(iou2,35) in,nc,(cost(i,in),i=1,3),nc
   35 format('GRID',7x,i5,7x,i1,3(f8.5),7x,i1)
  100 continue
      write(iou1,40)
   40 format('  number of element ')
      do 200 ie = 1,ne
      write(iou1,*) ie,(lmst(i,ie),i=1,3)
      write(iou2,45) ie,nd,(lmst(i,ie),i=1,3)
   45 format('CTRIA3',5x,i5,7x,i1,3(3x,i5))
  200 continue
      close(iou1)
      close(iou2)
      return
      end
c
c ---------------------------------------------------------------------
c
```

ภาคผนวก ค.

รายละเอียดโปรแกรม SPACE

รายละเอียดโปรแกรม SPACE จะมีรายละเอียดเริ่มจากโปรแกรมหลัก และตามด้วยโปรแกรมย่อยต่าง ๆ ดังนี้

```
C     PROGRAM SPACE
C
C     A PROGRAM TO COMPUTE NODAL SPACINGS FOR CONSTRUCTING NEW ADAPTIVE
C     MESH FOR 3-D BUILT-UP STRUCTURES.
C
      PARAMETER  (MXPOI=10000)
      PARAMETER  (MXNOD= 5000, MXELE= 5000)
C
C     MXPOI IS THE MAX. NO. OF NODES ALLOWED FOR THE ENTIRE MODEL
C     MXNOD & MXELE ARE MAX NO. OF NODES & ELEMENTS ALLOWED ON A FACE
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION  VG(MXPOI), VL(MXNOD), DELTA(MXPOI)
      DIMENSION  X(MXNOD), Y(MXNOD)
      DIMENSION  EIGEN(MXNOD), ONE(MXNOD), EIGBL(MXPOI)
      DIMENSION  DDX(MXNOD), DDY(MXNOD)
      DIMENSION  DDXX(MXNOD), DDYY(MXNOD), DDXY(MXNOD)
C
      INTEGER  INTMAT(MXELE,3), NGBL(MXNOD)
      CHARACTER FILNAM*12,CV*4
C
C     OPEN INPUT/OUTPUT FILES
C
      WRITE(*,*) ' CHECK INPUT FILES FOR  1. ?.N_'
      WRITE(*,*) '                        2. ?.L_'
      WRITE(*,*) '                        3. ?.DIM'
      WRITE(*,*) '                        4. ?.V_'
      WRITE(*,'(/,A,$)') ' ENTER PROBLEM NAME : '
      READ(*,'(A)') FILNAM
      WRITE(*,'(A,$)') ' ENTER CURRENT VERSION NUMBER : '
      READ(*,'(A)') CV
      L = NAMLEN(FILNAM)
      IF (L.EQ.0) GO TO 333
      OPEN(UNIT= 8, FILE= FILNAM(1:L)//'.N'//CV, STATUS='OLD',ERR= 333)
      OPEN(UNIT= 9, FILE= FILNAM(1:L)//'.L'//CV, STATUS='OLD',ERR= 333)
      OPEN(UNIT=10, FILE= FILNAM(1:L)//'.DIM',    STATUS='OLD',ERR= 333)
      OPEN(UNIT=12, FILE= FILNAM(1:L)//'.V'//CV, STATUS='OLD',ERR= 333)
      OPEN(UNIT=13, FILE= FILNAM(1:L)//'.R'//CV, STATUS='NEW',ERR= 333)
C
      DO 5  I=1,MXPOI
      DELTA(I) = 0.
      EIGBL(I) = 0.
    5 CONTINUE
C
C     READ NODAL VON MISES STRESSES FOR THE ENTIRE MODEL:
C
      READ(12,*)  NPOIG
      DO 10  I=1,NPOIG
      READ(12,*)  N, VG(I)
      IF(I.NE.N)  WRITE(6,15)
   15 FORMAT(' ** CHECK VON MISES STRESS DATA **  NODES NOT IN ORDER')
   10 CONTINUE
C
C     LOOP OVER THE NUMBER OF FACES:
C
```

```
      READ(10,*)  NFACES
      DO 1000  NF=1,NFACES
C
C     READ ACTUAL FACE DIMENSIONS:
C     (CURRENTLY RESTRICTED TO RECTANGULAR SHAPE, CAN GENERALIZE LATER)
C
      READ(10,*)  AA, BB
C
C     READ LOCAL & GLOBAL NODE NUMBERS AND THEIR LOCAL X-Y COORDINATES
C     IN THE PARAMETER PLANE (0. TO 1.) AND CONVERT TO THE ACTUAL
C     X-Y COORDINATES:
C
      DO 80  I=1,MXNOD
      NGBL(I) = -999
      X(I) = 0.
      Y(I) = 0.
      VL(I)= 0.
   80 CONTINUE
C
      READ(8,*)  NPOIN
      IF(NPOIN.GT.MXNOD)  WRITE(6,103)  NPOIN
  103 FORMAT(' INCREASE  MXNOD  TO', I5)
      IF(NPOIN.GT.MXNOD)  STOP
      DO 100  I=1,NPOIN
      READ(8,*)  N, NGBL(I), XX, YY
      IF(I.NE.N)  WRITE(6,105)  NF
  105 FORMAT(' ** ERROR ** CHECK LOCAL NODE NO. ON FACE', I5)
      X(I) = XX*AA
      Y(I) = YY*BB
  100 CONTINUE
C
C     READ LOCAL ELEMENT NODAL CONNECTIVITIES:
C
      DO 110  I=1,MXELE
      DO 110  J=1,3
      INTMAT(I,J) = -888
  110 CONTINUE
C
      READ(9,*)  NELEM
      IF(NELEM.GT.MXELE)  WRITE(6,123)  NELEM
  123 FORMAT(' INCREASE  MXELE  TO', I5)
      IF(NELEM.GT.MXELE)  STOP
      DO 120  IE=1,NELEM
      READ(9,*)  N, (INTMAT(IE,J), J=1,3)
      IF(IE.NE.N)  WRITE(6,125)  NF
  125 FORMAT(' ** ERROR ** CHECK LOCAL ELEMENT NO. ON FACE', I5)
  120 CONTINUE
C
C     OBTAIN LOCAL NODAL VON MISES STRESSES FOR THIS FACE:
C
      DO 130  I=1,NPOIN
      N = NGBL(I)
      VL(I) = VG(N)
  130 CONTINUE
C
C     COMPUTE EIGENVALUES FOR ALL THE NODES ON THIS FACE:
C
      CALL EGVALUE(MXNOD, MXELE, NPOIN, NELEM, X, Y, INTMAT,
     &             VL, ONE, DDX, DDY, DDXX, DDYY, DDXY, EIGEN)
C
C     OBTAIN NODAL EIGENVALUES FOR THE GLOBAL MODEL:
C
      DO 140  I=1,NPOIN
      N = NGBL(I)
      EINEW = EIGEN(I)
      IF(EINEW.GT.EIGBL(N))  EIGBL(N) = EINEW
  140 CONTINUE
C
 1000 CONTINUE
C
C     THEN COMPUTE THE PROPER NODAL SPACINGS FOR THE ENTIRE MODEL:
C
      WRITE(6,150)
  150 FORMAT(' PLEASE INPUT THE MINIMUM & MAXIMUM SPACINGS', /)
      READ(5,*)  HMIN, HMAX
C
C     FIND THE MAXIMUM EIGENVALUE OF THE ENTIRE MODEL:
```

```
C
      EGMAX = 0.
      DO 200  I=1,NPOIG
      IF(EIGBL(I).GT.EGMAX)  EGMAX = EIGBL(I)
  200 CONTINUE
C
C     POSSIBILITY OF ZERO EIGENVALUE:
C
      TOL = 1.E-14
      DO 250  I=1,NPOIG
      IF(EIGBL(I).LT.TOL)  EIGBL(I) = TOL
  250 CONTINUE
C
      WRITE(13,305)  NPOIG
      WRITE( 6,305)  NPOIG
  305 FORMAT(I12)
      DO 300  I=1,NPOIG
      H2 = HMIN*HMIN*EGMAX/EIGBL(I)
      DELTA(I) = SQRT(H2)
      IF(DELTA(I).GT.HMAX)  DELTA(I) = HMAX
      WRITE(13,310)  I, DELTA(I)
      WRITE( 6,310)  I, DELTA(I)
  310 FORMAT(I8, E20.8)
  300 CONTINUE
C
      GO TO 344
  333 WRITE(*,*)' *** FILE NAME NOT EXIST *** '
  344 STOP
      END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE EGVALUE(MXNOD, MXELE, NPOIN, NELEM, X, Y, INTMAT,
     *                   VL, ONE, DDX, DDY, DDXX, DDYY, DDXY, EIGEN)
C
C     COMPUTE NODAL EIGENVALUES FOR EACH FACE
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION  X(MXNOD), Y(MXNOD)
      DIMENSION  VL(MXNOD), EIGEN(MXNOD)
      DIMENSION  ONE(MXNOD), DDX(MXNOD), DDY(MXNOD)
      DIMENSION  DDXX(MXNOD), DDYY(MXNOD), DDXY(MXNOD)
C
      INTEGER  INTMAT(MXELE,3)
C
      DO 10  I=1,NPOIN
      EIGEN(I) = 0.
      ONE(I)   = 0.
      DDX(I)   = 0.
      DDY(I)   = 0.
      DDXX(I)  = 0.
      DDYY(I)  = 0.
      DDXY(I)  = 0.
   10 CONTINUE
C
C     LOOP OVER NO. OF ELEMENTS ON THIS FACE:
C
      DO 1000  IE=1,NELEM
C
      II = INTMAT(IE,1)
      JJ = INTMAT(IE,2)
      KK = INTMAT(IE,3)
C
      B1 = Y(JJ) - Y(KK)
      B2 = Y(KK) - Y(II)
      B3 = Y(II) - Y(JJ)
      C1 = X(KK) - X(JJ)
      C2 = X(II) - X(KK)
      C3 = X(JJ) - X(II)
      AREA = 0.5*(B2*X(JJ) + B1*X(II) + B3*X(KK))
C
C     COMPUTE THE FIRST DERIVATIVES FOR THIS ELEMENT:
C
      V1 = VL(II)
      V2 = VL(JJ)
      V3 = VL(KK)
      DVDX = (B1*V1 + B2*V2 + B3*V3)/(2.*AREA)
```

```
      DVDY = (C1*V1 + C2*V2 + C3*V3)/(2.*AREA)
C
C     CONTRIBUTIONS TO NODAL QUANTITIES:
C
      DO 100  J=1,3
      IN = INTMAT(IE,J)
      DDX(IN) = DDX(IN) + DVDX
      DDY(IN) = DDY(IN) + DVDY
      ONE(IN) = ONE(IN) + 1.
  100 CONTINUE
C
 1000 CONTINUE
C
C     OBTAIN AVERAGE NODAL FIRST DERIVATIVES:
C
      DO 200  I=1,NPOIN
      IF(ONE(I).EQ.0.)  WRITE(6,210)  I
  210 FORMAT(' NO FIRST DRIVATIVE CONTRIBUTION FOR LOCAL NODE', I5)
      IF(ONE(I).EQ.0.)  ONE(I) = 1.
      DDX(I) = DDX(I)/ONE(I)
      DDY(I) = DDY(I)/ONE(I)
  200 CONTINUE
C
C     LOOP OVER NO. OF ELEMENTS ON THIS FACE AGAIN TO COMPUTE
C     THE SECOND DERIVATIVES:
C
      DO 2000  IE=1,NELEM
C
      II = INTMAT(IE,1)
      JJ = INTMAT(IE,2)
      KK = INTMAT(IE,3)
C
      B1 = Y(JJ) - Y(KK)
      B2 = Y(KK) - Y(II)
      B3 = Y(II) - Y(JJ)
      C1 = X(KK) - X(JJ)
      C2 = X(II) - X(KK)
      C3 = X(JJ) - X(II)
      AREA = 0.5*(B2*X(JJ) + B1*X(II) + B3*X(KK))
C
C     COMPUTE SECOND DERIVATIVES WRT. X & Y:
C
      P1    = DDX(II)
      P2    = DDX(JJ)
      P3    = DDX(KK)
      DVDXX = (B1*P1 + B2*P2 + B3*P3)/(2.*AREA)
      DVDXY = (C1*P1 + C2*P2 + C3*P3)/(2.*AREA)
C
      Q1    = DDY(II)
      Q2    = DDY(JJ)
      Q3    = DDY(KK)
      DVDYX = (B1*Q1 + B2*Q2 + B3*Q3)/(2.*AREA)
      DVDYY = (C1*Q1 + C2*Q2 + C3*Q3)/(2.*AREA)
C
C     CONTRIBUTIONS TO NODAL QUANTITIES:
C
      DO 300  J=1,3
      IN = INTMAT(IE,J)
      DDXX(IN) = DDXX(IN) + DVDXX
      DDYY(IN) = DDYY(IN) + DVDYY
      DDXY(IN) = DDXY(IN) + 0.5*(DVDXY + DVDYX)
  300 CONTINUE
C
 2000 CONTINUE
C
C     OBTAIN AVERAGE NODAL SECOND DERIVATIVES:
C
      DO 400  I=1,NPOIN
      DDXX(I) = DDXX(I)/ONE(I)
      DDYY(I) = DDYY(I)/ONE(I)
      DDXY(I) = DDXY(I)/ONE(I)
  400 CONTINUE
C
C     OBTAIN THE TWO PRINCIPLE EIGENVALUES:
C
      DO 500  I=1,NPOIN
      SXX   = DDXX(I)
```

```
      SYY    = DDYY(I)
      SXY    = DDXY(I)
      AVG    = (SXX+SYY)/2.
      DIF    = (SXX-SYY)/2.
      TERM2 = DIF*DIF + SXY*SXY
      TERM2 = SQRT(TERM2)
      EG1    = AVG + TERM2
      EG2    = AVG - TERM2
      EG1    = ABS(EG1)
      EG2    = ABS(EG2)
      EIGEN(I) = EG1
      IF(EG2.GT.EG1)  EIGEN(I) = EG2
  500 CONTINUE
C
      RETURN
      END
C
C*--------------------------------------------------------------------*
C*    [NAMLEN] COUNTS THE NUMBER OF CHARACTERS IN FILNAM              *
C*--------------------------------------------------------------------*
      INTEGER FUNCTION NAMLEN(FILNAM)
C
      CHARACTER*12 FILNAM
C
      NAMLEN = 0
      DO 10 I = 12,1,-1
      IF (FILNAM(I:I).EQ.' ') GO TO 10
      NAMLEN = I
      GO TO 20
   10 CONTINUE
   20 RETURN
      END
```

ประวัติผู้วิจัย

นายสุพัฒนพงศ์ สิกขาบัณฑิต เกิดเมื่อวันที่ 6 เดือนกุมภาพันธ์ พุทธศักราช 2516 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมเครื่องกล ภาควิชาวิศวกรรมเครื่องกล คณะวิศวกรรมศาสตร์ จากสถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ เมื่อปีการศึกษา 2537 เข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต ภาควิชาวิศวกรรมเครื่องกล คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2539 ปัจจุบันเป็นวิศวกร ระดับ 5 แผนกวิทยาการ กองพัฒนาพลังงานทดแทน สำนักงานวิจัยและพัฒนา การไฟฟ้าฝ่ายผลิตแห่งประเทศไทย