LBL-29925
UC-404

Lawrence Berkeley Laboratory

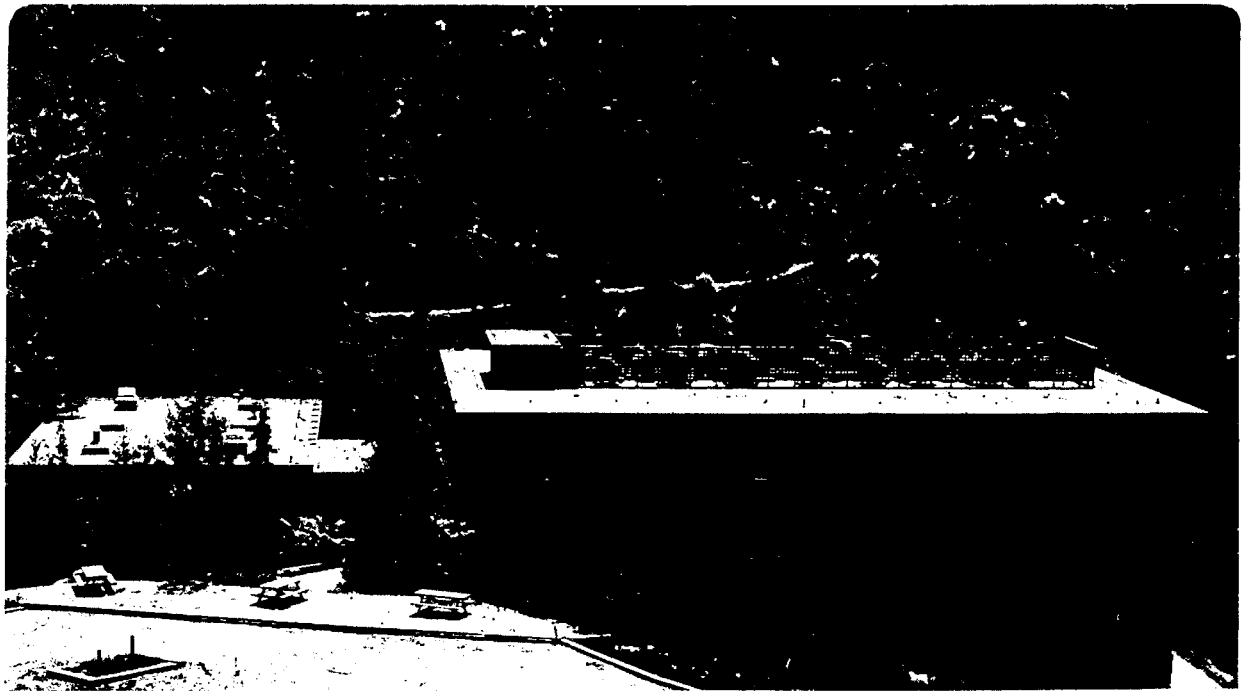
UNIVERSITY OF CALIFORNIA

Materials & Chemical Sciences Division

Computer Programs for Simulation of Electrodeposition

K.G. Jordan and C.W. Tobias

December 1990



LOAN COPY
Circulates
for 2 weeks
LBL-29925
Copy 2
Bldg. 50 Library.

Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

LBL-29925

COMPUTER PROGRAMS FOR SIMULATION OF ELECTRODEPOSITION

Kenneth G. Jordan and Charles W. Tobias

Department of Chemical Engineering
University of California, Berkeley

Chemical Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720

December 1990

This work was supported by the Assistant Secretary for Conservation and Renewable Energy, Deputy Assistant Secretary for Utility Technologies, Office of Energy Management, Advanced Utility Concepts Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Contents

Introduction 1
Overview of Finite Element Programs 2
Overview of Boundary Element Programs 3
ECFMESH 6
ECFDEFINE 22
ECFPREPRO 42
ECFCOMPUTE 44
ECFPLOT 65
ECFISO 76
ECFSUBS 80
ECF.INC* 101
ECBPAR 120
ECBPREPRO 125
ECBM 131
ECBINT 154
ECBPLOT 155
ECBSUBS 162
ECB.INC* 244
SUBS 264
MATHSUBS 318

Introduction

In this report, FORTRAN programs and subroutines useful for modelling moving boundaries in electrochemical systems are listed. These programs were used by Kenneth Guy Jordan[1] to simulate moving boundaries during electrodeposition in presence of transport limited additives that acted either by (i) blocking the electrode to electrodeposition, or (ii) dissolving the electrode instantaneously upon their arrival, lowering the current efficiency. The programs were also used to analyze the behavior of evolving surface profiles during periodic current reversal.

The following elements of programming style were used:

(1) Local subroutines (subroutines used only be that program) are listed in alphabetical order in the program listing, following the END of the main program module,

(2) Global subroutines (subroutines used by more than one program) are listed in alphabetical order in the subroutine file listing following the program listings. The global subroutine files are ECFSUBS, ECBSUBS, SUBS, or MATHSUBS.

(3) Variables in common blocks are defined in "Include" files. These files contain the declarations, common block, and description of the variables. The statements in the file are incorporated into the FORTRAN programs during compilation by using the INCLUDE statement. The "Include" files are listed in alphabetical order following the subroutines.

There are two distinct sets of programs: one set based on the finite element method, and another set based on the boundary element method. Both sets are written in FORTRAN for VAX/VMS. In the following, the intended sequence for using the programs, as well as the purpose of the program, the input required, and the output generated are briefly described.

---

[1] K.G. Jordan, ''Levelling Of Microprofiles In Electrodeposition'', Ph. D. Thesis, U.C. Berkeley, Dec (1990)

Overview of Finite Element Method Programs

There are six basic programs in the set using FEM. The intended order of use is to first run ECFMESH, then ECFDEFINE, followed by ECFPREPRO, and ECFCOMPUTE. The results from ECFCOMPUTE may be prepared for graphs by then running ECFPLOT, and ECFISO. A brief description of each program follows

ECFMESH

This program takes a concise geometric description of the FEM problem to be solved from a *.MAC file (see subroutine ReadECFMac) and generates mesh used to solve the problem: the boundary node numbers, locations, and condition, as well as node numbers and locations. The description of the FEM problem is written into *.HDR, *.MSH, *.BC, *.PAR, *.RES and *.NAM files. (See subroutines WriteMeshHdr, WriteMesh, WriteBC, WritePar, and WriteRes)

ECFDEFINE

This program allows a new FEM problem description to be entered, or an old FEM problem description to be modified. Information is read from and written to all the files that describe the problem (see subroutine WriteFEMUser.)

ECFPREPRO

This program computes the bandwidth for the FEM problem. It reads the entire FEM problem description (see subroutine ReadFEMUser) but only the mesh header file *.HDR is written (see subroutine WriteMeshHdr.)

ECFCOMPUTE

This program uses the Galerkin Finite Element Method to calculate the solution to the current distribution or convective-diffusion problem described in the files created by ECFMESH, ECFDEFINE, and ECFPREPRO. The entire FEM problem description is read (see subroutine ReadFEMUser), the results calculated, only the results are written (file *.RES, see subroutine WriteRes.)

ECFPLOT

This program takes the FEM results and prepares them for use a suitable graphics package such as *TELL-A-GRAPH*, or a user program that uses the package *DISSPLA*. The entire FEM problem description is read, and files created with X,Y data isopotential or iso-concentration contours (*.ISO), as well as concentration of potential values on the boundary (*.VAL) and the slopes of these on the bondary (*.SLP).

ECFISO

This program will take data from a *.ISO file and generate a graph using the *DISS-PLA* graphics package.

Overview of Boundary Element Method Programs

There are five programs in the set that uses the boundary element method. These are usually run in the order, ECBPAR, ECBPREPRO, ECBM, ECBINT, and ECBPLOT. A brief description of each program follows.

ECBPAR

This program reads a concise description of the type of BEM problem to be solved from a *.DIM file (see subrouine ReadUserParams), creates an appropriate *.PAR file (see subroutine WritePar), and prints a report on the problem to be solved (see subroutine PrintUserParams.)

ECBPREPRO

This program reads a concise geometrical description of the BEM problem to be solved (including boundary conditions) from file *.MAC (see subroutine ReadMAC) and creates the boundary mesh, stored in file *.MSH (see subroutine WriteMesh), and the source points, stored in file *.SRC (see subroutine WriteSrc).

ECBM

This program uses the boundary element method to solve potential problems, or simultaneous diffusion and potential problems, for the current distributions in electrochemical systems with moving boundaries. The BEM problem description is read from files *.PAR, *.MSH, and *.MAC. Time vs. maximum profile depth is stored in file *.TD. The location of the moving boundary is stored in a series of files named *.BND. The final results are stored in *.MSH.

ECBINT

This program reads the solution from *.MSH and a list of internal points from *.INT (see subroutine ReadInt), and computes the value of the potential at all of the internal points.

The value of the potential at all of the internal points is written to file *.INT.

ECBPLOT

From the results generated by ECBM, this program generates files suitable for use in a number of grapics packages, such as *TELL-A-GRAF*, *DISSPLA*, or *LOTUS*. The x,y locations of the nodes are placed in file ECBPLOT.NOD; the x,y locations of the source points are written to file ECBPLOT.SRC. If desired, the user can cause solution values to be stored in file ECBPLOT.SOL, and BC values to be storeed in ECBPLOT.SET. This program will also perform an integration of the flux along a given portion of the boundary. This is useful for computing cell resistances.

```
      PROGRAM ECFMesh

C MakeMesh Program Takes FEM.MAC input file and generates
C       BdNodes()
C       NLoc(,)         x
C       Nodes(,) x
C       NoOfBDNodes
C       NoOfDim                 x
C       NoOfElems       x
C       NoOfNodes       x

      INCLUDE 'ECFHDR.INC'
      INCLUDE 'ECFMAC.INC'

      INTEGER MacElm,FilLen
      LOGICAL Shared(4)
      CHARACTER*80 FileName

      CALL ChInp('File name? (no extension)',FileName)
      CALL StringLength(FileName,FilLen)

      CALL ReadECFMac(FileName(1:FilLen)//'.MAC')

      DO MacElm=1,NoOfMacElem

        Shared(1)=.FALSE.
        Shared(2)=.FALSE.
        Shared(3)=.FALSE.
        Shared(4)=.FALSE.

        CALL SharedSides(MacElm,Shared)
        IF (MacElemType(MacElm).EQ.'S') THEN
          CALL ProcSqrMacElm(MacElm,Shared)
        ELSE IF (MacElemType(MacElm).EQ.'T') THEN
          CALL ProcTriMacElm(MacElm,Shared)
        ELSE
          WRITE(*,*) 'Error: Unknown macro element type=',
     & MacElemType(MacElm)
          STOP 'ECFMesh: Main Program'
        END IF
      END DO

      CALL GenerateBDNodes
      ECFMesh=.TRUE.

      CALL PrintMacReport

      CALL WriteMeshHdr(FileName(1:FilLen)//'.HDR')
      CALL WriteMesh(FileName(1:FilLen)//'.MSH')
      CALL WriteBC(FileName(1:FilLen)//'.BC')
      CALL WritePar(FileName(1:FilLen)//'.PAR')
      CALL WriteRes(FileName(1:FilLen)//'.RES')

      OPEN(UNIT=1,FILE=FileName(1:FilLen)//'.NAM',STATUS='NEW',
     & FORM='FORMATTED')
      WRITE(1,*) '''',FileName(1:FilLen)//'.HDR','''
      WRITE(1,*) '''',FileName(1:FilLen)//'.MSH','''
      WRITE(1,*) '''',FileName(1:FilLen)//'.MAC','''
```

```
      WRITE(1,*) '''',FileName(1:FilLen)//'.BC',''''
      WRITE(1,*) '''',FileName(1:FilLen)//'.PAR',''''
      WRITE(1,*) '''',FileName(1:FilLen)//'.RES',''''
      WRITE(1,*) '''',FileName(1:FilLen)//'.ISO',''''
      WRITE(1,*) '''',FileName(1:FilLen)//'.SLP',''''
      WRITE(1,*) ''' '''
      WRITE(1,*) ''' '''
      CLOSE(1)

      STOP 'MakeMesh: Normal End'
      END

C ************************************************************** GenerateBDNodes

      SUBROUTINE GenerateBDNodes

      INCLUDE 'ECFHDR.INC'
      INCLUDE 'ECFMSH.INC'
      INCLUDE 'ECFMAC.INC'

      LOGICAL SideNotFound
      INTEGER BDSeg,BDSegp1,MacElem,MacSeg,MacSegp1,M,Vert1,Vert2
      INTEGER Upper

      DO BDSeg=1,NoOfMacBDNodes

        CALL IncrementI(BDSeg,BDSegp1,1,NoOfMacBDNodes)
        SideNotFound=.TRUE.

C The following DO Loop locates the side on the boundary.

        DO MacElem=1,NoOfMacElem
          CALL GetUpper(MacElemType(MacElem),Upper)
          DO MacSeg=1,Upper

            CALL IncrementI(MacSeg,MacSegp1,1,Upper)

            Vert1=MacElemVert(MacElem,MacSeg)
            Vert2=MacElemVert(MacElem,MacSegp1)

            IF ((Vert1.EQ.MacBDNodes(BDSeg)).AND.
     & (Vert2.EQ.MacBDNodes(BDSegp1)) ) THEN

              MacBDNodeNo(Vert1)=NoOfBDNodes+1
D               WRITE(*,5) Vert1,NoOfBDNodes+1
5             FORMAT(X,'Mac BD Node No ',I5,' is Boundary Node ',I5)
              SideNotFound=.FALSE.

                DO M=1,NoOfMicElm(MacElem,MacSeg)
                NoOfBDNodes=NoOfBDNodes+1
                  BDNodes(NoOfBDNodes)=SharedNodes(MacElem,MacSeg,M)
D                 WRITE(*,10) NoOfBDNodes,BDNodes(NoOfBDNodes)
10                FORMAT(X,'     BDNodes(',I5,')=',I5)
              END DO

            END IF
          END DO
        END DO
```

```
          IF (SideNotFound) THEN
            WRITE(*,*) 'Error: Boundary Side not found.'
            WRITE(*,*) '       BD Seg=',BDSeg
            STOP 'In Routine: GenerateBDNodes'
          END IF

        END DO

        RETURN
        END

C ********************************************************************** MakeSpine

        SUBROUTINE MakeSpine(StaPtX,StaPtY,EndPtX,EndPtY,
     & NoOfMicElems,SpaType,SpaFac,
     & SpineX,SpineY)

C StaPtX,Y       Starting Point Location
C EndPtX,Y       Ending Point Location
C NoOfMicElems        The number of microelements that must be on this side
C SpaType        Spacing Type for microelements
C SpaFac         Spacing Factor for microelements
C SpineX()       X Locations for spine
C SpineY()       Y Locations for spine.

        INTEGER NoOfMicElems
        REAL*8 StaPtX,StaPtY,EndPtX,EndPtY,SpineX(*),SpineY(*),SpaFac
        CHARACTER*(*) SpaType

        INTEGER I

        IF (SpaType.EQ.'LIN') THEN

          SpineX(1)=StaPtX
          SpineY(1)=StaPtY

          DO I=1,NoOfMicElems
            SpineX(I)=StaPtX + (EndPtX-StaPtX) * FLOAT(I-1)
     & / FLOAT(NoOfMicElems)
            SpineY(I)=StaPtY + (EndPtY-StaPtY) * FLOAT(I-1)
     & / FLOAT(NoOfMicElems)
          END DO

          SpineX(NoOfMicElems+1)=EndPtX
          SpineY(NoOfMicElems+1)=EndPtY

        ELSE

          WRITE(*,*) 'Error: Unknown Spacing Type=',SpaType
          STOP 'In Routine: MakeSpine'

        END IF

        RETURN
        END

C ********************************************************************** PrintMacReport
```

```
      SUBROUTINE PrintMacReport

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFMAC.INC'

      INTEGER I,J,MacNodeNo,Upper
      CHARACTER*80 Dummy,Output

      CALL PrintProbHeader
      WRITE(*,*) ' '

C 123456789a123456789b123456789c123456789d123456789e12
C ------------------------------------------------- Macro Vertices & Elements
C                                                   iiii     iiii
C 123456789a123456789b123456789c123456789d123456789e123456789    1234567

      WRITE(*,300) NoOfMacVert,NoOfMacElem
300   FORMAT(X,52('-'),' Macro Vertices & Elements'/
   & X,59X,I4,7X,I4)

      WRITE(*,*) ' '
      WRITE(*,310)
310   FORMAT(X,23X,'Vertex No',3X,'X Coord',7X,'Y Coord')
      WRITE(*,*) ' '

      DO I=1,NoOfMacVert
        WRITE(*,320) I,MacVertX(I),MacVertY(I)
      END DO
320   FORMAT(X,25X,I4,4X,G11.4,3X,G11.4)

C 123456789a12345      123      1234567
C            Vertex No   X Coord      Y Coord
C              iiii    sd.dddde+dd   sd.dddde+dd
C 123456789a12345    1234123456789a1123

C      MACRO---------------------------     MICRO---------------
C      Elem   Vertex     Spacing            Elems  Vertex   Bdry
C       No            Type   Factor              No     Node

C      iiii    iiii   aaaa  sd.dddde+dd    iiii    iiii    iiii
C      1234    1234   1234  123456789a1     1234    1234    1234

      WRITE(*,*) ' '
      WRITE(*,340)

      DO I=1,NoOfMacElem
        MacNodeNo=MacBDNodeNo(MacElemVert(I,1))
        IF (MacNodeNo.NE.0) THEN
          WRITE(*,350) I,MacElemType(I),
   & MacElemVert(I,1),SpaType(I,1),SpaFac(I,1),
   & NoOfMicElm(I,1),BDNodes(MacNodeNo),MacNodeNo
        ELSE
          WRITE(*,350) I,MacElemType(I),
   & MacElemVert(I,1),SpaType(I,1),SpaFac(I,1),
   & NoOfMicElm(I,1),BDNodes(MacNodeNo)
        END IF
```

```
          CALL GetUpper(MacElemType(I),Upper)

          DO J=2,Upper
            MacNodeNo=MacBDNodeNo(MacElemVert(I,J))
            IF (MacNodeNo.NE.0) THEN
              WRITE(*,360) MacElemVert(I,J),SpaType(I,J),SpaFac(I,J),
     & NoOfMicElm(I,J),BDNodes(MacNodeNo),MacNodeNo
            ELSE
              WRITE(*,360) MacElemVert(I,J),SpaType(I,J),SpaFac(I,J),
     & NoOfMicElm(I,J),BDNodes(MacNodeNo)
            END IF
          END DO
          WRITE(*,*) ' '
        END DO

340     FORMAT(X,7X,'MACRO',27('-'),5X,'MICRO',15('-')/
     & X,7X,'Elem   Vertex     Spacing',12X,'Elems  Vertex   Bdry'/,
     & X,8X,'No',12X,'Type   Factor',18X,'No     Node'/)

c                                12345
C1234567    123456789a123456789b1234567          123456789a12345
C          123      12345       123456789a12     12      123
C12345678  123456789a1    123       123456789a12345678  12345
C1234567   1234    123    12          12345    1234    1234

350     FORMAT(X,7X,I4,A1,3X,I4,3X,A4,2X,G11.4,5X,I4,4X,I4,:,4X,I4)
360     FORMAT(X,7X,4X,4X,I4,3X,A4,2X,G11.4,5X,I4,4X,I4,:,4X,I4)

        RETURN
        END

C ************************************************************** ProcSqrMacElm

        SUBROUTINE ProcSqrMacElm(MacElm,Shared)

C Process MacroElement

        INCLUDE 'ECFHdr.INC'
        INCLUDE 'ECFMsh.INC'
        INCLUDE 'ECFMAC.INC'

        INTEGER MacElm
        LOGICAL Shared(*)

        REAL*8 SpineX12(MaxNoOfMicVert),SpineY12(MaxNoOfMicVert)
        REAL*8 SpineX23(MaxNoOfMicVert),SpineY23(MaxNoOfMicVert)
        REAL*8 SpineX34(MaxNoOfMicVert),SpineY34(MaxNoOfMicVert)
        REAL*8 SpineX41(MaxNoOfMicVert),SpineY41(MaxNoOfMicVert)
        REAL*8 SpineX(MaxNoOfMicVert),SpineY(MaxNoOfMicVert)
        REAL*8 StaPtX,StaPtY,EndPtX,EndPtY
        INTEGER MicNode,Point,MicSta,MicEnd,Error,I,J

C Make Spine for First, Second, Third, and Fourth Sides
C (The following calls are really typing aids to the call of MakeSpine)

        CALL SideSpine(MacElm,1,2,1,SpineX12,SpineY12)
        CALL SideSpine(MacElm,2,3,2,SpineX23,SpineY23)
```

```
      CALL SideSpine(MacElm,4,3,1,SpineX34,SpineY34)
      CALL SideSpine(MacElm,1,4,2,SpineX41,SpineY41)

C Store Spine Into Mesh (for side 4)

C +3+  Spine Side definition. | 4-3  Loc. Vert. Def.
C 4 2                         | ||
C +1+                         | 1-2

      IF (Shared(1)) THEN
        MicSta=2
      ELSE
        MicSta=1
      END IF

      IF (Shared(3)) THEN
        MicEnd=NoOfMicElm(MacElm,2)
      ELSE
        MicEnd=NoOfMicElm(MacElm,2)+1
      END IF

D     WRITE(*,20) MicSta,MicEnd
20    FORMAT(X,'Square Macro Elem: Start=',I5,'   End=',I5)

      IF (.NOT.Shared(4)) THEN
        CALL TransferSpine(MicSta,MicEnd,MacElm,4,1,
     & SpineX41,SpineY41)
      END IF

C Now do spines next to side 4

      DO MicNode=2,NoOfMicElm(MacElm,1)

C Locate Intersections Between Crossing Spines

C Store first point on next spine.

        SpineX(1)=SpineX12(MicNode)
        SpineY(1)=SpineY12(MicNode)

C Store last point on next spine.

        SpineX(NoOfMicElm(MacElm,2)+1)=SpineX34(MicNode)
        SpineY(NoOfMicElm(MacElm,2)+1)=SpineY34(MicNode)

C Find intermediate points.

        DO Point=2,NoOfMicElm(MacElm,2)

          CALL ComputeIntersection(SpineX(1),SpineY(1),
     & SpineX(NoOfMicElm(MacElm,2)+1),
     & SpineY(NoOfMicElm(MacElm,2)+1),
     & SpineX41(Point),SpineY41(Point),
     & SpineX23(Point),SpineY23(Point),
     & SpineX(Point),SpineY(Point),Error)

          IF (Error.NE.0) THEN
            WRITE(*,*) 'Error: Spines do not intersect.'
```

```
            STOP 'In Routine: ProcMacElm'
          END IF
        END DO

C IF--- Store node numbers of possible shared nodes. (on side 1)
C CALL- Transfer spine locations into Nodes(). Save node numbers
C IF--- Store node numbers of possible shared nodes. (on side 3)

        IF (MicSta.EQ.1) THEN
          SharedNodes(MacElm,1,MicNode)=NoOfNodes+1
D         WRITE(*,10) MacElm,1,MicNode,NoOfNodes+1
10        FORMAT(X,'SQRSharedNodes(',I5,',',I5,',',I5,')=',I5)
        END IF
        CALL TransferSpine(MicSta,MicEnd,0,0,MicNode,SpineX,SpineY)
        IF (MicEnd.EQ.NoOfMicElm(MacElm,2)+1) THEN
          SharedNodes(MacElm,3,NoOfMicElm(MacElm,3)+2-MicNode)
    & =NoOfNodes
D         WRITE(*,10) MacElm,3,NoOfMicElm(MacElm,3)+2-MicNode,
D    & NoOfNodes
        END IF

      END DO

C If Side 2 is not shared, (last side parallel to side 4), store it.

      IF (.NOT.Shared(2))
    & CALL TransferSpine(MicSta,MicEnd,MacElm,2,
    & NoOfMicElm(MacElm,1)+1,
    & SpineX23,SpineY23)

C Store corner nodes as appropriate

C Side 4

      SharedNodes(MacElm,1,1)
    & = SharedNodes(MacElm,4,NoOfMicElm(MacElm,4)+1)
      SharedNodes(MacElm,3,NoOfMicElm(MacElm,3)+1)
    & = SharedNodes(MacElm,4,1)

D     WRITE(*,10) MacElm,1,1,SharedNodes(MacElm,1,1)
D     WRITE(*,10) MacElm,3,NoOfMicElm(MacElm,3)+1,
D    & SharedNodes(MacElm,4,1)

C Side 2

      SharedNodes(MacElm,3,1)
    & = SharedNodes(MacElm,2,NoOfMicElm(MacElm,2)+1)
      SharedNodes(Macelm,1,NoOfMicElm(MacElm,1)+1)
    & = SharedNodes(MacElm,2,1)

D     WRITE(*,10) MacElm,3,1,SharedNodes(MacElm,3,1)
D     WRITE(*,10) MacElm,1,NoOfMicElm(MacElm,3)+1,
D    & SharedNodes(MacElm,2,1)
D     WRITE(*,*) ' '

C Store node #s of mesh along shared sides of macro element.

      DO I=1,NoOfMicElm(MacElm,2)+1
```

```
          IF (Shared(2)) MacNodes(NoOfMicElm(MacElm,1)+1,I)
     & =SharedNodes(MacElm,2,I)
          IF (Shared(4)) MacNodes(1,I)
     & =SharedNodes(MacElm,4,NoOfMicElm(MacElm,4)+2-I)
        END DO

        DO I=1,NoOfMicElm(MacElm,1)+1
          IF (Shared(1)) MacNodes(I,1)
     & =SharedNodes(MacElm,1,I)
          IF (Shared(3)) MacNodes(I,NoOfMicElm(MacElm,2)+1)
     & =SharedNodes(MacElm,3,NoOfMicElm(MacElm,3)+2-I)
        END DO

C Use saved node numbers to generate NLoc matrix, and element type.

        DO J=1,NoOfMicElm(MacElm,1)
          DO I=1,NoOfMicElm(MacElm,2)
            NoOfElems=NoOfElems+1
            ElemType(NoOfElems)='S'
            NLoc(NoOfElems,1)=MacNodes(J,I)
            NLoc(NoOfElems,2)=MacNodes(J+1,I)
            NLoc(NoOfElems,3)=MacNodes(J+1,I+1)
            NLoc(NoOfElems,4)=MacNodes(J,I+1)
          END DO
        END DO

        RETURN
        END

C ************************************************************** ProcTriMacElm

        SUBROUTINE ProcTriMacElm(MacElm,Shared)

C Process MacroElement

        INCLUDE 'ECFHdr.INC'
        INCLUDE 'ECFMsh.INC'
        INCLUDE 'ECFMAC.INC'

        INTEGER MacElm
        LOGICAL Shared(*)

        REAL*8 SpineX12(MaxNoOfMicVert),SpineY12(MaxNoOfMicVert)
        REAL*8 SpineX21(MaxNoOfMicVert),SpineY21(MaxNoOfMicVert)
        REAL*8 SpineX32(MaxNoOfMicVert),SpineY32(MaxNoOfMicVert)
        REAL*8 SpineX31(MaxNoOfMicVert),SpineY31(MaxNoOfMicVert)

        REAL*8 SpineX(MaxNoOfMicVert),SpineY(MaxNoOfMicVert)
        REAL*8 StaPtX,StaPtY,EndPtX,EndPtY
        INTEGER MicNode,Point,MicSta,MicEnd,Error,I,J,Upper

C Make Spine for First, Second, Third, and Fourth Sides
C (The following calls are really typing aids to the call of MakeSpine)

        CALL SideSpine(MacElm,2,1,1,SpineX21,SpineY21)
        CALL SideSpine(MacElm,1,2,1,SpineX12,SpineY12)
        CALL SideSpine(MacElm,3,2,2,SpineX32,SpineY32)
        CALL SideSpine(MacElm,3,1,3,SpineX31,SpineY31)
```

```
C +     Spine Side definition. | 2    Loc. Vert. Def.
C 21                           | |
C +3+                          | 3-1

C Store Spine Into Mesh for side 2.

        IF (Shared(3)) THEN
          MicSta=2
        ELSE
          MicSta=1
        END IF

        Upper=NoOfMicElm(MacElm,2)+1
        IF (Shared(1)) THEN
          MicEnd=Upper-1
        ELSE
          MicEnd=Upper
        END IF

D       WRITE(*,20) MicSta,MicEnd
20      FORMAT(X,'Triang Macro Elem: Start=',I5,'   End=',I5)

        IF (.NOT.Shared(2))
     & CALL TransferSpine(MicSta,MicEnd,MacElm,2,1,
     & SpineX32,SpineY32)

C Now do spines next to side 2

        DO MicNode=2,NoOfMicElm(MacElm,3)

C MicEnd must be decreased by one for each subsequent spine.

          MicEnd=MicEnd-1
          Upper=Upper-1

C Locate Intersections Between Crossing Spines

C Store first point on next spine.

          SpineX(1)=SpineX31(MicNode)
          SpineY(1)=SpineY31(MicNode)

C Store last point on next spine.

          SpineX(Upper)=SpineX21(MicNode)
          SpineY(Upper)=SpineY21(MicNode)

C Find intermediate points.

          DO Point=2,Upper-1

            CALL ComputeIntersection(SpineX(1),SpineY(1),
     & SpineX(Upper),SpineY(Upper),
     & SpineX32(Point),SpineY32(Point),
     & SpineX12(Point),SpineY12(Point),
     & SpineX(Point),SpineY(Point),Error)

            IF (Error.EQ.2) THEN
```

```
              WRITE(*,*) 'Error: Spines do not intersect.'
              STOP 'In Routine: ProcMacElm'
            END IF

          END DO

C IF--- Store node numbers of possible shared nodes. (on side 1)
C CALL- Transfer spine locations into Nodes(). Save node numbers
C IF--- Store node numbers of possible shared nodes. (on side 3)

          IF (MicSta.EQ.1) THEN
            SharedNodes(MacElm,3,MicNode)
     & =NoOfNodes+1
D           WRITE(*,10) MacElm,3,MicNode,
D    & NoOfNodes+1
10          FORMAT(X,'TRISharedNodes(',I5,',',I5,',',I5,')=',I5)
          END IF
          CALL TransferSpine(MicSta,MicEnd,0,0,MicNode,SpineX,SpineY)
          IF (MicEnd.EQ.Upper) THEN
            SharedNodes(MacElm,1,NoOfMicElm(MacElm,1)+2-MicNode)
     & =NoOfNodes
D           WRITE(*,10) MacElm,1,NoOfMicElm(MacElm,1)+2-MicNode,
D    & NoOfNodes
          END IF
        END DO

        IF ((.NOT.Shared(1)).AND.(.NOT.Shared(3))) THEN
          NoOfNodes=NoOfNodes+1
          SharedNodes(MacElm,1,1)=NoOfNodes
          SharedNodes(MacElm,3,NoOfMicElm(MacElm,3)+1)=NoOfNodes
          Nodes(NoOfNodes,1)=MacVertX(MacElemVert(MacElm,1))
          Nodes(NoOfNodes,2)=MacVertY(MacElemVert(MacElm,1))
          MacNodes(NoOfMicElm(MacElm,3)+1,1)=NoOfNodes
        END IF

C Store corner nodes as appropriate

C Side 2

        SharedNodes(MacElm,1,NoOfMicElm(MacElm,1)+1)
     & = SharedNodes(MacElm,2,1)
        SharedNodes(MacElm,3,1)
     & = SharedNodes(MacElm,2,NoOfMicElm(MacElm,2)+1)

D       WRITE(*,10) MacElm,1,NoOfMicElm(MacElm,1)+1,
D    & SharedNodes(MacElm,2,1)
D       WRITE(*,10) MacElm,3,1,SharedNodes(MacElm,3,1)

C Side 1

        SharedNodes(MacElm,1,1)
     & = SharedNodes(MacElm,3,NoOfMicElm(Macelm,3)+1)
D       WRITE(*,10) MacElm,1,1,SharedNodes(MacElm,1,1)

D       WRITE(*,*) ' '

C Store node #s of mesh along shared sides of macro element.
```

```
      DO I=1,NoOfMicElm(MacElm,1)+1

        IF (Shared(1)) MacNodes(I,NoOfMicElm(MacElm,2)+2-I)
     & =SharedNodes(MacElm,1,NoOfMicElm(MacElm,1)+2-I)

        IF (Shared(2)) MacNodes(1,I)
     & =SharedNodes(MacElm,2,NoOfMicElm(MacElm,2)+2-I)

        IF (Shared(3)) MacNodes(I,1)
     & =SharedNodes(MacElm,3,I)

      END DO

C Use saved node numbers to generate NLoc matrix, and element type.

      DO J=1,NoOfMicElm(MacElm,3)

        DO I=1,NoOfMicElm(MacElm,2)-J
          NoOfElems=NoOfElems+1
          ElemType(NoOfElems)='S'
          NLoc(NoOfElems,1)=MacNodes(J,I)
          NLoc(NoOfElems,2)=MacNodes(J+1,I)
          NLoc(NoOfElems,3)=MacNodes(J+1,I+1)
          NLoc(NoOfElems,4)=MacNodes(J,I+1)
        END DO

        NoOfElems=NoOfElems+1
        ElemType(NoOfElems)='T'
        I=NoOfMicElm(MacElm,2)+1-J
        NLoc(NoOfElems,1)=MacNodes(J,I)
        NLoc(NoOfElems,2)=MacNodes(J+1,I)
        NLoc(NoOfElems,3)=MacNodes(J,I+1)

      END DO

      RETURN
      END

C ********************************************************************** ReadECFMAC

      SUBROUTINE ReadECFMAC(FileName)

      INCLUDE 'ECFHDR.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMAC.INC'

      CHARACTER*(*) FileName
      INTEGER I,J,NoOfVert
      LOGICAL FileEXIST

      INQUIRE(FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) THEN
        WRITE(*,*) 'Error: Unknown File Name=',FileName
        CALL LIB$SPAWN('TYPE ECFMAC.STR')
        STOP 'In Routine: ReadECFMAC'
      END IF

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')
```

```
      READ(1,*) BasisDeg,Tolerance,NoOfGaussPts,MaxNoOfIter
      READ(1,*) NoOfEqu,NoOfDim
      READ(1,*) Debugging,Convection,PolyFlow,Guess,Nonnegative

C Read Macro Vertices

      READ(1,*) NoOfMacVert
      DO I=1,NoOfMacVert
        READ(1,*) MacVertX(I),MacVertY(I)
      END DO

C Read Macro Elements

      READ(1,*) NoOfMacElem
      DO I=1,NoOfMacElem
        READ(1,*) MacElemType(I)

        CALL GetUpper(MacElemType(I),NoOfVert)

        READ(1,*) (MacElemVert(I,J),J=1,NoOfVert)
        READ(1,*) (NoOfMicElm(I,J),J=1,NoOfVert)

        IF (MacElemType(I).EQ.'S') THEN
C Square Macro Element
          NoOfMicElm(I,3)=NoOfMicElm(I,1)
          NoOfMicElm(I,4)=NoOfMicElm(I,2)
        ELSE
C Triangular Macro Element
          NoOfMicElm(I,2)=NoOfMicElm(I,1)
          NoOfMicElm(I,3)=NoOfMicElm(I,1)
        END IF

C The following DO-Loop to check that number of micro elements on each
C side is less than the maximum number of micro elements that this program
C can handle.

        DO J=1,NoOfVert
          IF (NoOfMicElm(I,J).GE.MaxNoOfMicVert) THEN
            WRITE(*,*) 'Error: Too many microelements!  Either'
              WRITE(*,*) '       a) reduce number of microelements, or'
              WRITE(*,*) '       b) increase MaxNoOfMicVert in ECFMAC.INC'
              WRITE(*,*) '          and recompile programs.'
            WRITE(*,*) '       NoOfMicElm=',NoOfMicElm(I,J)
            WRITE(*,*) '   MaxNoOfMicVert=',MaxNoOfMicVert
            STOP 'In Routine: ReadECFMAC'
          END IF
        END DO
        READ(1,*) (SpaType(I,J),J=1,NoOfVert)
        READ(1,*) (SpaFac(I,J),J=1,NoOfVert)
      END DO

C Read the macrovertices on the boundary

      READ(1,*) NoOfMacBDNodes

      DO I=1,NoOfMacBDNodes
        READ(1,*) MacBDNodes(I)
      END DO
```

```
      CLOSE(1)

C FEM.MAC File Structure

c BasisDeg,Tolerance,NoOfGaussPts,MaxNoOfIter          )- goes to mesh hdr file
c NoOfEqu,NoOfDim                                      )
c Debugging,Convection,PolyFlow,Guess,Nonnegative      )

C NoOfMacVert                                          )- goes to mesh file
C X1,Y1
C X2,Y2
C ...
C XN,YN

C NoOfMacElm

C MacElemType
C MacVert1,MacVert2,MacVert3,MacVert4
C NoOfMicElm12,23,34,41
C SpaType12,23,34,41
C SpaFac12,23,34,41

C ...

C NoOfMacBDNodes
C MacBDNodes(1)
C ...
C MacBDNodes(N)

      RETURN
      END

C ********************************************************************** SharedSides

      SUBROUTINE SharedSides(MacElm,Shared)

      INCLUDE 'ECFMAC.INC'

      LOGICAL Shared(*)
      INTEGER MacElm,PrevElm,K,L,M,Kp1,Lp1,Km1
      INTEGER UpperPrev,UpperMac

      CALL GetUpper(MacElemType(MacElm),UpperMac)

      DO PrevElm=1,MacElm-1

        CALL GetUpper(MacElemType(PrevElm),UpperPrev)

        DO K=1,UpperMac

          CALL IncrementI(K,Kp1,1,UpperMac)

          DO L=1,UpperPrev

              CALL IncrementI(L,Lp1,1,UpperPrev)

            IF (
     & (MacElemVert(MacElm,K).EQ.MacElemVert(PrevElm,Lp1)).AND.
```

```
     & (MacElemVert(MacElm,Kp1).EQ.MacElemVert(PrevElm,L)) ) THEN

D                WRITE(*,10) MacElm,K,PrevElm,L
10       FORMAT(X,'Note: Mac Elem ',I5,' Side ',I5,
     & ' is shared with Mac Elem ',I5,' Side ',I5)
                   Shared(K)=.TRUE.
                 NoOfMicElm(MacElm,K)=NoOfMicElm(PrevElm,L)

                   DO M=1,NoOfMicElm(MacElm,K)+1
                     SharedNodes(MacElm,K,M)
     & =SharedNodes(PrevElm,L,NoOfMicElm(MacElm,K)+2-M)
                   END DO

                   CALL IncrementI(K,Km1,-1,UpperMac)
                 SharedNodes(MacElm,Kp1,1)=SharedNodes(MacElm,K,
     & NoOfMicElm(MacElm,K)+1)
                 SharedNodes(MacElm,Km1,
     & NoOfMicElm(MacElm,Km1)+1)=SharedNodes(MacElm,K,1)
               END IF

             END DO
           END DO
         END DO

         RETURN
         END

C ********************************************************************** SideSpine

         SUBROUTINE SideSpine(MacElm,StaVert,EndVert,Side,SpineX,SpineY)

C MacElm      The current macro element being processed
C StaVert     The local macro vertex for starting point.
C EndVert     The local macro vertex for ending point.
C Side        The side of the mesh.
C SpineX      The X locations on the spine
C SpineY      The Y locations on the spine

         INCLUDE 'ECFMAC.INC'

         REAL*8 SpineX(*),SpineY(*)
         INTEGER MacElm,StaVert,EndVert,Side

         REAL*8 StaPtX,StaPtY,EndPtX,EndPtY

         StaPtX=MacVertX(MacElemVert(MAcElm,StaVert))
         StaPtY=MacVertY(MacElemVert(MacElm,StaVert))

         EndPtX=MacVertX(MacElemVert(MacElm,EndVert))
         EndPtY=MacVertY(MacElemVert(MacElm,EndVert))

         CALL MakeSpine(StaPtX,StaPtY,EndPtX,EndPtY,
     & NoOfMicElm(MacElm,Side),
     & SpaType(MacElm,Side),SpaFac(MacElm,Side),
     & SpineX,SpineY)

         RETURN
         END
```

```
C ************************************************************** TransferSpine

      SUBROUTINE TransferSpine(MicSta,MicEnd,MacElm,MacSide,SpeNode,
     & SpineX,SpineY)

C This subroutine stores the node locations from one spine into Nodes.

      INCLUDE 'ECFHDR.INC'
      INCLUDE 'ECFMAC.INC'
      INCLUDE 'ECFMSH.INc'

C SpineX()      X Locations
C SpineY()      Y Locations
C MacElm
C MacSide
C SpeNode
C MicSta        1st (local) mic node number
C MicEnd        last (local) mic node number

      INTEGER MicSta,MicEnd,MacElm,MacSide,SpeNode
      REAL*8 SpineX(*),SpineY(*)
      INTEGER MicNod

      DO MicNod=MicSta,MicEnd
        NoOfNodes=NoOfNodes+1

        IF (NoOfNodes.GT.MaxNoOfNodes) THEN
          WRITE(*,*) 'Error: Too many micro nodes.'
          WRITE(*,*) '       Maximum allowed=',MaxNoOfNodes
          WRITE(*,*) '       Either scale back, or increase ',
     & 'MaxNoOfNodes in ECFHDR.INC'
          WRITE(*,*) '       and recompile.'
          STOP 'In Routine: TransferSpine'
        END IF

        IF (MacSide.EQ.4) THEN

          SharedNodes(MacElm,MacSide,
     & NoOfMicElm(MacElm,MacSide)+2-MicNod)=NoOfNodes
D         WRITE(*,10) MacElm,MacSide,
D    & NoOfMicElm(MacElm,MacSide)+2-MicNod,NoOfNodes
10        FORMAT(X,'TFRSharedNodes(',I5,',',I5,',',I5,')=',I5)

        ELSE IF (MacSide.EQ.2) THEN

          IF (MacElemType(MacElm).EQ.'S') THEN
            SharedNodes(MacElm,MacSide,MicNod)=NoOfNodes
D           WRITE(*,10) MacElm,MacSide,MicNod,NoOfNodes
          ELSE
            SharedNodes(MacElm,MacSide,
     & NoOfMicElm(MacElm,MacSide)+2-MicNod)=NoOfNodes
D           WRITE(*,10) MacElm,MacSide,
D    & NoOfMicElm(MacElm,MacSide)+2-MicNod,NoOfNodes
          END IF

        END IF

        Nodes(NoOfNodes,1)=SpineX(MicNod)
```

```
    Nodes(NoOfNodes,2)=SpineY(MicNod)
    IF (SpeNode.GT.0) MacNodes(SpeNode,MicNod)=NoOfNodes
  END DO

  RETURN
  END
```

```
      PROGRAM ECFDefine

C This program get the problem definition from the user.
C Written by: Ken Jordan                          8/1/87
C Modified by: Mike Armstrong                          9/1/87
C       Added multiple equations.
C Modified by: Ken Jordan                              9/24/87
C       Debugged multiple equations.
C Modified by: Ken Jordan                              10/22/87
C       Added: EC Problem Report (prints parameters used to solve EC problem)
C       Added: Uses POLYFLOW Boundary Node numbers, converts to EC bdnodes
C             internally.

C ******************************************************************************
C Variables
C ******************************************************************************

C ******************************************************************************
C Subroutines
C ******************************************************************************

C ComputeCLoc          Computes the contents of the CLoc matrix.
C ComputeCorners       Computes the location of the corner points based
C                      on the boundary node numbers from polymesh and the
C                      degree of the basis functions.
C DefineBC             Allows the user to define the upper, lower electrode
C                      potential, and the type of current distribution.
C DefineECFParams
C PrintProb            Writes a report on the CRT about the FEM problem
C                      to be solved.
C SetCoef
C Titles
C WriteFEMUser                 Writes the FEM.* files for later reading by
C                      FEM2dBAND
C WriteRes()           Writes the C and CLoc variables

C ******************************************************************************
C Main Program Follows
C ******************************************************************************

      INCLUDE 'ECFNam.INC'
      INCLUDE 'ECFHdr.INC'

      CHARACTER*80 MenuTitle/'EC Problem Modification'/,
     & MenuTxt(13)/
     1 'Code  To',
     2 ' ',
     3 ' P    Print a problem report',
     4 ' ',
     5 ' F    Modify FEM Parameters',
     7 ' B    Modify Boundary Conditions',
     8 ' C    Modify Convection or Kinetics Parameters',
     9 ' N    Modify File Names',
     a ' ',
     1 ' X    Exit (saves modifications & recomputes first guess)',
     2 ' Q    Quit (forgets modifications)',
     3 ' ',
     4 ' ?   Repeats this list'/
```

```
      CHARACTER*80 MenuPrompt/'Enter a Code'/
      CHARACTER*1 MenuResps(8)/'P','F','B','C','N','X','Q','?'/,
     & DefResp/'X'/,Resp
      INTEGER NTxt/13/,NResps/8/

      INTEGER Length
      LOGICAL FileExist,Editing,Another,Reset,OldProb
      CHARACTER*80 FileName/'FEM.NAM'/

      CALL Titles

1     CALL EditCh('Names file?',FileName)

      IF (FileName.NE.' ') THEN

        INQUIRE(File=FileName,Exist=FileExist)
        IF (FileExist) THEN

          WRITE(*,*) 'Reading data from old files'
          CALL ReadNames(FileName)
          CALL ReadFEMUser
          IF (.NOT.ECFMesh) CALL UnComputeCorners
          CALL PrintProb(FileName)
          OldProb=.TRUE.

        ELSE

          OldProb=.FALSE.
          CALL Defaults

          CALL StringLength(FileName,Length)
          Length=MAX(Length,1)

          WRITE(*,*) 'Warning: File ',FileName(1:Length),
     & ' doesn''t exist.'
          WRITE(*,*) '        I am assuming that you are entering',
     & ' data for a new problem.'

        END IF

      ELSE

        OldProb=.FALSE.
        FileExist=.FALSE.
        CALL Defaults
        WRITE(*,*) 'Getting data for a new problem!'

      END IF

      IF (FileExist) THEN

        Editing=.TRUE.
        DO WHILE(Editing)

          CALL Menu(MenuTitle,MenuTxt,NTxt,MenuPrompt,
     & MenuResps,NResps,DefResp,Resp)

C P=print problem report
```

```
C F=Method, Problem Parameters
C B=Boundary Conditions
C C=Kinetics, Convection Parameters
C N=File Names Definition

C X=Exit (saves modifications)
C Q=Quit (no modifications saved)

          IF (Resp.EQ.'P') CALL PrintProb(FileName)

          IF (Resp.EQ.'F') CALL DefineECFParams(FileName)
          IF (Resp.EQ.'B') CALL DefineBC
          IF (Resp.EQ.'C') CALL DefinePar
          IF (Resp.EQ.'N') CALL DefineNames(FileName)

          IF (Resp.EQ.'X') Editing=.FALSE.
          IF (Resp.EQ.'Q') STOP 'Quit--No Modifications Saved!'

        END DO

      ELSE

        CALL DefineECFParams(FileName)

        CALL DefineBC
        CALL DefinePar

        CALL DefineNames(FileName)

      END IF

      IF (.NOT.ECFMesh) THEN

C The mesh was generated using POLYMESH

        WRITE(*,*) 'Transferring mesh from polymesh file'
        CALL ComputeCorners

C The following call transfers the mesh from the polyflow file to
C a EC file.

        CALL MeshTransfer

      END IF

C Compute the number of coefficients.  Check to make sure we have enough memory
C to solve this problem.

      NoOfCoef=NoOfEqu*NoOfNodes
      IF (NoOfCoef.GT.MaxNoOfCoef) THEN
        WRITE(*,*) 'Error: No of coefficients is too large.'
        WRITE(*,*) '          NoOfCoef=',NoOfCoef
        WRITE(*,*) '       MaxNoOfCoef=',MaxNoOfCoef
        STOP 'ECFDEFINE: Abnormal End'
      END IF

C The following call computes the matrix CLoc used to locate the
C coefficients for each equation.
```

```
      CALL ComputeCLoc

C The following call computes a first guess for the coefficients

      IF (OldProb) THEN
        CALL YorN('Reset Coefficients?',Reset)
      ELSE
        Reset=.TRUE.
      END IF

      IF (Reset) CALL SetCoef

C The following two calls store the information for later use by FEM2D

      CALL WriteNames(FileName)
      CALL WriteFEMUser

      CALL EditYorN('Another?',Another)
      IF (Another) GOTO 1

      STOP 'Normal End'
      END

C ******************************************************************** ComputeCLoc

      SUBROUTINE ComputeCLoc

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFRes.INC'
      INTEGER I,ElemCol,ElemRow,J,NodeNo,K,
     & LocNodes((MaxNoOfBasisDeg+1)**(MaxNoOfDim))
      INTEGER Upper

C The following DO-Loop Computes the CLoc (or NOP) Matrix
C (Currently set up for BasisDeg=1 or 2) from NLoc, the node numbers
C for each element.

      IF ((BasisDeg.NE.1).AND.(BasisDeg.NE.2)) THEN
        WRITE(*,*) 'Error: Degree of Basis Functions incorrect.'
        WRITE(*,*) '       BasisDeg=',BasisDeg
        STOP 'In Routine: ComputeCLoc'
      END IF

      DO I=1,NoOfElems
        CALL GetUpper(ElemType(I),Upper)
        DO K=1,Upper
          DO J=1,NoOfEqu

C This is the most effifient way to store coefficients.
C Coefficients are numbers C1=node1,eq1, c2=node1,eq2,...
              CALL ComputeCoefLoc(NLoc(I,K),J,CLoc(J,I,K))
D             WRITE(*,*) 'NLoc',I,K,Nloc(I,K)
D               WRITE(*,*) 'CLoc',J,I,K,CLoc(J,I,K)

C For Coefficients stored as C1=node1, eq1; c2=node2, eq1,...
C             CLoc(J,I,K)=NLoc(I,K)-(J-1)*NoOfNodes
```

```
          END DO
        END DO
      END DO

      RETURN
      END

C ************************************************************** ComputeCorners

      SUBROUTINE ComputeCorners

C This subroutine computes the corner points for the problem based
C on the corner points from the POLYFLOW Mesh, and on the BasisDeg chosen
C by the user.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFbc.INC'

      INTEGER Equ,Crnr

      IF ((BasisDeg.EQ.1).OR.(BasisDeg.EQ.3)) THEN
        DO Equ=1,NoOfEqu
          DO Crnr=1,NoOfCorners(Equ)
            Corners(Equ,Crnr)=(Corners(Equ,Crnr)+1)/2
          END DO
        END DO
      END IF

      RETURN
      END

C ******************************************************************** Defaults

      SUBROUTINE Defaults

      INCLUDE 'ECFNam.INC'
      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'

      MeshHdrFile='EC.HDR'
      MeshFile='EC.MSH'
      BCFile='EC.BC'
      ParFile='EC.PAR'

     ResFile='EC.RES'
      PlotFile='EC.ISO'
      SlopeFile='EC.SLP'

      NoOfEqu=1
      NoOfDim=2
      BasisDeg=1
      Tolerance=0.00001
      NoOfGaussPts=4
      MaxNoOfITer=10

      Debugging=.FALSE.
      Convection=.FALSE.
      PolyFlow=.TRUE.
```

```
      Guess=.FALSE.

      Nonnegative=.FALSE.

      RETURN
      END

C ********************************************************************** DefineBC

      SUBROUTINE DefineBC

C This subroutine allows the user to define the type of current distribution
C and the voltage at the upper and lower plates (Vu,Vl)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFbc.INC'

      INTEGER I,Ip1,Equ,J
      INTEGER Corn(MaxNoOfCorners)

      CHARACTER*80 Prompt,Dummy
      LOGICAL Inputting,EntKin

      CHARACTER*80 BCMenuTitle/'Boundary Condition Selection'/,
     & BCMenuTxt(13)/
     1 'Code  Boundary Condition',
     2 ' ',
     3 ' P   Primary Current Distribution ',
     4 '       (Surface is equipotential)',
     5 ' S   Secondary Current Distribution',
     6 '       (Current depends on voltage near surface)',
     7 ' I   Insulator',
     8 '       (No current through surface)',
     9 ' E   Essential',
     a '       (Potential at surface set by external subroutine',
     1 ' N   Natural',
     2 '       (Current at surface set by external subroutine',
     3 ' ?   Repeats this list'/
      CHARACTER*80 MenuPrompt
      CHARACTER*1 BCMenuResps(6)/'P','S','I','E','N','?'/,
     & DefResp/'?'/,UsrResp
      INTEGER BCNTxt/13/,BCNResps/6/

      CHARACTER*80 KIMenuTitle/'Kinetics Selection'/,
     & KIMenuTxt(6)/
     1 'Code  Electrode Kinetics Selected',
     2 ' ',
     3 ' BV   Butler Volmer',
     4 ' LI   Linear',
     5 ' TA   Tafel',
     6 ' ?   Repeats this list'/
      CHARACTER*2 KIMenuResps(4)/'BV','LI','TA','?'/,KIDefResp/'?'/,
     & KIUsrResp
      INTEGER KINTxt/6/,KINResps/4/

      DO Equ=1,NoOfEqu
        EntKin=.FALSE.
```

```
      WRITE(*,*) ' '
      WRITE(*,*) 'BC Conditions-----------------------------------'

C The following statements get the corner points from the user for
C eqution #L

      IF (NoOfEqu.GT.1) WRITE(*,*) 'For Equation ',Equ
      WRITE(*,*) ' '

      DO J=1,NoOfCorners(Equ)
        Corn(J)=Corners(Equ,J)
      END DO

      CALL CodeI(Corn,NoOfCorners(Equ),Dummy)

      IF (ECFMesh) THEN
        WRITE(*,*) 'Mesh generated by ECFMESH'
      ELSE
        WRITE(*,*) 'Mesh generated by POLYMESH'
      END IF

      CALL EditCh(
     & 'Enter Corner Points (boundary node #s)?',Dummy)

      CALL DecodeI(Dummy,Corn,NoOfCorners(Equ),MaxNoOfCorners)
      DO J=1,NoOfCorners(Equ)
        Corners(Equ,J)=Corn(J)
      END DO

C The following statements and DO-loop allow the user to select the
C current distribution. (Primary or Secondary)

      DO I=1,NoOfCorners(Equ)

        CALL IncrementI(I,Ip1,1,NoOfCorners(Equ))

        WRITE(MenuPrompt,FMT=27) I,Equ,
     & Corners(Equ,I),Corners(Equ,Ip1)
27      FORMAT('Boundary Condition for side ',I2,
     & ' Equ ',I2,' (Nodes ',I3,' to ',I3,')')

        DefResp=BCs(Equ,I)
        IF (DefResp.EQ.' ') DefResp='?'

        Inputting=.TRUE.
        DO WHILE (Inputting)
          CALL Menu(bcMenuTitle,bcMenuTxt,bcNTxt,MenuPrompt,
     & bcMenuResps,bcNResps,DefResp,UsrResp)
          Inputting=(UsrResp.EQ.'?')
        END DO
        BCs(Equ,I)=UsrResp

C The following statements get whatever parameters are required for the
C boundary condition.

        IF ((BCs(Equ,I).EQ.'S').OR.(BCs(Equ,I).EQ.'P')) THEN
          WRITE(Prompt,FMT=30) I,Equ,BCs(Equ,I)
30        FORMAT('Voltage applied to side ',I2,
```

```
     & ' Equ ',I2,' (BC=',A1,')')
                CALL EditDP(Prompt,BCPar(Equ,I))

                IF (BCs(Equ,I).EQ.'S') EntKin=.TRUE.

              ELSE IF (BCs(Equ,I).EQ.'E') THEN

                WRITE(Prompt,FMT=31) I,Equ,BCs(Equ,I)
31              FORMAT('Essential Parameter? (side',I2,
     & ' equ ',I2,') (BC=',A1,')')
                CALL EditDP(Prompt,BCPar(Equ,I))

              ELSE IF (BCs(Equ,I).EQ.'N') THEN

                WRITE(Prompt,FMT=32) I,Equ,BCs(Equ,I)
32              FORMAT('Natural Parameter? (side',I2,
     & ' equ ',I2,') (BC=',A1,')')
                CALL EditDP(Prompt,BCPar(Equ,I))

              ELSE IF (BCs(Equ,I).EQ.'I') THEN
                BCPar(Equ,I)=0
              END IF

            END DO

C Get the kinetics code to use for this problem.

            IF (EntKin) THEN
              WRITE(MenuPrompt,Fmt=10) Equ
10            FORMAT('Kinetics Code? (Eqn ',I2,')')

              KIDefResp=Kinetics(Equ)
              IF (KiDefResp.EQ.' ') KIDefResp='?'

              Inputting=.TRUE.
              DO WHILE (Inputting)
                CALL Menu(KIMenuTitle,KIMenuTxt,KINTxt,
     & MenuPrompt,KIMenuResps,KINResps,KIDefResp,KIUsrResp)
                Inputting=(KIUsrResp.EQ.'?')
              END DO
              Kinetics(Equ) =KiUsrResp
            ELSE
              Kinetics(Equ)= ' '
            END IF

          END DO

          RETURN
          END

C ******************************************************************** DefineECFParams

          SUBROUTINE DefineECFParams(FileName)

          INCLUDE 'ECFHdr.INC'
          INCLUDE 'ECFGauss.INC'

          CHARACTER*(*) FileName
```

```
      CHARACTER*3 Code
      CHARACTER*80 Dummy
      INTEGER Length

C The following CALL obtains the tolerance for testing convergence of the
C Newton-Raphson iterations.

      OPEN(UNIT=1,STATUS='NEW',FORM='FORMATTED',FILE='ECFDEF.TMP')

      IF (Convection) THEN
        IF (PolyFlow) THEN
          Code='PFC'
        ELSE
          Code='EXC'
        END IF
      ELSE IF (Binary) THEN
        Code='BIN'
      ELSE IF (Supporting) THEN
        Code='SUP'
      ELSE
        Code='LAP'
      END IF

      WRITE(1,*) '''',Code,''' '' Problem Code:'
      WRITE(1,*) '              '' PFC=polyflow convective-diffusion'
      WRITE(1,*) '              '' EXC=external convective-diffusion'
      WRITE(1,*) '              '' BIN, SUP=',
     & 'binary, or supporting electrolyte'
      WRITE(1,*) '              '' LAP=Laplace''s Equation',
     & ' for potential or conc.'
      WRITE(1,*) ' '

      WRITE(1,*) BasisDeg,'  '' Degree of Basis Functions (0-2)'
      WRITE(1,*) NoOfGaussPts,
     & ' '' No. of Gaussian Integration Points ',
     & '(1-',MaxNoOfGaussPts,')'
      WRITE(1,*) NoOfDim,'  ''Number of dimensions (2)'
      WRITE(1,*) NoOfEqu,'  ''Number of Equations (1-)'
      WRITE(1,*) MaxNoOfIter,'        ''Maximum number of NR Iterations'
      WRITE(1,*) Tolerance,'  '' Convergence Tolerance'
      WRITE(1,*) ' '

      WRITE(1,*) Debugging,' ''T if debugging'
      WRITE(1,*) Guess,'      '' T for external first guess routine'
      WRITE(1,*) Nonnegative,
     & ' '' T for nonegative coeff. constraint'

      WRITE(1,*) ' '
      CALL StringLength(FileName,Length)
      WRITE(1,*) 'FEM Method Parameters-------------------------------'
      WRITE(1,*) 'File: ',FileName(1:Length)
      CLOSE(1)

      CALL LIB$SPAWN('EDIT ECFDEF.TMP')

      OPEN(UNIT=1,STATUS='OLD',FORM='FORMATTED',FILE='ECFDEF.TMP')

      READ(1,*) Code
```

```
      READ(1,10) Dummy(1:1)
10    FORMAT(A1)
      READ(1,10) Dummy(1:1)
      READ(1,10) Dummy(1:1)
      READ(1,10) Dummy(1:1)
      READ(1,10) Dummy(1:1)

      IF (Code.EQ.'PFC') THEN
        Convection=.TRUE.
        Polyflow=.TRUE.
        Binary=.FALSE.
        Supporting=.FALSE.
      ELSE IF (Code.EQ.'EXC') THEN
        Convection=.TRUE.
        Polyflow=.FALSE.
        Binary=.FALSE.
        Supporting=.FALSE.
      ELSE IF (Code.EQ.'BIN') THEN
        Convection=.FALSE.
        Binary=.TRUE.
        Supporting=.FALSE.
      ELSE IF (Code.EQ.'LAP') THEN
        Convection=.FALSE.
        Binary=.FALSE.
        Supporting=.FALSE.
      ELSE IF (Code.EQ.'SUP') THEN
        Convection=.FALSE.
        Binary=.FALSE.
        Supporting=.TRUE.
      ELSE
        WRITE(*,*) 'Error: Unknown problem code.'
      END IF

      READ(1,*) BasisDeg

      IF ((BasisDeg.LT.0).OR.(BasisDeg.GT.2)) THEN
        WRITE(*,*) 'Error: Degree of Basis Functions is out of range'
      END IF

      READ(1,*) NoOfGaussPts
      IF ((NoOfGaussPts.GT.MaxNoOfGaussPts)
  & .OR.(NoOfGaussPts.LT.1)) THEN
        WRITE(*,*) 'Error: No. of gauss pts is out of range.'
      END IF

      READ(1,*) NoOfDim
      READ(1,*) NoOfEqu

      READ(1,*) MaxNoOfIter
      READ(1,*) Tolerance
      READ(1,10) Dummy(1:1)

      READ(1,*) Debugging
      READ(1,*) Guess
      READ(1,*) Nonnegative

      CLOSE(1)
```

```
      RETURN
      END

C ******************************************************************** DefineNames

      SUBROUTINE DefineNames(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFNam.INC'
      CHARACTER*(*) FileName

C The following lines put in default names

      WRITE(*,*) 'File Names Selection----------------------------'

      WRITE(*,*) ' '
      WRITE(*,*) '****** Mesh Source File ******'
      WRITE(*,*) ' '

      CALL GetFileName(
     & 'PolyFlow Mesh File to transfer to EC Mesh?',
     & MeshSrcFile)

      WRITE(*,*) ' '
      WRITE(*,*) ' ****** Input Files for ECCompute ******'
      WRITE(*,*) ' '

      CALL EditCh('Names file?',FileName)
      WRITE(*,*) ' '

      CALL EditCh('EC Mesh Header File?',MeshHdrFile)
      CALL EditCh('EC Mesh File?',MeshFile)
      CALL EditCh('BC File?',BCFile)
      CALL EditCh('Miscellaneous Parameters File?',ParFile)

      IF (Convection.AND.PolyFlow) THEN

        WRITE(*,*) ' '
        WRITE(*,*) '****** Velocity Profile Files From POLYFLOW ',
     & 'Program ******'
        WRITE(*,*) ' '

        CALL GetFileName('Polyflow Mesh Filename?',PFMeshFile)
        CALL GetFileName('Polyflow Results FileName?',PFCoefFile)

      END IF

      WRITE(*,*) ' '
      WRITE(*,*) '****** Output Files from ECCompute ******'
      WRITE(*,*) ' '

      CALL EditCh('Results File?',ResFile)
      WRITE(*,*) ' '
      WRITE(*,*) '****** Graphics Files from FEM2dPlot ******'
      WRITE(*,*) ' '
      CALL EditCh('Plot File?',PlotFile)
      CALL EditCh('Slope File?',SlopeFile)
```

```
      RETURN
      END

C ********************************************************************** DefinePar

      SUBROUTINE DefinePar

C This subroutine will allow the user to define the various parameters used
C in the problem.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFPar.INC'
      INCLUDE 'ECFbc.INC'

      LOGICAL Kinet
      INTEGER I,Equ

      Kinet=.FALSE.
      DO Equ=1,NoOfEqu
        DO I=1,NoOfCorners(Equ)
          Kinet=Kinet.OR.(BCs(Equ,I).EQ.'S')
        END DO

        IF (Kinet) THEN

          IF (.NOT.Binary) THEN
            WRITE(*,*) ' '
            WRITE(*,*)' WagnerLin = Kappa R T / L F i0'
          ELSE

C For Binary or Supporting cases.

            WRITE(*,*) ' '
            WRITE(*,*)' WagnerLin = 2 F D1 Cref / L i0'
          END IF
          CALL EditDP('WagnerLin?',WagnerLin)
          CALL EditDP('Anodic Transfer Coefficient',aA)
          CALL EditDP('Cathodic Transfer Coefficient',aC)

          IF ((Binary).OR.(Supporting)) THEN
            CALL EditDP('Integration Constant (binary)?',BinaryK2)
          END IF
          IF (Supporting) THEN
            CALL EditDP('Integration Constant (supporting)?',
     & SupportingK3)
          END IF
        END IF

        IF (Convection) THEN
          CALL EditDP('Peclet number?',Peclet)
        END IF
      END DO

      RETURN
      END

C ********************************************************************** MeshTransfer
```

```
      SUBROUTINE MeshTransfer

C This subroutine will generate NLoc(,) and Nodes(,) from data in a polyflow
C mesh file generated by POLYMESH

      INCLUDE 'ECFNam.INC'
      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'POLYMesh.INC'

      INTEGER I,NodeMax,Zero/0/,Nv,J,Stp
      LOGICAL Triangular

      CALL ReadPFMesh(MeshSrcFile)

C Transfer the boundary node information

      NoOfBDNodes=0

      IF ((BasisDeg.EQ.1).OR.(BasisDeg.EQ.3)) THEN
        Stp=2
      ELSE IF (BasisDeg.EQ.2) THEN
        Stp=1
      END IF

C Check Limits too see if max no of ... needs to be set higher.

      IF (PolyNBd/Stp.GT.MaxNoOfNodes) THEN
        WRITE(*,*) 'Error: too many nodes.  Problem can''t be'
        WRITE(*,*) '       solved by present program configuration'
        WRITE(*,*) '       Max No Of Nodes=',MaxNoOfNodes
        WRITE(*,*) '       Cur No Of Nodes=',PolyNBd/Stp
        STOP '(ECFDEFINE) In Routine: MeshTransfer'
      END IF

      IF (PolyNElem.GT.MaxNoOfNodes) THEN
        WRITE(*,*) 'Error: Too many mesh elements.  Either decrease',
     & ' the size of the mesh, '
        WRITE(*,*) '       or increase MaxNoOfNodes in ECFHDR.INC'
        WRITE(*,*) '            PolyNElem=',PolyNElem
        WRITE(*,*) '          MaxNoOfNodes=',MaxNoOfNodes
        STOP '(ECFDEFINE) In Routine: MeshTransfer'
      END IF

      DO I=1,PolyNbd,Stp
        NoOfBDNodes=NoOfBDNodes+1
        BdNodes(NoOfBDNodes)=PolyBDNod(I)
      END DO

C Find the maximum node number and Transfer the node numbers to NLoc

      NodeMax=0
      DO I=1,PolyNElem
        IF (PolyNod(9,I).EQ.Zero) THEN
          Triangular=.TRUE.
        ELSE
          Triangular=.FALSE.
        END IF
```

```
          IF (BasisDeg.EQ.1) THEN

            IF (Triangular) THEN
              Nv=3
            ELSE
              Nv=4
            END IF

          ELSE IF (BasisDeg.EQ.2) THEN

            IF (Triangular) THEN
              Nv=6
            ELSE
              Nv=9
            END IF

          ELSE

            Nv=0
            WRITE(*,*) 'Error: Routine not written for BasisDeg=',
     & BasisDeg
            WRITE(*,*) '       BasisDeg must be 1 or 2'
            STOP 'In routine: MeshTransfer'

          END IF

C Make NLoc

          DO J=1,Nv
            IF (NodeMax.LT.PolyNod(J,I)) NodeMax=PolyNod(J,I)
            NLoc(I,J)=PolyNod(J,I)
          END DO

        END DO

C Transfer the node locations from POLYX, POLYY to Nodes(,)

        IF (NodeMax.GT.MaxNoOfNodes) THEN
          WRITE(*,*) 'Error: Too many nodes.  Number of Nodes = ',NodeMax
          WRITE(*,*) '                    Max Number of Nodes = ',MaxNoOfNodes
          WRITE(*,*) '       Either decrease mesh, or increase MaxNoOfNodes',
     & ' in ECFHDR.INC'
          WRITE(*,*) '       and recompile, relink ECF programs.'
          STOP 'ECFDEFINE=In Routine: MeshTransfer-Abnormal End'
        END IF

        DO I=1,NodeMax
          Nodes(I,1)=PolyX(I)
          Nodes(I,2)=PolyY(I)
        END DO

        NoOfElems=PolyNElem
        NoOfNodes=NodeMax

        RETURN
        END
```

```
C ********************************************************************** PrintProb

      SUBROUTINE PrintProb(NamesFile)

C This subroutine outputs a report on the problem defined by the user.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFbc.INC'
      INCLUDE 'ECFPar.INC'
      INCLUDE 'ECFNam.INC'

      INTEGER I,L,J,Equ,Length
      INTEGER Corn(MaxNoOfCorners)
      LOGICAL Kinet
      CHARACTER*80 Dummy,Output
      CHARACTER*(*) NamesFile

      CALL PrintProbHeader

      WRITE(*,230) NoOfElems,NoOfNodes,Nlc,NoOfBDNodes,Nuc

230   FORMAT(X,7X,'Number of Elements: ',I6,/,
     & X,10X,'Number of Nodes: ',I6,4X,'Codiagonals:  lower=',I4,
     & /,X,' Number of Boundary Nodes: ',I6,18X,'upper=',I4)

C 1234567                    1234          12
C       Number of Elements: iiiiii
C          Number of Nodes: iiiiii    Codiagonals:  lower=iiii
C  Number of Boundary Nodes: iiiiii                 upper=iiii
C 1                         123456789a12345678

C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

      WRITE(*,10)
10    FORMAT(X,58('-'),' Boundary Conditions')

C 123456789a123456789b123456789c123456789d123456789e12345678
C ------------------------------------------------------ Boundary Conditions

C                    123456789a123456789b123456789c123456789d123456789e1
C Equation ii     iii Corners: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
C 123456789a1234567
C                Side iii: BC (a) BC Parm (dd.dddddd e+dd) Kinetics (aa)

      DO L=1,NoOfEqu

        DO J=1,NoOfCorners(L)
          Corn(J)=Corners(L,J)
        END DO

        Dummy=' '
        CALL CodeI(Corn,NoOfCorners(L),Dummy)
        IF (NoOfEqu.EQ.1) THEN
          WRITE(*,21) NoOfCorners(1),Dummy
21        FORMAT(X,'          ',2X,3X,I3,' Corners: ',A51)
        ELSE
          WRITE(*,20) L,NoOfCorners(L),Dummy
```

```
20          FORMAT(X,'Equation ',I2,3X,I3,' Corners: ',A51)
          END IF

          DO I=1,NoOfCorners(L)
            IF (I.EQ.L) THEN
              IF (Kinetics(L).NE.' ') THEN
                WRITE(*,30) I,BCs(L,I),BCPar(L,I),Kinetics(L)
              ELSE
                WRITE(*,30) I,BCs(L,I),BCPar(L,I)
              END IF
            ELSE
              WRITE(*,30) I,BCs(L,I),BCPar(L,I)
            END IF
30          FORMAT(X,17X,'Side ',I3,': BC (',A1,')   Param. (',G14.7,
   & ')',:,'  Kinetics (',A2,')')
          END DO
        END DO

C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        Kinet=.FALSE.
        DO Equ=1,NoOfEqu
          DO I=1,NoOfCorners(Equ)
            Kinet=Kinet.OR.(BCs(Equ,I).EQ.'S')
          END DO
        END DO

C 123456789a123456789b123456789c123456789d123456789e123
C ------------------------------------------------- Miscellaneous Parameters
C         123456789a12              1234              1234
C Constants:          R=ddddddddddddddd   F=ddddddddddddddd   T=ddddddddddddddd
C Transport Parameters: K=ddddddddddddddd  Pe=ddddddddddddddd
C Kinetic Parameters:  i0=ddddddddddddddd  aA=ddddddddddddddd  aC=ddddddddddddddd

        IF (Kinet.OR.Convection) THEN
          WRITE(*,100)
100       FORMAT(X,53('-'),' Miscellaneous Parameters')
120       FORMAT(X,'Transport Parameters:Pe=',G14.7)
130       FORMAT(X,'Kinetic Parameters:  Wa=',G14.7,'   aA=',G14.7,
   & '  aC=',G14.7)
135       FORMAT(X,'Kinetic Par.:        k2=',G14.7)

          IF (Convection) THEN
            WRITE(*,120) Peclet
          ELSE IF (Binary) THEN
            WRITE(*,130) WagnerLin,aA,aC
            WRITE(*,135) Binaryk2
          ELSE
            WRITE(*,130) WagnerLin,aA,aC
          END IF

        END IF

C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

C 123456789a123456789b123456789c123456789d123456789e123456789f1234567
C ------------------------------------------------------------------ File Names
```

```
      WRITE(*,300)
300   FORMAT(X,67('-'),' File Names')

C          123456789a123456789b123456789c123456789          123456789a12345
C Input Files:                                  Names File: AAAAAAAAAAAAAAAA

      WRITE(*,310) NamesFile
310   FORMAT(X,'Input Files:',34X,'Names File: ',A20)
C 123                 123456789a12345

C   (PolyFlow Mesh Source: AAAAAAAAAAAAAAAA)

      IF (.NOT.ECFMesh) THEN
        CALL StringLength(MeshSrcFile,Length)
        WRITE(*,320) MeshSrcFile(1:Length)
320     FORMAT(X,'   (PolyFlow Mesh Source: ',A<Length>,')')
      END IF

C 123456789a123                     1234
C          Mesh Header: AAAAAAAAAAAAAAAA    Mesh: AAAAAAAAAAAAAAAA

      WRITE(*,330) MeshHdrFile,MeshFile
330   FORMAT(X,13X,'Mesh Header: ',A20,4X,'Mesh: ',A20)

C 12345
C     Boundary Conditions: AAAAAAAAAAAAAAAA
C Miscellaneous Parameters: AAAAAAAAAAAAAAAA
C

      WRITE(*,340) BCFile,ParFile
340   FORMAT(X,5X,'Boundary Conditions: ',A20,/,X,
     & 'Miscellaneous Parameters: ',A20,/)

C Velocity Profile Files:
C            123456789a123456789b123                123456789a123456789b
C Polyflow Mesh: AAAAAAAAAAAAAAAAAAAA  Polyflow Results: AAAAAAAAAAAAAAAAAAAA
C

      IF (Convection.AND.PolyFlow) THEN
        WRITE(*,350) PFMeshFile,PFCoefFile
350     FORMAT(X,'Velocity Profile Files:',/,X,
     & 'Polyflow Mesh File: ',A20,3X,'Polyflow Results: ',A20,/)
      END IF

C Output Files:
C 123456
C     Results: AAAAAAAAAAAAAAAAAAAA
C 123456789                         123
C        Plot: AAAAAAAAAAAAAAAAAAAA  Slope: AAAAAAAAAAAAAAAAAAAA

      WRITE(*,360) ResFile,PlotFile,SlopeFile
360   FORMAT(X,'Output Files: ',/,X,
     & 6X,'Results: ',A20,/,X,
     & 9X,'Plot: ',A20,3X,'Slope: ',A20)

      RETURN
      END
```

```
C ******************************************************************** SetCoef

      SUBROUTINE SetCoef

C This subroutine sets up the values of the potential for the first guess.
C (Note, currently the subroutine will only handle 1st
C  order basis functions)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFRes.INC'
      INCLUDE 'ECFbc.INC'

      REAL*8 dV,Avg,X(MaxNoOfDim)
      INTEGER J,NodeNo,K,Equ,Side,Sidep1,Elem,Upper

      IF ((BasisDeg.GT.2).OR.(BasisDeg.LT.0)) THEN
        WRITE(*,*) 'Warning: illegal degree of basis functions'
        WRITE(*,*) '         BasisDeg=',BasisDeg
        STOP 'In routine: SetCoef'
      END IF

      DO Equ=1,NoOfEqu

        IF (.NOT.Guess) THEN

            Avg=0.0
            DO Side=1,NoOfCorners(Equ)
              Avg=Avg+BCPar(Equ,Side)
            END DO
            Avg=Avg/(NoOfCorners(Equ)-2)

        END IF

C The following do-loop makes the first guess for coefficients

        IF ((Binary).OR.(Supporting)) THEN
          Avg=ABS(Avg)
        END IF

        DO Elem=1,NoOfElems
          CALL GetUpper(ElemType(Elem),Upper)
          DO J=1,Upper

            IF (.NOT.Guess) THEN
              C(CLoc(Equ,Elem,J))=Avg
            ELSE
              DO K=1,NoOfDim
                X(K)=Nodes(NLoc(Elem,J),K)
              END DO
              CALL FirstGuess(1,Elem,X,C(CLoc(Equ,Elem,J)))
            END IF

          END DO
        END DO
    END DO

C The following DO-Loop sets the voltage of the plate for primary
```

```
C current distribution.

      DO Equ=1,NoOfEqu
         DO Side=1,NoOfCorners(Equ)

            CALL IncrementI(Side,Sidep1,1,NoOfCorners(Equ))

            IF (BCs(Equ,Side).EQ.'P') THEN

              DO J=Corners(Equ,Side),Corners(Equ,Sidep1)
                C((BDNodes(J)-1)*NoOfEqu+Equ)=BCPar(Equ,Side)
              END DO

            ELSE IF (BCs(Equ,Side).EQ.'E') THEN

              IF (Corners(Equ,Side).LE.Corners(Equ,Sidep1)) THEN

                DO J=Corners(Equ,Side),Corners(Equ,Sidep1)
                  DO K=1,NoOfDim
                    X(K)=Nodes(BDNodes(J),K)
                  END DO
                  CALL Essential(Side,1,X,BCPar(Equ,Side),
     & C((BDNodes(J)-1)*NoOfEqu+Equ))
                END DO

              ELSE

                DO J=Corners(Equ,Side),NoOfBDNodes
                  DO K=1,NoOfDim
                    X(K)=Nodes(BDNodes(J),K)
                  END DO
                  CALL Essential(Side,1,X,BCPar(Equ,Side),
     & C((BDNodes(J)-1)*NoOfEqu+Equ))
                END DO

                DO J=1,Corners(Equ,Sidep1)
                  DO K=1,NoOfDim
                    X(K)=Nodes(BDNodes(J),K)
                  END DO
                  CALL Essential(Side,1,X,BCPar(Equ,Side),
     & C((BDNodes(J)-1)*NoOfEqu+Equ))
                END DO

              END IF

            END IF
         END DO
      END DO

        RETURN
        END

C ************************************************************************** Titles

        SUBROUTINE Titles

C This subroutine prints the run-time-message that lets the user know
C the name of the program being run, etc.
```

```
      WRITE(*,*) 'Program ECFDefine'
      WRITE(*,*) 'Version 2'
      WRITE(*,*) ' '
      WRITE(*,*) 'This program allows the user to define the type'
      WRITE(*,*) 'of problem the FEM program will solve'
      WRITE(*,*) ' '

      RETURN
      END

C ************************************************************** UnComputeCorners

      SUBROUTINE UnComputeCorners

C This subroutine computes the corner points of the polyflow mesh based
C on the corner points for the problem and on the BasisDeg chosen
C by the user.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFbc.INC'

      INTEGER Equ,Crnr

      IF ((BasisDeg.EQ.1).OR.(BasisDeg.EQ.3)) THEN
        DO Equ=1,NoOfEqu
          DO Crnr=1,NoOfCorners(Equ)
            Corners(Equ,Crnr)=2*Corners(Equ,Crnr)-1
          END DO
        END DO
      END IF

      RETURN
      END

c *
```

```
      PROGRAM ECFPrePro

C This program computes the bandwidth for the EC problem.

C Written by: Ken Jordan
C Modified by: Mike Armstrong
C Debugged by: Ken Jordan

C ******************************************************************************
C VARIABLES
C ******************************************************************************

      INCLUDE 'ECFNam.INC'
      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'

      CHARACTER*72 FileName
      INTEGER Elm,I,J,BandWidth,MaxN,MinN,MaxDiff,Upper

C ******************************************************************************
C Main Program
C ******************************************************************************

C The following call reads the values of variables that define the problem and
C solution technique (BasisDeg, NoOfElems, NoOfNodes, NoOfXElems, NoOfYElems,
C Tolerance)

      CALL GetFileName('Names file?',FileName)
      CALL ReadNames(FileName)

      CALL ReadFEMUser

C New routine to calculate bandwidth based on max difference between node
C number on each element.

      MaxDiff=1

      DO Elm=1,NoOfElems

        MaxN=NLoc(Elm,1)
        MinN=MaxN

        CALL GetUpper(ElemType(Elm),Upper)

        DO I=2,Upper
          MaxN=MAX(MaxN,NLoc(Elm,I))
          MinN=MIN(MinN,NLoc(Elm,I))
        END DO

        MaxDiff=MAX(MaxDiff,MaxN-MinN)

      END DO

      Nuc=MaxDiff
      Nlc=MaxDiff
      Bandwidth=Nlc+Nuc+1

      WRITE(*,*) 'Number of Lower Codiagonals=',Nlc
```

```
WRITE(*,*) 'Number of Upper Codiagonals=',Nuc
WRITE(*,*) '   Bandwidth=',Bandwidth
WRITE(*,*) 'Matrix Order=',NoOfCoef,',',NoOfCoef

CALL WriteMeshHdr(MeshHdrFile)

STOP 'Normal End'
END
```

```
      PROGRAM ECFCompute

C This program uses the galerkin method in two dimensions to
C solve migration-diffusion problems.

C The differential equation is:

C   d^2Phi   d^2Phi
C   ------ + ------ - Pe (Vx dPhi/dX + Vy dPhi/dY) = 0
C    dy^2     dx^2

C with the following possible boundary conditions,

C       Constant Potential
C       Arbitrary Potential Profile

C       Insulator (Slope=0)
C       Constant Slope
C       Tafel, Linear, or BV Kinetics (Slope dependent on overpotential)

C Written by: Ken Jordan                4/2/86

C Modified                              8/3/87

C ***************************************************************************
C Variables
C ***************************************************************************

C Converged     .TRUE. when solution has converged.
C FileName      The name of the names file. (Contains file names
C               for reading remaining data. E.g. HULL.NAM)

C ***************************************************************************
C Subroutines
C ***************************************************************************

C                                                                   Internal

C AdjustForKinetics     Computes the BC for secondary current distribution.
C AdjustForKnown        Resets R and RJacobian for known coefficients.
C AssembleBC            Adjusts the R, RJacobian matrices for the
C                       boundary conditions.
C AssembleMatrices      Assembles the R, RJacobian matrices.
C VecNormDP             Compares 2-norm of dC to Tolerance to check for
C                       convergence
C ComputeDelPotDotNorm          Computes the normal wall current (used in secondary
C                       current distribution boundary conditions.)
C ComputeIsoPara        Given the Element number, and PhiZeta, computes
C                       XZeta, and the determinant of the jacobian.
C ComputePot            Computes the value of the potential and the potential
C                       derivative.
C NewtonRaphson                 Performs the newton-raphson technique to solve the
C                       differential equation.
C PrintCLoc             Prints the contents of the CLoc matrix.
C PrintMat              Prints the Jacobian Matrix, and the Residual Vector
C PrintNodes            Prints the node locations
C PrintResults          Prints the coefficients and parameter settings used
C                       to run problem.
```

```
C UpdateC              Adds dC to C and stores result in C
C ZeroRJ               Zeros the R,RJacobian,dRdP vectors.

C                                                          External

C Compute1DPhis              Computes the value of the local weighting functions
C                      Phi and dPhi/dZeta() (Zeta() is the local variable) in
C                      1 dimension.
C Compute2DPhis              Computes the value of the local weighting functions
C                      and derivatives for 2D.
C GaussPtsWts          Generates the Gauss Pts and Wts in two vectors for
C                      1-6 Gaussian Quadrature integration points.

C ******************************************************************************
C Files
C ******************************************************************************

C *.BC Contains the boundary conditions
C *.CMP Comparison between two slope files.
C *.HDR        Contains problem header information. (i.e. No of coefficients,
C       Number of dimensions, Number of equations...)
C *.ISO Contains the data needed for isopotential plots.
C *.MSH        Contains the mesh and node locations and boundary node locations
C *.NAM        Contains the names of the files to use in solving the problem
C       E.g. *.NAM defines the filenames for *.bc, *.hdr, *.par, *.msh,
C       *.msh
C *.PAR        Contains parameter information. E.g. Kinetic parameters such
C       as the exchange current density, the transfer coefficients...
C *.RESContains the Coefficients and CLoc(,,)
C *.SLP Contains slope at the boundary

C ******************************************************************************
C Main Program Follows
C ******************************************************************************

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFNam.INC'
      INCLUDE 'ECFGauss.INC'

      LOGICAL Converged
      CHARACTER*80 FileName
      INTEGER NoOfIterations,TotalIterations

*** IMSL workspace for DLSARB. The workspace is based on N, NLCA, and
***       NUCA, where N = 377, NLCA = 27, and NUCA = 27 are given.

      COMMON /WORKSP/ RWKSP
      REAL RWKSP(3343481)
      CALL IWKIN(3343481)

C The following call reads the values of variables that
C define the problem and solution technique (BasisDeg, NoOfElems,
C NoOfNodes,NoOfXElems,NoOfYElems,Tolerance)

      CALL GetFileName('Names file?',FileName)

      IF (FileName.EQ.' ') THEN
        WRITE(*,*) 'Error: No file name entered.'
```

```
      STOP 'In Program: ECFCompute'
    END IF

    CALL ReadNames(FileName)

    CALL ReadFEMUser

    IF (Convection.AND.PolyFlow) THEN
      CALL ReadPFMesh(PFMeshFile)
      CALL ReadPFCoef(PFCoefFile)
    END IF

C The following call obtains the Gauss Points and Weights for
C integrals over the interval 0 to 1.

    CALL GaussPtsWts(NoOfGaussPts,Pts,Wts)
    CALL TriPtsWts

    IF (Debugging) THEN
      CALL PrintNodes
      CALL PrintCLoc
    END IF

    TotalIterations=0

    CALL NewtonRaphson(NoOfIterations,Converged)

    CALL WriteRes(ResFile)

    IF (Converged) THEN

      TotalIterations=TotalIterations+NoOfIterations
      IF (Debugging) THEN
        CALL PrintNodes
        CALL PrintResults(TotalIterations,TotalIterations)
      END IF

    ELSE

      STOP 'Abnormal End: Not Converged, try continuation?'

    END IF

    STOP 'Normal End'
    END

C ************************************************************ AdjustForKinetics

    SUBROUTINE AdjustForKinetics(Nodals,Phi1D,DelPotDotNorm,
  & DelPotDotNormPot,WtsJx,dR)

    INCLUDE 'ECFHdr.INC'
    INCLUDE 'ECFRes.INC'
    INCLUDE 'ECFCal.INC'

    INTEGER Nodals(*),BandCol
    REAL*8 Phi1D(*),WtsJx,dR,DelPotDotNorm,DelPotDotNormPot
```

```
      INTEGER LocRow,LocCol

C The following statements and DO-Loops adjust the residual and jacobian
C for the boundary conditions on a plate when there are kinetics.

      DO LocRow=1,BasisDeg+1

        R(Nodals(LocRow))=R(Nodals(LocRow))-WtsJx*
     & DelPotDotNorm*Phi1D(LocRow)*dR

        DO LocCol=1,BasisDeg+1
          CALL ComputeBandCol(Nodals(LocRow),Nodals(LocCol),BandCol)
          RJacobian(BandCol,Nodals(LocRow))=
     & RJacobian(BandCol,Nodals(LocRow))
     & +WtsJx*Phi1D(LocRow)*Phi1D(LocCol)*DelPotDotNormPot
     & *dR
        END DO
      END DO

      RETURN
      END

C ************************************************************** AdjustForKnown

      SUBROUTINE AdjustForKnown(CNodals,Val)

C This subroutine sets the appropriate residual and jacobian
C elements to zero or one for known coefficients (i.e. b.c.'s like
C T=1 at X=3...)
C This subroutine is usually called by AJUSTFORBC

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFRes.INC'
      INCLUDE 'ECFCal.INC'

      INTEGER CNodals(*)

      INTEGER I,BandCol,LocRow
      REAL*8 Val

      DO LocRow=1,BasisDeg+1
        C(CNodals(LocRow))=Val
        R(CNodals(LocRow))=0.0
        DO I=1,NoOfCoef
          CALL ComputeBandCol(CNodals(LocRow),I,BandCol)
          IF ((BandCol.GT.0).AND.(BandCol.LE.Nuc+Nlc+1)) THEN
            RJacobian(BandCol,CNodals(LocRow))=0.0
          END IF
        END DO
        CALL ComputeBandCol(CNodals(LocRow),CNodals(LocRow),
     & BandCol)
        RJacobian(BandCol,CNodals(LocRow))=1.0
      END DO

      RETURN
      END

C ************************************************************** AdjustForNatural
```

```
      SUBROUTINE AdjustForNatural(Nodals,Phi1D,BCPar,WtsJx,dR)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFCal.INC'

C Variables

C Phi1D        1D Phis (At Surface)
C WtsJx        Wts(Jx)

      INTEGER Nodals(*)
      REAL*8 Phi1D(*),WtsJx,dR,BCPar

      INTEGER LocRow

C The following statements and DO-Loops adjust the residual
C for the natural boundary conditions.

      DO LocRow=1,BasisDeg+1
        R(Nodals(LocRow))=R(Nodals(LocRow))-WtsJx*
     & BCPar*Phi1D(LocRow)*dR
      END DO

      RETURN
      END

C ************************************************************************ AssembleBC

      SUBROUTINE AssembleBC

C This subroutine adjusts the Jacobian and Residual vector for the essential
C boundary conditions. The variables passed are described in the main
C program.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFbc.INC'

      INTEGER I,J,Jp1,Equ

C The following IF-THEN blocks check to see that the basis degree is in
C the range that is used by the program.

      IF ((BasisDeg.NE.1).AND.(BasisDeg.NE.2)) THEN
        WRITE(*,*) 'Warning: BasisDeg is not 1 or 2.'
        WRITE(*,*) '        BasisDeg=',BasisDeg
        STOP 'In routine: AssembleBC'
      END IF

      DO Equ=1,NoOfEqu
        DO J=1,NoOfCorners(Equ)

          CALL IncrementI(J,Jp1,1,NoOfCorners(Equ))

          IF ((BCs(Equ,J).EQ.'P').OR.(BCs(Equ,J).EQ.'E')) THEN

C For the primary current distribution, the value of the potential at the
```

```
C upper and lower plates is known.  These are essential b.c.'s
C BC type P or E

              IF (Corners(Equ,J).LE.Corners(Equ,Jp1)) THEN
                CALL SubKnown(Corners(Equ,J),Corners(Equ,Jp1)-1,
     & Equ,J)
              ELSE
                CALL SubKnown(Corners(Equ,J),NoOfBDNodes,Equ,J)
                CALL SubKnown(1,Corners(Equ,Jp1)-1,Equ,J)
              END IF

            ELSE IF (BCs(Equ,J).EQ.'S') THEN

C For secondary current distribution

              IF (Corners(Equ,J).LE.Corners(Equ,Jp1)) THEN
                CALL SubSecondary(Corners(Equ,J),Corners(Equ,Jp1)-1,
     & Equ,J)
              ELSE
                CALL SubSecondary(Corners(Equ,J),NoOfBDNodes,Equ,J)
                CALL SubSecondary(1,Corners(Equ,Jp1)-1,Equ,J)
              END IF

            ELSE IF (BCs(Equ,J).EQ.'N') THEN

C For natural boundary condition (slope set)
C (No need to call if condition is insulator, because, slope =0, adds
C nothing to residual or jacobian)

              IF (Corners(Equ,J).LE.Corners(Equ,Jp1)) THEN
                CALL SubNatural(Corners(Equ,J),Corners(Equ,Jp1)-1,
     & Equ,J)
              ELSE
                CALL SubNatural(Corners(Equ,J),NoOfBDNodes,Equ,J)
                CALL SubNatural(1,Corners(Equ,Jp1)-1,Equ,J)
              END IF

            END IF

          END DO
        END DO

        RETURN
        END

C ************************************************************** AssembleMatrices

        SUBROUTINE AssembleMatrices

C Internal Variables

C NoOfGaussPts        The number of points to use in the Gaussian
C                     quadrature approximatioin of the integration.
C Phi()               The value of the Local Weighting function at the current
C                     gauss point.  (Phi(1) is local weighting function 1,
C                     while Phi(2) is local weighting function 2.)
C PhiZeta()           The slope of the weighting functions at the
C                     current gauss point.
```

```
      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'
      INTEGER Elem,Equ

      IF (NoOfDim.NE.2) THEN
        WRITE(*,*) 'Warning: NoOfDim is not 2'
        WRITE(*,*) '        The code must be altered!'
        STOP 'In Routine: AssembleMatrices'
      END IF

C The following CALL zeros the residual vector and the jacobian

      CALL ZeroRJ

      DO Equ=1,NoOfEqu

C The following DO-LOOP loops over the number of elements

        DO Elem=1,NoOfElems

          IF (ElemType(Elem).EQ.'S') THEN
            CALL AssembleSquare(Equ,Elem)
          ELSE IF (ElemType(Elem).EQ.'T') THEN
            CALL AssembleTriangle(Equ,Elem)
          END IF

        END DO
      END DO

      RETURN
      END

C ***************************************************************** AssembleSquare

      SUBROUTINE AssembleSquare(Equ,Elem)

C Internal Variables

C NoOfGaussPts        The number of points to use in the Gaussian
C                     quadrature approximatioin of the integration.
C Phi()               The value of the Local Weighting function at the current
C                     gauss point. (Phi(1) is local weighting function 1,
C                     while Phi(2) is local weighting function 2.)
C PhiZeta()           The slope of the weighting functions at the
C                     current gauss point.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFRes.INC'
      INCLUDE 'ECFCal.INC'
      INCLUDE 'ECFPar.INC'

      REAL*8 Zeta(MaxNoOfDim),XZeta(MaxNoOfDim,MaxNoOfDim),DetXZeta,
     & Vlcty(MaxNoOfDim)
      REAL*8 PHI((MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiZeta(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiX(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
```

```
     & Pot(MaxNoOfEqu),PotX(MaxNoOfDim,MaxNoOfEqu),
     & dX(MaxNoOfDim),X(MaxNoOfDim)
        INTEGER LocRow,LocCol,Elem,J,Jx,Jy,BandCol,Equ,Upper

C The following DO-LOOP loops over the Gauss Points (performing the gaussian
C quadrature numerical integration). It forms the RJacobian matrix and the R
C vector in an element by element fashion.

        CALL GetUpper(ElemType(Elem),Upper)

        DO Jx=1,NoOfGaussPts
        DO Jy=1,NoOfGaussPts

          Zeta(1)=Pts(Jx)
          Zeta(2)=Pts(Jy)

          CALL Compute2DPhis(Zeta,Phi,PhiZeta)
          CALL ComputeIsoPara(Elem,PhiZeta,XZeta,DetXZeta
     & ,Upper)
          CALL ComputePhiX(Upper,PhiZeta,XZeta,DetXZeta,PhiX)
          CALL ComputePot(Elem,Upper,Phi,PhiX,Pot,PotX)

C The following call computes the velocity in the zeta(1) and
C the zeta(2) directions. To do this, it needs to know the
C current values of Zeta(1) and Zeta(2), as well as the current
C element number.

          IF (Convection) THEN
            CALL ComputeLoc(Phi,Elem,X)
            CALL Velocity(X,Vlcty)
          END IF

          DO LocRow=1,Upper

C The following statements compute the Residual vector

            R(CLoc(Equ,Elem,LocRow))=R(CLoc(Equ,Elem,LocRow))
     & +Wts(Jx)*Wts(Jy)*(PotX(1,Equ)*PhiX(1,LocRow)+
     & PotX(2,Equ)*PhiX(2,LocRow))*DetXZeta

            IF (Convection) THEN
              R(CLoc(Equ,Elem,LocRow))=R(CLoc(Equ,Elem,LocRow))
     & +Wts(Jx)*Wts(Jy)*(Peclet*Phi(LocRow)*
     & (Vlcty(1)*PotX(1,Equ)+Vlcty(2)*PotX(2,Equ)))*DetXZeta
            END IF

            DO LocCol=1,Upper

C The following statements compute the Jacobian

              CALL ComputeBandCol(CLoc(Equ,Elem,LocRow),
     & CLoc(Equ,Elem,LocCol),BandCol)

                RJacobian(BandCol,CLoc(Equ,Elem,LocRow))=
     & RJacobian(BandCol,CLoc(Equ,Elem,LocRow))-Wts(Jx)*Wts(Jy)*
     & (PhiX(1,LocRow)*PhiX(1,LocCol)+
     & PhiX(2,LocRow)*PhiX(2,LocCol))*DetXZeta
```

```
            IF (Convection) THEN
                RJacobian(BandCol,CLoc(Equ,Elem,LocRow))=
     & RJacobian(BandCol,CLoc(Equ,Elem,LocRow))-Wts(Jx)*Wts(Jy)*
     & (Peclet*Phi(LocRow)*(Vlcty(1)*PhiX(1,LocCol)+
     & Vlcty(2)*PhiX(2,LocCol)))*DetXZeta
            END IF

          END DO
        END DO
      END DO
      END DO

      RETURN
      END

C ************************************************************** AssembleTriangle

      SUBROUTINE AssembleTriangle(Equ,Elem)

C Internal Variables

C NoOfGaussPts        The number of points to use in the Gaussian
C                     quadrature approximatioin of the integration.
C Phi()               The value of the Local Weighting function at the current
C                     gauss point. (Phi(1) is local weighting function 1,
C                     while Phi(2) is local weighting function 2.)
C PhiZeta()           The slope of the weighting functions at the
C                     current gauss point.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFTriPts.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFRes.INC'
      INCLUDE 'ECFCal.INC'
      INCLUDE 'ECFPar.INC'

      REAL*8 Zeta(MaxNoOfDim),XZeta(MaxNoOfDim,MaxNoOfDim),DetXZeta,
     & Vlcty(MaxNoOfDim)
      REAL*8 PHI((MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiZeta(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiX(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & Pot(MaxNoOfEqu),PotX(MaxNoOfDim,MaxNoOfEqu),
     & dX(MaxNoOfDim),X(MaxNoOfDim)
      INTEGER LocRow,LocCol,Elem,J,Jx,Jy,BandCol,Equ,Upper

C The following DO-LOOP loops over the Gauss Points (performing the gaussian
C quadrature numerical integration). It forms the RJacobian matrix and the R
C vector in an element by element fashion.

      CALL GetUpper(ElemType(Elem),Upper)
      DO Jx=1,NoOfTriPts

        Zeta(1)=TriPts(Jx,1)
        Zeta(2)=TriPts(Jx,2)

        CALL ComputeTriPhis(Zeta,Phi,PhiZeta)
        CALL ComputeIsoPara(Elem,PhiZeta,XZeta,DetXZeta,Upper)
        CALL ComputePhiX(Upper,PhiZeta,XZeta,DetXZeta,PhiX)
```

```
      CALL ComputePot(Elem,Upper,Phi,PhiX,Pot,PotX)

C The following call computes the velocity in the zeta(1) and
C the zeta(2) directions.  To do this, it needs to know the
C current values of Zeta(1) and Zeta(2), as well as the current
C element number.

      IF (Convection) THEN
        CALL ComputeLoc(Phi,Elem,X)
        CALL Velocity(X,Vlcty)
      END IF

      DO LocRow=1,Upper

C The following statements compute the Residual vector

        R(CLoc(Equ,Elem,LocRow))=R(CLoc(Equ,Elem,LocRow))
     & +TriWts(Jx)*(PotX(1,Equ)*PhiX(1,LocRow)+
     & PotX(2,Equ)*PhiX(2,LocRow))*DetXZeta

        IF (Convection) THEN
          R(CLoc(Equ,Elem,LocRow))=R(CLoc(Equ,Elem,LocRow))
     & +TriWts(Jx)*(Peclet*Phi(LocRow)*
     & (Vlcty(1)*PotX(1,Equ)+Vlcty(2)*PotX(2,Equ)))*DetXZeta
        END IF

        DO LocCol=1,Upper

C The following statements compute the Jacobian

          CALL ComputeBandCol(CLoc(Equ,Elem,LocRow),
     & CLoc(Equ,Elem,LocCol),BandCol)

            RJacobian(BandCol,CLoc(Equ,Elem,LocRow))=
     & RJacobian(BandCol,CLoc(Equ,Elem,LocRow))-TriWts(Jx)*
     & (PhiX(1,LocRow)*PhiX(1,LocCol)+
     & PhiX(2,LocRow)*PhiX(2,LocCol))*DetXZeta

          IF (Convection) THEN
              RJacobian(BandCol,CLoc(Equ,Elem,LocRow))=
     & RJacobian(BandCol,CLoc(Equ,Elem,LocRow))-TriWts(Jx)*
     & (Peclet*Phi(LocRow)*(Vlcty(1)*PhiX(1,LocCol)+
     & Vlcty(2)*PhiX(2,LocCol)))*DetXZeta
          END IF

        END DO
      END DO
    END DO

    RETURN
    END

C ************************************************************** ComputeBandCol

    SUBROUTINE ComputeBandCol(Row,Col,BandCol)

C This function computes the column in a banded matrix given the
C row and column of a square matrix, and (from ECFHdr) the number of
```

```
C lower columns (Nlc)

      INCLUDE 'ECFHdr.INC'
      INTEGER Row,Col,BandCol

      BandCol=Col+Nlc+1-Row

      RETURN
      END

C ************************************************************ ComputeDelPotDotNorm

      SUBROUTINE ComputeDelPotDotNorm(V,Kinetics,Pot,DelPotDotNorm,
     & DelPotDotNormPot)

C This subroutine computes the normal current/conductivity (DelPotDotNorm) and
C the derivative of the normal current/conductivity (DelPotDotNormPot) with
C respect to potential for a given wall voltage (V) and electrolyte potential
C (Pot)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFPar.INC'

      REAL*8 Pot,DelPotDotNorm,DelPotDotNormPot,V,Curr,CurrPot
      CHARACTER*(*) Kinetics

      REAL*8 Eta,dEtadPot
      REAL*8 C2,dPotdC2,CurrC2,C1,C3,dC1dC2

      IF ((BINARY).OR.(Supporting)) THEN

C This block handles butler-volmer kinetic conditions for a 1:1 binary
C electrolyte where metal is depositing. The exchange current density is
C assumed to vary with the surface concentration to the first power.

        C2=Pot
        Pot= LOG(C2/BinaryK2)
        dPotdC2= 1.0/ C2

        Eta=(v-Pot)
        dEtadPot= -1.0

        IF (Binary) THEN
          C1=C2
          dC1dC2=1.0
        ELSE IF (Supporting) THEN
          C3=BinaryK2*SupportingK3/C2
          C1=C2-C3
          dC1dC2=1.0+BinaryK2*SupportingK3/(C2*C2)
        END IF

        Curr = 1.0/WagnerLin * c1 * (EXP(aA*Eta)-EXP(-aC*Eta))
        CurrC2 = 1.0/WagnerLin *
     & ((+dC1dC2+aA*dEtadPot*dPotdC2*c1)*EXP(aA*Eta) +
     &  (-dC1dC2+aC*dEtadPot*dPotdC2*c1)*EXP(-aC*Eta))

C The following two variables would be better named (for this block of code
C only) DelC2DotNorm and DelC2DotNormC2
```

```
          DelPotDotNorm = Curr
          DelPotDotNormPot= CurrC2

        ELSE

C The following definition for the overpotential (V>0, Pot>0) gives eta<0 for
C cathodic current.

          Eta=(V-Pot)
          dEtadPot=-1.0

          IF (Kinetics.EQ.'BV') THEN

            Curr = 1.0/WagnerLin * (EXP(aA*Eta)-EXP(-aC*Eta))
            CurrPot = 1.0/WagnerLin
     & *(aA*EXP(aA*Eta)+aC*EXP(-aC*Eta))*dEtadPot

            DelPotDotNorm = Curr
            DelPotDotNormPot = CurrPot

          ELSE IF (Kinetics.EQ.'LIN') THEN

            Curr = (aA+aC)*Eta/WagnerLin
            CurrPot =(aA+aC)*dEtadPot/WagnerLin

            DelPotDotNorm=Curr
            DelPotDotNormPot=CurrPot

          ELSE IF (Kinetics.EQ.'USER') THEN

            CALL UserKinetics(V,Pot,DelPotDotNorm,DelPotDotNormPot)

          ELSE

            WRITE(*,*) 'Error: Unknown Kinetics=',Kinetics
            STOP 'ECFCompute: In Routine: ComputeDelPotDotNorm'

          END IF

        END IF

        RETURN
        END

C ************************************************************** ComputeTangent

        SUBROUTINE ComputeTangent(Phi1DZeta,Nodals,dX,dY)

C The following DO Loop computes the change in pathlength for a change in
C zeta (integration variable)

        INCLUDE 'ECFHdr.INC'
        INCLUDE 'ECFMsh.INC'

        REAL*8 dX,dY,Phi1DZeta(*)
        INTEGER Nodals(*),K

        dX=0.0
```

```
      dY=0.0
      DO K=1,BasisDeg+1
        dX=dX+Nodes(Nodals(K),1)*Phi1DZeta(K)
        dY=dY+Nodes(Nodals(K),2)*Phi1DZeta(K)
      END DO

      RETURN
      END

C ********************************************************************** GETNODALS

      SUBROUTINE GetNodals(I,Equ,Nodals,CNodals)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'

      INTEGER I,Equ
      INTEGER Nodals(*),CNodals(*)

      INTEGER K,IpK

C The following DO-loop loads the nodal locations for the current element
C into Nodals(*), and coefficient locations into CNodals()

      DO K=0,BasisDeg
        CALL IncrementI(I,IpK,K,NoOfBDNodes)
        Nodals(K+1)=BDNodes(IpK)
        CALL ComputeCoefLoc(BDNodes(IpK),Equ,CNodals(K+1))
      END DO

      RETURN
      END

C ********************************************************************** NewtonRaphson

      SUBROUTINE NewtonRaphson(NoOfIterations,Converged)

C This subroutine uses the iterative Newton Raphson technique to obtain a
C solution to the differential equation.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFCal.INC'
      INCLUDE 'ECFRes.Inc'

      INTEGER NoOfIterations
      LOGICAL Converged,FirstLoop

      REAL*8 NormR,NormdC,LastNormdC,dC(MaxNoOfCoef)
      INTEGER I,J
      REAL*8 VecNormDP
      EXTERNAL VecNormDP

C The following DO-LOOP Performs the Newton-Raphson iterations.

      Converged=.FALSE.
      LastNormdC=0.0
      NormdC=SQRT(Tolerance)
      NoOfIterations=0
```

```
      FirstLoop=.TRUE.

      DO WHILE(.NOT.Converged)

C The following CALLS assemble the matrices used in the Newton-Raphson
C technique, and solve the set of simultaneous equations, and Updatec the
C coefficient vector.

        CALL AssembleMatrices

C Note: the following call adjusts the residual and Jacobian for the
C essential and natural b.c.'s

        CALL AssembleBC

        IF (Debugging) CALL PrintMat

C The following statements prepare for and solve the system of linear
C equations. The subroutine DLSARB is an IMSL subroutine for solving
C a banded system of equations.

        CALL DLSARB(NoOfCoef,RJacobian,MaxBandWidth,Nlc,Nuc,
   & R,2,dC)

C The following DO-Loop transfers the coefficient values from the DLSARB
C routine to the dC() vector.

        IF (NonNegative) THEN
          DO I=1,NoOfCoef
            IF (dC(I)+C(I).LT.0) dC(I)=-C(I)
          END DO
        END IF

C The following statements check the norm of the change in coefficients
C for quadratic convergence.

        LastNormdC=NormdC
        NormdC=VecNormDP(NoOfCoef,dC)

        IF ((.NOT.FirstLoop).AND.(NormdC.GT.1000)) THEN
          WRITE(*,*) 'Warning: Guesses might be diverging.'
          WRITE(*,*) '        Current Guess Norm=',NormdC
          WRITE(*,*) '           Last Guess Norm=',LastNormdC
        END IF
        FirstLoop=.FALSE.

        CALL UpdateC(dC)

        NormR=VecNormDP(NoOfCoef,R)
        Converged=((NormR.LT.tolerance).OR.(NormdC.LT.Tolerance))

C The following WRITE statement prints the 2-norm of dC for
C each iteration. This allows the user to check for quadratic
C convergence.

        WRITE(*,90) NormdC,NormR
90      FORMAT(X,10X,
   & 'dC=',G14.5,:,'  Residual=',G14.5)
```

```
C         ‖dC‖=gggggggggggggg  ‖Residual‖=gggggggggggggg
            NoOfIterations=NoOfIterations+1

C The following IF-THEN block prevents the subroutine from performing
C more than MaxNoOfIter iterations.

            IF (NoOfIterations.GT.MaxNoOfIter) THEN
              WRITE(*,*) 'Warning: number of newton-raphson iterations'
              WRITE(*,*) 'exceeds ',MaxNoOfIter
              STOP 'In routine: NewtonRaphson'
            END IF

          END DO

C The following write statement concludes the report for this newton iteration
C (# of Iterations, and whether or  not the last two steps
C converged quadratically)

          IF (LastNormdC**2.LT.NormdC) THEN
            WRITE(*,100) NoOfIterations
100         FORMAT(X,' **** ',14X,'   # of Iterations:',I3,
     & '    UNQUADRATIC CONVERGENCE')
          ELSE
            WRITE(*,101) NoOfIterations
101         FORMAT(X,'      ',14X,'   # of Iterations:',I3,
     & '    >= Quadratic Convergence')
          END IF

          IF ((NormdC.GT.0).AND.(LastNormdC.GT.0).AND.(LastNormdC.LT.1))
     & WRITE (*,110) LOG(NormdC)/LOG(LastNormdC)
110       FORMAT(//,X,' Order of Convergence=',G14.7,//)

          IF (NormR.GT.Tolerance)
     & WRITE(*,*) 'Warning: ‖Residual‖ is greater than ',Tolerance

          RETURN
          END

C ********************************************************************** PrintCLoc

          SUBROUTINE PrintCLoc

C This subroutine prints the CLoc matrix for user verification.

          INCLUDE 'ECFHdr.INC'
          INCLUDE 'ECFRes.INC'

          INTEGER I,J,K

          WRITE(*,99)
99        FORMAT(X,'Matrix CLoc',/,
     & X,'Eqn Elem  Local Coefficients',/,
     & X,'            1     2     3     4',/)

          DO I=1,NoOfElems
            DO J=1,NoOfEqu
              WRITE(*,100) J,I,(CLoc(J,I,K),K=1,(BasisDeg+1)**NoOfDim)
100           FORMAT(X,I2,3X,I2,3X,<(BasisDeg+1)**NoOfDim>(I4,XX))
```

```
        END DO
      END DO

C Eqn Elem  Local Nodes
C iixxxiixxx...

      RETURN
      END

C ******************************************************************** PrintMat

      SUBROUTINE PrintMat

C The following subroutine prints the jacobian and residual vectors.  This
C subroutine is useful for debugging the program.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFCal.INC'

      INTEGER I,J

      WRITE(*,*) ' '
      WRITE(*,*) ' Jacobian '
      WRITE(*,*) ' '

      DO I=1,NoOfCoef
        WRITE(*,*) '--- ROW (',I,')'
        WRITE(*,10) (RJacobian(J,I),J=1,Nlc+1+Nuc)
      END DO
      WRITE(*,*) '--- END '
10    FORMAT(X,8(G10.3))

      WRITE(*,*) ' '
      WRITE(*,*) ' Residual Vector'
      WRITE(*,*) ' '

      WRITE(*,10) (R(I),I=1,NoOfCoef)

      RETURN
      END

C ******************************************************************** PrintNodes

      SUBROUTINE PrintNodes

C The following subroutine prints the node locations.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'

      INTEGER I,K

      WRITE(*,*) ' '
      WRITE(*,*) 'Node Locations'
      WRITE(*,*) ' '

      DO I=1,NoOfNodes
        WRITE(*,100) I,(Nodes(I,K),K=1,NoOfDim)
```

```
      END DO
100   FORMAT(X,'Node(',I3,')=(',F10.5,<NoOfDim-1>(',',F10.5),')')

      RETURN
      END

C ******************************************************************* PrintResults

      SUBROUTINE PrintResults(TotalIterations,NoOfTimeSteps)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFRes.INC'
      INTEGER I,TotalIterations,NoOfTimeSteps

      WRITE(*,*) '-----------------------------------------------'
      WRITE(*,10) TotalIterations,NoOfTimeSteps
10    FORMAT(X,'Final Coefficients   (',I6,' iterations)',/,
  &       X,'                    (',I6,' Total Iterations)')
      WRITE(*,*) ' '

      WRITE(*,11) BasisDeg,NoOfElems
11    FORMAT(X,'                    ',10X,
  & '  Degree of Basis Functions=',I1,'    No Of Elems=',I3)

      WRITE(*,13) NoOfNodes
13    FORMAT(X,'                    ',10X,
  & '                          No Of Nodes=',I3,/)

      DO I=1,NoOfCoef
        WRITE(*,100) I,C(I)
      END DO
100   FORMAT(X,'C(',I3,')=',F10.5)

      RETURN
      END

C ********************************************************************** SubKnown

      SUBROUTINE SubKnown(Start,End,Equ,J)

C This subroutine loops over known coefficients, and calls a subroutine
C that sets them to zero.

      INTEGER Start,End,Equ,J

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFbc.INC'

      INTEGER Nodals(MaxNoOfBasisDeg+1),I
      INTEGER CNodals(MaxNoOfBasisDeg+1)

      DO I=Start,End,BasisDeg
        CALL GetNodals(I,Equ,Nodals,CNodals)
        CALL AdjustForKnown(CNodals,BCPar(Equ,J))
      END DO

      RETURN
      END
```

```
C ******************************************************************** SubNatural

      SUBROUTINE SubNatural(Start,End,Equ,Side)

C This subroutine loops over boundary nodes that have Natural boundary
C conditions.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFCal.INC'
      INCLUDE 'ECFbc.INC'

      INTEGER Start,End,Equ,Side

      REAL*8 dX,dY,dR,
   & Phi1D(MaxNoOfBasisDeg+1),Phi1DZeta(MaxNoOfBasisDeg+1)
      INTEGER Jx,Nodals(MaxNoOfBasisDeg+1),IpK,I,K
      INTEGER CNodals(MaxNoOfBasisDeg+1)

C I     Boundary Node index
C K     Local Node Index

      DO I=Start,End,BasisDeg

        CALL GetNodals(I,Equ,Nodals,CNodals)

C The following DO-loop loops over the integration points.

        DO Jx=1,NoOfGaussPts

C The following statements compute the slope of the path with respect to
C the integration parameter at the Gauss Point.

          CALL Compute1DPhis(BasisDeg,Pts(Jx),Phi1D,Phi1DZeta)
          CALL ComputeTangent(Phi1DZeta,Nodals,dX,dY)
          dR=SQRT(dX*dX+dY*dY)

          CALL AdjustForNatural(CNodals,Phi1D,
   & BCPar(Equ,Side),Wts(Jx),dR)

        END DO
      END DO

      RETURN
      END

C ******************************************************************** SubSecondary

      SUBROUTINE SubSecondary(Start,End,Equ,Side)

C This subroutine loops over boundary nodes that have secondary boundary
C conditions.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFRes.INC'
```

```
      INCLUDE 'ECFCal.INC'
      INCLUDE 'ECFbc.INC'

      INTEGER Start,End,Equ,Side

      REAL*8 dX,dY,dR,Phi1D(MaxNoOfBasisDeg+1),
     & Phi1DZeta(MaxNoOfBasisDeg+1)
      REAL*8 Pot,DelPotDotNorm,DelPotDotNormPot
      INTEGER Jx,Nodals(MaxNoOfBasisDeg+1),IpK,I,K
      INTEGER CNodals(MaxNoOfBasisDeg+1)

C The following DO-loop goes along the boundary.

      DO I=Start,End,BasisDeg

        CALL GetNodals(I,Equ,Nodals,CNodals)

C The following DO-loop loops over the integration points.

        DO Jx=1,NoOfGaussPts

C The following statements compute the slope of the path length vs. the
C integration variable at the intregation point.

          CALL Compute1DPhis(BasisDeg,Pts(Jx),Phi1D,Phi1DZeta)
          CALL ComputeTangent(Phi1DZeta,Nodals,dX,dY)
          dR=SQRT(dX*dX+dY*dY)

C The following DO-loop compute the potential at the boundary location.

          Pot=0.0
          DO K=1,(BasisDeg+1)
            Pot=Pot+C(CNodals(K))*Phi1D(K)
          END DO

C The following CALL computes (Del Potential).(Unit Normal) and
C d[(Del Potential).(Unit Normal)]/d[Potential]

          CALL ComputeDelPotDotNorm(BCPar(Equ,Side),Kinetics(Equ),
     & Pot,DelPotDotNorm,DelPotDotNormPot)

          CALL AdjustForKinetics(CNodals,
     & Phi1D,DelPotDotNorm,DelPotDotNormPot,Wts(Jx),dR)

        END DO
      END DO

      RETURN
      END

C ******************************************************************** TriPtsWts

      SUBROUTINE TriPtsWts
      INCLUDE 'ECFTriPts.INC'

C See Burnett p 596 for other than NoOfTriPts=4
C (Note: weights are divided by 2 as indicated in formula given on p 596
C Gaussian integration points for a triangular element.
```

```
      NoOfTriPts=4

      TriPts(1,1)=1.0/3.0
      TriPts(1,2)=1.0/3.0
      TriWts(1)=-27.0/48.0/2.0

      TriPts(2,1)=1.0/5.0
      TriPts(2,2)=1.0/5.0
      TriWts(2)=25.0/48.0/2.0

      TriPts(3,1)=3.0/5.0
      TriPts(3,2)=1.0/5.0
      TriWts(3)=25.0/48.0/2.0

      TriPts(4,1)=1.0/5.0
      TriPts(4,2)=3.0/5.0
      TriWts(4)=25.0/48.0/2.0

      RETURN
      END

C ********************************************************************** UpdateC

      SUBROUTINE UpdateC(dC)

C This subroutine Updates the coefficients C() by adding the correction
C dC() to them.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFRes.INC'

      REAL*8 dC(*),Norm
      INTEGER I,K

      DO I=1,NoOfCoef
        C(I)=C(I)+dC(I)
        IF (Debugging) WRITE(*,*) 'C(',I,')=',C(I)
      END DO

      RETURN
      END

C ********************************************************************** ZeroRJ

      SUBROUTINE ZeroRJ

C This subroutine zeros the residual vector, the jacobian, and the continuance
C vector

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFCal.INC'

      INTEGER I,J

      DO I=1,NoOfCoef
        R(I)=0.0
        DO J=1,Nuc+Nlc+1
          RJacobian(J,I)=0.0
```

```
      END DO
    END DO

    RETURN
    END

c *
```

```
      PROGRAM ECFPlot

C This program reads a results file and makes a plot file

C Written by: Ken Jordan                8/29/87

C ******************************************************************************
C Variables
C ******************************************************************************

C FileName      The name of the names file. (Contains file names
C               for reading remaining data.)

C ******************************************************************************
C Files
C ******************************************************************************

C *.BC  Contains the boundary conditions
C *.HDR         Contains problem header information. (i.e. No of coefficients,
C       Number of dimensions, Number of equations...)
C *.PAR         Contains parameter information. E.g. Kinetic parameters such
C       as the exchange current density, the transfer coefficients...
C *.MSH         Contains the mesh and node locations and boundary node locations
C *.RESContains the Coefficients and CLoc(,,)
C *.NAM         Contains the names of the files to use in solving the problem
C       E.g. *.NAM defines the filenames for *.bc, *.hdr, *.par, *.msh,
C       *.msh
C *.iso  Contains the data needed for isopotential plots.
C *.SLP contains slope along boundary.

C ******************************************************************************
C Main Program Follows
C ******************************************************************************

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFNam.INC'
      CHARACTER*80 FileName,ValFile

C The following call reads the values of variables that
C define the problem and solution technique (BasisDeg, NoOfElems,
C NoOfNodes,NoOfXElems,NoOfYElems,Tolerance)

      CALL GetFileName('Names file?',FileName)
      CALL ReadNames(FileName)

      CALL ReadFEMUser

      CALL FindIso(PlotFile)
      CALL PlotBdrySlope(SlopeFile)
      ValFile='Bdry.Val'
      CALL PlotBdryVal(ValFile)

      STOP 'Normal End'
      END

C ************************************************************** ComputeLocStep

      SUBROUTINE ComputeLocStep(BDNodeLoc,dX,XBase)
```

```
C The following statements compute the step to take and in which
C direction on the local element.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'

      INTEGER BDNodeLoc
      REAL*8 dX(*),XBase(*)

C For Square Elements, there are four cases:
C 4-3   B,BP1= 1,2     dX(1)=1.0/NoOfGaussPts dX(2)=0
C ||           2,3     dX(1)=0                dX(2)=1.0/NoOfGaussPts
C 1-2          3,4     dX(1)=-1.0/NoOfGaussPtsdX(2)=0
C              4,1     dX(1)=0                dX(2)=-1.0/NoOfGaussPts

      dX(1)=0.0
      dX(2)=0.0

      IF (BDNodeLoc.EQ.1) THEN

        XBase(1)=0.0
        XBase(2)=0.0
        dX(1)=1.0/NoOfGaussPts

      ELSE IF (BDNodeLoc.EQ.2) THEN

        XBase(1)=1.0
        XBase(2)=0.0
        dX(2)=1.0/NoOfGaussPts

      ELSE IF (BDNodeLoc.EQ.3) THEN

        XBase(1)=1.0
        XBase(2)=1.0
        dX(1)=-1.0/NoOfGaussPts

      ELSE IF (BDNodeLoc.EQ.4) THEN

        XBase(1)=0.0
        XBase(2)=1.0
        dX(2)=-1.0/NoOfGaussPTs

      ELSE

        WRITE(*,*) 'Error: Location of Boundary node in Local Element'
        WRITE(*,*) '       is unknown.'
        WRITE(*,*) '       Local Element Location=',BDNodeLoc
        WRITE(*,*) '       Perhaps basis degree is larger than one?'
        STOP 'In Routine: ComputeLocStep'

      END IF

      RETURN
      END

C ************************************************************** ComputeNormal

      SUBROUTINE ComputeNormal(BdNode,BDNodeP1,UnitNormal)
```

```
C This subroutine computes the outward pointing unit normal given the boundary
C node number. The normal is perpendicular to the line segment from BdNode
C to BdNode+1.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'

      REAL*8 UnitNormal(*),Distance,dX(MaxNoOfDim),Sum
      INTEGER BdNode,BdNodeP1,I

      Sum=0.0
      DO I=1,NoOfDim
        dX(I) = Nodes(BDNodes(BDNodeP1),I)-Nodes(BDNodes(BDNode),I)
        Sum=Sum+dX(I)*dX(I)
      END DO
      Distance=SQRT(Sum)

C UnitDistance(1)= dX(1)/Distance
C UnitDistance(2)= dX(2)/Distance

      UnitNormal(1) = dX(2)/Distance
      UnitNormal(2) = -dX(1)/Distance

      RETURN
      END

C ************************************************************************ FindIso

      SUBROUTINE FindIso(FileName)

C This subroutine compute the location of isopotential points and stores
C them in file FileName.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFRes.INC'
      INCLUDE 'ECFPar.INC'

      PARAMETER MaxNoOfPot=25
      CHARACTER*(*) FileName
      REAL*8 PotMax,PotMin,dPot,Pots(MAxNoOfPot+1),Zero/0.0/,One/1.0/,
     & XPts(MaxNoOfPot+1,2*MaxNoOfNodes),
     & YPts(MaxNoOfPot+1,2*MaxNoOfNodes),
     & XTemp(2*MaxNoOfNodes),YTemp(2*MaxNoOfNodes)
      REAL*8 C3Min,C3Max
      INTEGER NoOfPPts(MaxNoOfPot+1),I,J,K,L,RNode,LNode,
     & Pointer(2*MaxNoOfNodes),Kp1,NoOfPot,Upper,FilLen

C The following statements find the max and min potentials.

      PotMax=C(1)
      PotMin=C(1)

      DO I=1,NoOfCoef
        PotMax=MAX(PotMax,C(I))
        PotMin=MIN(PotMin,C(I))
      END DO
```

```
      WRITE(*,*) ' '
      WRITE(*,*) ' Iso Potential Plot Information: File ',FileName
      WRITE(*,*) ' '
      IF (.NOT.((Binary).OR.(Supporting))) THEN
        WRITE(*,*) 'Maximum Potential Found:',PotMax
        WRITE(*,*) 'Minimum Potential Found:',PotMin
      ELSE
        WRITE(*,*) 'Maximum Potential Found:',LOG(PotMax/BinaryK2)
        WRITE(*,*) 'Minimum Potential Found:',LOG(PotMin/BinaryK2)
        WRITE(*,*) ' '
        IF (.NOT.Supporting) THEN
          WRITE(*,*) 'Maximum Concentration Found:',PotMax
          WRITE(*,*) 'Minimum Concentration Found:', PotMin
        ELSE

          C3Min=BinaryK2*SupportingK3/PotMax
          C3Max=BinaryK2*SupportingK3/PotMin

          WRITE(*,*) 'Maximum Concentration 3 Found:',C3Max
          WRITE(*,*) 'Minimum Concentration 3 Found:',C3Min
          WRITE(*,*) ' '
          WRITE(*,*) 'Maximum Concentration 2 Found:',PotMax
          WRITE(*,*) 'Minimum Concentration 2 Found:', PotMin
        END IF
      END IF

      WRITE(*,*) ' '

      CALL IInp('No Of Iso- curves?',NoOfPot)
      CALL EditDP('Maximum ',PotMax)
      CALL EditDP('Minimum ',PotMin)

C The following statements find the (NoOfPot+1) isopotentials that will be
C plotted.

      dPot=(PotMax-PotMin)/Float(NoOfPot)

      DO L=1,NoOfPot+1
        Pots(L)=PotMin+(L-1)*dPot
      END DO

      CALL PlotPots(NoOfPot,Pots,FileName)

      IF ((Binary).OR.(Supporting)) THEN

C The following statements find the (NoOfPot+1) isopotentials that will be
C plotted.

        dPot=(LOG(PotMax/BinaryK2)-LOG(PotMin/BinaryK2))
     & /Float(NoOfPot)

        DO L=1,NoOfPot+1
          Pots(L)=LOG(PotMin/BinaryK2)+(L-1)*dPot
          Pots(L)=BinaryK2*EXP(Pots(L))
        END DO

        CALL StringLength(FileName,FilLen)
        CALL PlotPots(NoOfPot,Pots,FileName(1:FilLen)//'Pot')
```

```
      END IF

      IF (Supporting) THEN
        dPot=(C3Max-C3Min)/Float(NoOfPot)

        DO L=1,NoOfPot+1
          Pots(L)=C3Min+(L-1)*dPot
          Pots(L)=BinaryK2*SupportingK3/(Pots(L))
        END DO

        CALL StringLength(FileName,FilLen)
        CALL PlotPots(NoOfPot,Pots,FileName(1:FilLen)//'C3')

      END IF

      RETURN
      END

C ************************************************************** LocateGlobalElem

      SUBROUTINE LocateGlobalElem(BDNode,BDNodeP1,Elem,BDNodeLoc)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'

      INTEGER BDNode,BDNodeP1,Elem,BDNodeLoc,LocNode
      LOGICAL BDNodeFound,BDNodeP1Found

C This subroutine locates the global element containing the current boundary
C node and the next boundary node.

C Loop over elements
C    Check to see if BDNode Found
C    Check to see if BDNodeP1 Found.

      BDNodeFound=.FALSE.
      BDNodeP1Found=.FALSE.

      Elem=0

      DO WHILE(.NOT.(BDNodeFound.AND.BDNodeP1Found))

        Elem=Elem+1
        IF (Elem.GT.NoOfElems) THEN
          WRITE(*,*) 'Error: Boundary nodes not found in any element',
     & ' (Searched ',NoOfElems,' elements)'
          WRITE(*,*) '       Boundary Node Numbers:',BDNode
          WRITE(*,*) '       ',BDNodes(BDNode),BDNodes(BDNodeP1)
          STOP 'In Routine: LocateGlobalElem'
        END IF

        BDNodeFound=.FALSE.
        BDNodeP1Found=.FALSE.

C Search all the outside nodes (  Squares: 1,2,3,4,5,6,7,8)
C                              (Triangles: 1,2,3,4,5,6)

        DO LocNode=1,8
```

```
            IF (NLoc(Elem,LocNode).EQ.BDNodes(BdNode)) THEN
              BDNodeFound=.TRUE.
              BDNodeLoc=LocNode
            END IF
            IF (NLoc(Elem,LocNode).EQ.BDNodes(BdNodeP1)) BDNodeP1Found=.TRUE.
          END DO

        END DO

C We return when Elem has BDNode and BDNodeP1 on it!

        RETURN
        END

C ********************************************************************** Plot1Slope

        SUBROUTINE Plot1Slope(J,Upper,Elem)

        INCLUDE 'ECFHdr.INC'
        INCLUDE 'ECFMsh.INC'
        INCLUDE 'ECFPlot.INC'

        INTEGER J,Upper,Elem

        INTEGER K
        REAL*8 Cx,Xs(MaxNoOfDim)
        REAL*8 Zeta(MaxNoOfDim),XZeta(MaxNoOfDim,MaxNoOfDim),DetXZeta
        REAL*8 PHI((MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiZeta(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiX(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & Pot(MaxNoOfEqu),PotX(MaxNoOfDim,MaxNoOfEqu)

        DO K=1,NoOfDim
          Zeta(K)=XBase(K)+J*dX(K)
        END DO

        IF (ElemType(Elem).EQ.'S') THEN
          CALL Compute2DPhis(Zeta,Phi,PhiZeta)
        ELSE
          CALL ComputeTriPhis(Zeta,Phi,PhiZeta)
        END IF

        CALL ComputeIsoPara(Elem,PhiZeta,XZeta,DetXZeta,Upper)
        CALL ComputePhiX(Upper,PhiZeta,XZeta,DetXZeta,PhiX)
        CALL ComputePot(Elem,Upper,Phi,PhiX,Pot,PotX)

C The following DO-Loop computes the dot product between PotX and
C Normal.

        Cx=0
        DO K=1,NoOfDim
          Cx=Cx+Normal(K)*PotX(K,1)
        END DO

C The next few lines compute the change in distance between the last
C computation and this one.

        XLast=X
```

```
      YLast=Y

      CALL ComputeLoc(Phi,Elem,Xs)

      X=Xs(1)
      Y=Xs(2)

      Distance=Distance+SQRT((X-XLast)**2+(Y-YLast)**2)

C The next line writes the distance and the slope computed into a file.

      WRITE(1,*) Distance,Cx

      RETURN
      END

C ************************************************************** PlotBdrySlope

      SUBROUTINE PlotBdrySlope(FileName)

C This subroutine computes the slope of the potential at the boundary.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMsh.INC'
      INCLUDE 'ECFPlot.INC'

      CHARACTER*(*) FileName
      INTEGER I,J,K,BDNode,BDNodeP1,Elem,LocNode,BDNodeLoc
      INTEGER Upper

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')
      WRITE(1,*) NoOfBDNodes*(NoOfGaussPts+1)

      Distance=0
      X=Nodes(BDNodes(1),1)
      Y=Nodes(BDNodes(1),2)

C The following DO-Loop loops over the boundary nodes.

      DO BdNode=1,NoOfBDNodes

        CALL IncrementI(BDNode,BDNodep1,1,NoOfBDNodes)

        CALL ComputeNormal(BdNode,BdNodeP1,Normal)
        CALL LocateGlobalElem(BDNode,BDNodeP1,Elem,BDNodeLoc)
        CALL ComputeLocStep(BDNodeLoc,dX,XBase)
        CALL GetUpper(ElemType(Elem),Upper)

C The following DO-Loop Loops over the gauss points and computes the value
C of the potential and the potential slope at at each point on the boundary.

        IF (ElemType(Elem).EQ.'T') THEN
          CALL Plot1Slope(NoOfGaussPts/2,Upper,Elem)
        ELSE
          DO J=0,NoOfGaussPts
            CALL Plot1Slope(J,Upper,Elem)
          END DO
```

```
        END IF
      END DO

      CLOSE(1)
      CLOSE(2)
      CLOSE(3)

      RETURN
      END

C ******************************************************************** PlotBdryVal

      SUBROUTINE PlotBdryVal(FileName)

C This subroutine computes the value of the potential at the boundary.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFMsh.INC'

      CHARACTER*(*) FileName
      INTEGER I,J,K,BDNode,BDNodeP1,Elem,LocNode,BDNodeLoc
      INTEGER Upper
      REAL*8 Distance,X,Y,XLast,YLast,Xs(MaxNoOfDim)
      REAL*8 Normal(MaxNoOfDim),dX(MaxNoOfDim),XBase(MaxNoOfDim),
     & Zeta(MaxNoOfDim),XZeta(MaxNoOfDim,MaxNoOfDim),DetXZeta
      REAL*8 PHI((MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiZeta(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiX(MaxNoOfDim,(MAxNoOfBasisDeg+1)**MaxNoOfDim),
     & Pot(MaxNoOfEqu),PotX(MaxNoOfDim,MaxNoOfEqu)

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')
      WRITE(1,*) NoOfBDNodes*(NoOfGaussPts+1)

      Distance=0
      X=Nodes(BDNodes(1),1)
      Y=Nodes(BDNodes(1),2)

C The following DO-Loop loops over the boundary nodes.

      DO BdNode=1,NoOfBDNodes

        BDNodeP1=BdNode+1
        IF (BDNodeP1.GT.NoOfBDNodes) BDNodeP1=1

        CALL LocateGlobalElem(BDNode,BDNodeP1,Elem,BDNodeLoc)
        CALL ComputeLocStep(BDNodeLoc,dX,XBase)
        CALL GetUpper(ElemType(Elem),Upper)

C The following DO-Loop Loops over the gauss points and computes the value
C of the potential and the potential slope at at each point on the boundary.

        DO J=0,NoOfGaussPts

          DO K=1,NoOfDim
            Zeta(K)=XBase(K)+J*dX(K)
          END DO
```

```
            IF (ElemType(Elem).EQ.'S') THEN
              CALL Compute2DPhis(Zeta,Phi,PhiZeta)
            ELSE
              CALL ComputeTriPhis(Zeta,Phi,PhiZeta)
            END IF

            CALL ComputeIsoPara(Elem,PhiZeta,XZeta,DetXZeta,Upper)
            CALL ComputePhiX(Upper,PhiZeta,XZeta,DetXZeta,PhiX)
            CALL ComputePot(Elem,Upper,Phi,PhiX,Pot,PotX)

C The next few lines compute the change in distance between the last
C computation and this one.

            XLast=X
            YLast=Y

            CALL ComputeLoc(Phi,Elem,Xs)

            X=Xs(1)
            Y=Xs(2)

            Distance=Distance+SQRT((X-XLast)**2+(Y-YLast)**2)

C The next line writes the distance and the slope computed into a file.

            WRITE(1,*) Distance,Pot(1)

          END DO
        END DO

        CLOSE(1)
        CLOSE(2)
        CLOSE(3)

        RETURN
        END

C ********************************************************************** PlotPots

        SUBROUTINE PlotPots(NoOfPot,Pots,FileName)

        INCLUDE 'ECFHdr.INC'
        INCLUDE 'ECFMsh.INC'
        INCLUDE 'ECFRes.INC'

        PARAMETER MaxNoOfPot=25

        INTEGER NoOfPts
        REAL*8 Pots(*),PotMax,PotMin
        CHARACTER*(*) FileName

        REAL*8 Zero/0.0/,One/1.0/,
     & XPts(MaxNoOfPot+1,2*MaxNoOfNodes),
     & YPts(MaxNoOfPot+1,2*MaxNoOfNodes),
     & XTemp(2*MaxNoOfNodes),YTemp(2*MaxNoOfNodes)
        INTEGER NoOfPPts(MaxNoOfPot+1),I,J,K,L,RNode,LNode,
     & Pointer(2*MaxNoOfNodes),Kp1,NoOfPot,Upper
```

```
C Open the file that will contain endpoints of segments to be plotted.

      OPEN(UNIT=1,FORM='FORMATTED',FILE=FileName,STATUS='NEW')

C The following statements Loop through elements.

      DO J=1,NoOfElems

        CALL GetUpper(ElemType(J),Upper)

C The following DO-Loop clears the number of pts found at a given potential
C for the element.

        DO L=1,NoOfPot+1
          NoOfPPts(L)=0
        END DO

C The following DO-Loop loops through the local segments.

        DO K=1,Upper

          CALL IncrementI(K,Kp1,1,Upper)

          PotMax=MAX(C(CLoc(1,J,K)),C(CLoc(1,J,Kp1)))
          PotMin=MIN(C(CLoc(1,J,K)),C(CLoc(1,J,Kp1)))

C Next, we loop over all the isopotential values.

          DO L=1,NoOfPot+1

C The following IF-THEN block locates all the points on the boundary
C of the element that have potential values we are interested in.

            IF ((Pots(L).GE.PotMin).AND.(Pots(L).LE.PotMax)) THEN
              NoOfPPts(L)=NoOfPPts(L)+1
                CALL Interp1D(C(CLoc(1,J,K)),Nodes(NLoc(J,K),1),
     & C(CLoc(1,J,Kp1)),Nodes(NLoc(J,Kp1),1),Pots(L),
     & XPts(L,NoOfPPts(L)))
                CALL Interp1D(C(CLoc(1,J,K)),Nodes(NLoc(J,K),2),
     & C(CLoc(1,J,Kp1)),Nodes(NLoc(J,Kp1),2),Pots(L),
     & YPts(L,NoOfPPts(L)))
            END IF

          END DO
        END DO

C The following statements store the points found for this element into
C the plot file.

        DO L=1,NoOfPot+1
          IF (NoOfPPts(L).GT.1) THEN
            DO I=1,NoOfPPts(L)-1
              WRITE(1,*) XPts(L,I),YPts(L,I),Pots(L)
              WRITE(1,*) XPts(L,I+1),YPts(L,I+1),Pots(L)
            END DO
          END IF
        END DO
```

```
      END DO

      CLOSE(1)

      RETURN
      END

C ************************************************************** Subroutine Calls

C Internal Subroutines (ECFPLOT.FOR)

c     SUBROUTINE ComputeLocStep(BDNodeLoc,dX,XBase)
c     SUBROUTINE ComputeNormal(BdNode,BDNodeP1,UnitNormal)
c     SUBROUTINE FindIso(FileName)
c     SUBROUTINE LocateGlobalElem(BDNode,BDNodeP1,Elem,BDNodeLoc)
c     SUBROUTINE PlotBdrySlope(FileName)
c     SUBROUTINE PlotBdryVal(FileName)

C Calls to Internal Subroutines

c     CALL ComputeLocStep(BDNodeLoc,dX,XBase)
c     CALL ComputeLocStep(BDNodeLoc,dX,XBase)
c     CALL ComputeNormal(BdNode,BdNodeP1,Normal)
c     CALL FindIso(PlotFile)
c     CALL LocateGlobalElem(BDNode,BDNodeP1,Elem,BDNodeLoc)
c     CALL LocateGlobalElem(BDNode,BDNodeP1,Elem,BDNodeLoc)
c     CALL PlotBdrySlope(SlopeFile)
c     CALL PlotBdryVal(ValFile)

C Calls to External Subroutines (ECFSUBS.FOR)-------------------F.E.M. routines

c     CALL Compute2DPhis(Zeta,Phi,PhiZeta)
c     CALL Compute2DPhis(Zeta,Phi,PhiZeta)
c     CALL ComputeIsoPara(Elem,PhiZeta,XZeta,DetXZeta)
c     CALL ComputeIsoPara(Elem,PhiZeta,XZeta,DetXZeta)
c     CALL ComputeLoc(Phi,Elem,Xs)
c     CALL ComputeLoc(Phi,Elem,Xs)
c     CALL ComputePhiX(PhiZeta,XZeta,DetXZeta,PhiX)
c     CALL ComputePhiX(PhiZeta,XZeta,DetXZeta,PhiX)
c     CALL ComputePot(Elem,Phi,PhiX,Pot,PotX)
c     CALL ComputePot(Elem,Phi,PhiX,Pot,PotX)
c     CALL ReadMesh(MeshFile)
c     CALL ReadMeshHdr(MeshHdrFile)
c     CALL ReadNames(FileName)
c     CALL ReadRes(ResFile)

C Calls To External Subroutines (SUBS.FOR)---------------------General Routines

c     CALL EditDP('Maximum Potential',PotMax)
c     CALL EditDP('Minimum Potential',PotMin)
c     CALL GetFileName('Names file?',FileName)
c     CALL IInp('No Of Isopotential curves?',NoOfPot)

C Calls To External Subroutines (MATHSUBS.FOR)---------------------Math Routines

c     CALL Interp1D(C(CLoc(1,J,K)),Nodes(NLoc(J,K),1),
c     CALL Interp1D(C(CLoc(1,J,K)),Nodes(NLoc(J,K),2),
```

```
      PROGRAM ECFIso

C This program will read an *.ISO file and plot the contours.

      INCLUDE '[KJ.DATA]PLOTLABL.INC'

      INTEGER Length,NoOfPlots,NoOfSegs

      REAL Pot1,Pot2,X1,X2,Y1,Y2

      CHARACTER TermType*3,File*80

C Get Plotter and Set It Up

      CALL GetPlotter(TermType)

C The following statements will read the data from the data file.

      NoOfPlots=0
      File='*'
      DO WHILE(File.NE.' ')
        File=' '
        CALL LIB$SPAWN('DIR [...]*.ISO')
        CALL GetFileName('Iso-potential plot file?',File)

        CALL StringLength(File,Length)
        IF ((Length.EQ.0).AND.(NoOfPlots.EQ.0)) THEN

          WRITE(*,*) 'Error: no filename entered.'

        ELSE IF (Length.GT.0) THEN

          NoOfPlots=NoOfPlots+1

          CALL GetMaxMin(File,NoOfSegs)

          Title=File
          XLabel='X'
          YLabel='Y'
          CALL GetLabels

          CALL PlotCurve(TermType,File,NoOfSegs)

        END IF
      END DO

      CALL DonePl

      WRITE(*,*) 'Normal End: ',NoOfPlots,' plots.'
      STOP
      END

C ********************************************************************** GetLabels

      SUBROUTINE GetLabels

      INCLUDE '[KJ.DATA]PLOTLABL.INC'
      INTEGER Length
```

```
      REAL T

      OPEN(UNIT=2,FILE='PLOT.LAB',STATUS='UNKNOWN',
     & FORM='FORMATTED')
      CALL StringLength(Title,Length)
      WRITE(2,*) '''',Title(1:Length),''',''Title'''
      CALL STringLength(XLAbel,Length)
      WRITE(2,*) '''',XLAbel(1:Length),''',',XMin,XMax,
     & ',''XLabel'', XMin, XMax'
      WRITE(2,*) LogX,',''LogX'''
      CALL StringLength(YLabel,Length)
      WRITE(2,*) '''',YLabel(1:Length),''',',YMin,YMax,
     & ',''YLabel'', YMin, YMax'
      WRITE(2,*) LogY,',''LogY'''
      CLOSE(2)
      CALL LIB$SPAWN('EDIT PLOT.LAB')

      OPEN(UNIT=2,FILE='PLOT.LAB',STATUS='UNKNOWN',
     & FORM='FORMATTED')
      READ(2,*) Title
      READ(2,*) XLAbel,XMin,XMax
      READ(2,*) LogX
      READ(2,*) YLabel,YMin,YMax
      READ(2,*) LogY
      CLOSE(2)

c     WRITE(*,*) 'You must enter a title.'
c     CALL EditCh('Title?',Title)

C Get X & Y Axis type and label

c     CALL GetAxis('X',XLabel,XMin,XMax,LogX)
c     CALL GetAxis('Y',YLabel,YMin,YMax,LogY)

      RETURN
      END

C ********************************************************************** GetMaxMin

      SUBROUTINE GetMaxMin(File,NoOfSegs)

      INTEGER NoOfSegs
      CHARACTER*(*) File
      REAL X1,Y1,Pot1
      REAL X2,Y2,Pot2
      INCLUDE '[KJ.DATA]PLOTLABL.INC'

      NoOfSegs=0

      OPEN(UNIT=1,FILE=File,STATUS='OLD',FORM='FORMATTED')

90    READ(1,*,END=100) X1,Y1,Pot1
      READ(1,*,END=100) X2,Y2,Pot2

      NoOfSegs=NoOfSegs+1

      IF (NoOfSegs.EQ.1) THEN
        XMin=MIN(X1,X2)
```

```
          YMin=MIN(Y1,Y2)
          XMax=MAX(X1,X2)
          YMax=MAX(Y1,Y2)
        ELSE
          XMin=MIN(XMin,X2)
          YMin=MIN(YMin,Y2)
          XMax=MAX(XMax,X2)
          YMax=MAX(YMax,Y2)

          XMin=MIN(XMin,X1)
          YMin=MIN(YMin,Y1)
          XMax=MAX(XMax,X1)
          YMax=MAX(YMax,Y1)
        END IF

        GOTO 90

100     CLOSE(1)

        RETURN
        END

C ******************************************************************** PlotCurve

        SUBROUTINE PlotCurve(TermType,File,NoOfSegs)

        CHARACTER*(*) TermType,File
        INTEGER NoOfSegs

        INCLUDE '[KJ.DATA]PLOTLABL.INC'

        INTEGER I,J,K,Length
        PARAMETER MaxNoOfPoints=2

        REAL PtsX(MaxNoOfPoints),PtsY(MaxNoOfPoints)
        CHARACTER Xl*30,Yl*30,Ti*30
        REAL Pot1,Pot2

C Set up the Graphics output using DISSPLA

        CALL SetPlotter(TermType)

        CALL Height(0.25)
        CALL StringLength(Title,Length)
        IF (Length.GT.0) CALL Headin(Title,Length,1.1,1)

        CALL StringLength(XLabel,Length)
        CALL XName(XLabel,Length)
        CALL StringLength(YLabel,Length)
        CALL YName(YLabel,Length)

        CALL YAXANG(0.0)
        CALL Graf(XMin,'SCALE',XMax,YMin,'SCALE',YMax)

        CALL NoChek

        OPEN(UNIT=1,FILE=File,STATUS='OLD',FORM='FORMATTED')
```

```
      DO I=1,NoOfSegs

        READ(1,*,END=200) PtsX(1),PtsY(1),Pot1
        READ(1,*,END=200) PtsX(2),PtsY(2),Pot2

        CALL Marker(0)
        CALL Curve(PtsX,PtsY,2,0)

      END DO

200   CLOSE(1)

      CALL EndPl(0)

      RETURN
      END
```

```
C File ECSUBS.FOR

C ******************************************************************** Compute2DPhis

      SUBROUTINE Compute2DPhis(Zeta,Phi,PhiZeta)

C This subroutine computes the values of the local weighting functions, and
C the values of their slopes.

C Input Variables

C Zeta(NoOfDim)        The value of the local variable at which to evaluate the
C                 functions.
C(In COMMON BasisDeg=degree of basis functions, 3 are hermite cubics)
C(        NoOfDim)

C Output Variables

C Phi()  Phi(1) is the first local weighting function evaluated at
C        local node.  Phi(2) is the second.
C PhiZeta(NoOfDim,) These are the derivates (with respect to zeta)
C                 evaluated at zeta(i).

      INCLUDE 'ECFHdr.INC'

      REAL*8 Phi((MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiZeta(NoOfDim,(MaxNoOfBAsisDeg+1)*MaxNoOfDim),
     & Zeta(NoOfDim)

      IF (NoOfDim.NE.2) THEN
        WRITE(*,*) 'Warning: NoOfDim is not 2'
        STOP 'In Routine: Compute2DPhis'
      END IF

      IF (BasisDeg.EQ.1) THEN

C The following local bilinear basis functions are based on the
C numbering scheme shown below:

C               4-3
C               ||
C               1-2

        Phi(1)=(1.0-Zeta(1))    *(1.0-Zeta(2))
        Phi(4)=(1.0-Zeta(1))    *(Zeta(2))
        Phi(2)=(Zeta(1))        *(1.0-Zeta(2))
        Phi(3)=(Zeta(1))        *(Zeta(2))

        PhiZeta(1,1)=-1.0       *(1.0-Zeta(2))
        PhiZeta(1,4)=-1.0       *(Zeta(2))
        PhiZeta(1,2)=1.0        *(1.0-Zeta(2))
        PhiZeta(1,3)=1.0        *(Zeta(2))

        PhiZeta(2,1)=(1.0-Zeta(1))      *(-1.0)
        PhiZeta(2,4)=(1.0-Zeta(1))      *(1.0)
        PhiZeta(2,2)=(Zeta(1)) *(-1.0)
        PhiZeta(2,3)=(Zeta(1)) *(1.0)
```

```
      ELSE IF (BasisDeg.EQ.2) THEN

C The following local biquadratic basis functions are based on the
C numbering scheme shown below:

C 4-7-3
C| |
C 8 9 6
C| |
C 1-5-2

      ELSE

        WRITE(*,*) 'Warning: Higher order basis functions not'
        WRITE(*,*) 'yet programmed for 2D'
        STOP 'In Routine: Compute2DPhis'

      END IF

      RETURN
      END

C ***************************************************************** ComputeCoefLoc

      SUBROUTINE ComputeCoefLoc(NLoca,J,CLoca)

C This subroutine is used to compute coefficient locations.

      INTEGER NLoca,J,CLoca

      INCLUDE 'ECFHdr.INC'

      CLoca=(NLoca-1)*NoOfEqu + J

      RETURN
      END

C ***************************************************************** ComputeIsoPara

      SUBROUTINE ComputeIsoPara(Elem,PhiZeta,XZeta,DetXZeta,Upper)

C This subroutine computes XZeta (dx(j)/dzeta(i)) (j=col, i=row) and DetXZeta
C (Det XZeta) where XZeta is the jacobian for the isoparametric transformation
C between the global element (represented by the x's) and the local element
C (represented by the zeta's)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'

C Note: written for BasisDeg=1

      REAL*8 XZeta(MaxNoOfDim,MaxNoOfDim),DetXZeta,
     & PhiZeta(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & IsoJacobian(MaxNoOfDim,MaxNoOfDim)

      INTEGER Elem,I,J,K,Upper

C The following DO-loop loops over the global dimensions.
```

```
      DO J=1,NoOfDim

C The following DO-Loop loops over the local dimensions.

        DO I=1,NoOfDim
          XZeta(I,J)=0.0

C The following DO-Loop computes the value of XZeta(I,J)=
C sum (x(I,K).PhiZeta(K,J)

          DO K=1,Upper
            XZeta(I,J)=XZeta(I,J)
     & +PhiZeta(I,K)*Nodes(NLoc(Elem,K),J)
          END DO

        END DO
      END DO

C The following statement computes the determinant of the transformation
C Jacobian for 2 dimensions.

      DetXZeta=XZeta(1,1)*XZeta(2,2)-XZeta(1,2)*XZeta(2,1)

      IF (DetXZeta.EQ.0.0) THEN
        WRITE(*,*) 'Error: Determinant of Jacobian is ZERO!'
        STOP 'In Routine: ComputeIsoPara'
      END IF

      RETURN
      END

C ******************************************************************** ComputeLoc

      SUBROUTINE ComputeLoc(Phi,Elem,X)

C Given the element number and the Basis Functions at the Desired point,
C This subroutine computes the location within the element of the local
C point (i.e. the point in the transformed domain)

C Phi()  The basis functions evaluated at the point in the local element
C Elem   The global element number
C X()    The global location of the local point.

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'

      REAL*8 X(MaxNoOfDim),Phi((MaxNoOfBasisDeg+1)**MaxNoOfDim)
      INTEGER Elem,I,J,Upper

      CALL GetUpper(ElemType(Elem),Upper)
      DO I=1,NoOfDim
        X(I)=0.0
        DO J=1,Upper
          X(I)=X(I)+Nodes(NLoc(Elem,J),I)*Phi(J)
        END DO
      END DO

      RETURN
```

```
      END

C ******************************************************************** ComputePhiX

      SUBROUTINE ComputePhiX(Upper,PhiZeta,XZeta,DetXZeta,PhiX)

C This subroutine computes PhiX from PhiZeta and the Jacobian of the
C isoparametric mapping.

      INCLUDE 'ECFHdr.INC'

      REAL*8 PhiZeta(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & PhiX(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim)
      REAL*8 XZeta(MaxNoOfDim,MaxNoOfDim),DetXZeta
      INTEGER I,Upper

      IF (NoOfDim.EQ.1) THEN

C Inverse is easy

        DO I=1,BasisDeg+1
          PhiX(1,I)=PhiZeta(1,I)
        END DO

      ELSE IF (NoOfDim.EQ.2) THEN

C Inverse of 2x2 is known

        IF (DetXZeta.EQ.0.0) THEN
          WRITE(*,*) 'Error: Attempting to divide by zero!'
          WRITE(*,*) '       DetXZeta=0.0!'
          STOP 'In Routine: ComputePhiX'
        END IF

        DO I=1,Upper
          PhiX(1,I)=(XZeta(2,2)*PhiZeta(1,I)-
     & XZeta(1,2)*PhiZeta(2,I))/DetXZeta
          PhiX(2,I)=(XZeta(1,1)*PhiZeta(2,I)-
     & XZeta(2,1)*PhiZeta(1,I))/DetXZeta
        END DO

      ELSE

        WRITE(*,*) 'Error: NoOfDim too great!  I can''t invert'
        WRITE(*,*) '       the isoparametric mapping!'
        WRITE(*,*) '       NoOfDim=',NoOfDim
        STOP 'In Routine: ComputePhiX'

      END IF

      RETURN
      END

C ******************************************************************** ComputePot

      SUBROUTINE ComputePot(Elem,Upper,Phi,PhiX,Pot,PotX)

C This subroutine computes the local value of the potential and the derivative
```

C of the potential with respect to the local coordinates.

```
      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFRes.INC'

      INTEGER Elem,J,K,I,Upper
      REAL*8 Pot(MaxNoOfEqu),PotX(MaxNoOfDim,MaxNoOfEqu),
     & Phi((MaxNoOfBAsisDeg+1)**MaxNoOfDim),
     & PhiX(MaxNoOfDim,(MaxNoOfBAsisDeg+1)**MaxNoOfDim),
     & dX(MaxNoOfDim)
```

C The following DO-Loops zero the Pot and PotZeta matrices

```
      DO K=1,NoOfEqu
        Pot(K)=0.0
        DO I=1,NoOfDim
          PotX(I,K)=0.0
        END DO
      END DO
```

C The following DO-Loop computes the CONC(i) and slope PotZeta(I,K)

```
      DO I=1,NoOfEqu
        DO J=1,Upper
          Pot(I)=Pot(I)+C(CLoc(I,Elem,J))*Phi(J)
          DO K=1,NoOfDim
            PotX(K,I)=PotX(K,I)
     & + C(CLoc(I,Elem,J))*PhiX(K,J)
          END DO
        END DO
      END DO

      RETURN
      END
```

C ********** ComputeTriPhis

```
      SUBROUTINE ComputeTriPhis(Zeta,Phi,PhiZeta)

      INCLUDE 'ECFHdr.INC'

      REAL*8 Zeta(MaxNoOfDim)
      REAL*8 Phi((MaxNoOfBasisDeg+1)**MaxNoOfDim)
      REAL*8 PhiZeta(MaxNoOfDim,(MaxNoOfBasisDeg+1)**MaxNoOfDim)

      IF (BasisDeg.NE.1) THEN
        WRITE(*,*) 'Error: basis functions for triangular elements'
        WRITE(*,*) '       are only known for basis degree=1'
        WRITE(*,*) '       BasisDeg=',BasisDeg
        STOP 'In Routine: ComputeTriPhis'
      END IF

      Phi(1)=1-Zeta(1)-Zeta(2)
      Phi(2)=Zeta(1)
      Phi(3)=Zeta(2)

      PhiZeta(1,1)=-1.0
      PhiZeta(2,1)=-1.0
```

```
      PhiZeta(1,2)=1.0
      PhiZeta(2,2)=0.0

      PhiZeta(1,3)=0.0
      PhiZeta(2,3)=1.0

      RETURN
      END

C ********************************************************************** Essential

      SUBROUTINE Essential(Side,Eqn,X,BCPar,C)

C This subroutine is used to set essential boundary conditions.
C Input Variables

C Side  The side number currently set to its essential boundary condition
C X()   The location of the node
C BCPar         The bc input parameter assigned to the ith side.
C Eqn   The equation number.

C Output Variables

C C     The values of the coefficients at the node for equations

      REAL*8 X(*),BCPar,C
      INTEGER Side,Eqn

      WRITE(*,*) 'Warning: external subroutine ESSENTIAL has not been',
     & 'linked into'
      WRITE(*,*) '        ECDEFINE.  Please refer to users manual.'

      RETURN
      END

C ********************************************************************** FirstGuess

      SUBROUTINE FirstGuess(Eqn,Elem,X,C)

C This subroutine is used to make a first guess for the coefficients.

C Input Variables

C Elem  The current element.
C X()   The location of the node
C Eqn   The equation number.

C Output Variables

C C     The values of the coefficients at the node for equations

      REAL*8 X(*),C
      INTEGER Elem,Eqn

      WRITE(*,*) 'Warning: external subroutine FIRSTGUESS has not been',
     & 'linked into'
      WRITE(*,*) '        ECDEFINE.  Please refer to users manual.'
```

```
      RETURN
      END

C ******************************************************************** GetFEMProb

      SUBROUTINE GetFEMProb(Prompt,FileName)

      CHARACTER*(*) FileName,Prompt
      INTEGER Length,PrLen
      LOGICAL FileExist

      CALL StringLength(Prompt,PrLen)
      CALL EditCh('Names file for '//Prompt(1:PrLen)//'?',FileName)

      IF (FileName.NE.' ') THEN

        INQUIRE(File=FileName,Exist=FileExist)
        IF (FileExist) THEN

          WRITE(*,*) 'Reading data from old files'
          CALL ReadNames(FileName)
          CALL ReadFEMUser

        ELSE

          CALL StringLength(FileName,Length)
          Length=MAX(Length,1)

          WRITE(*,*) 'Error: File ',FileName(1:Length),
     & ' doesn''t exist.'
          STOP 'In Routine: GetFEMProb'

        END IF

      ELSE
        STOP 'In Routine: GetFEMProblem Normal End'
      END IF

      RETURN
      END

C ******************************************************************** GetUpper

      SUBROUTINE GetUpper(ElemType,Upper)

      CHARACTER*1 ElemType
      INTEGER Upper

      INCLUDE 'ECFHdr.INC'

C This routine only works for BasisDeg=1

      IF (ElemType.EQ.'S') THEN

C Square type of element

        Upper=(BasisDeg+1)**NoOfDim
```

```
      ELSE IF (ElemType.EQ.'T') THEN

C Triangular type of element

        Upper=3
        IF (BasisDeg.NE.1) THEN
          WRITE(*,*) 'Error: Can''t handle orders higher than 1 with'
          WRITE(*,*) '       triangular elements'
          STOP 'In Routine: GetUpper'
        END IF

      ELSE

C Unknown

        WRITE(*,*) 'Error: Uknown type of element=',ElemType
        STOP 'In Routine: GetUpper'

      END IF

      RETURN
      END

C ************************************************************** PrintProbHeader

      SUBROUTINE PrintProbHeader

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'
      INCLUDE 'ECFbc.INC'
      INCLUDE 'ECFNam.INC'

      CHARACTER*80 Output

C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

C 123456789a123456789b123456789c123456789d12345
C ---------------------------------------------- Finite Element Method Parameters

      WRITE(*,200)
200   FORMAT(X,45('-'),' Finite Element Method Parameters')

      Output=' '

C                    123456789a123456
      IF (Guess) Output='Ext. 1st Guess  '

      IF (Convection) THEN
        IF (PolyFlow) THEN
C                 123456789a1234
          Output='PF Convection '//Output
        ELSE
          Output='External Conv '//Output
        END IF
      ELSE IF (Binary) THEN
        Output='Binary Elec.  '//Output
      ELSE
        Output='              '//Output
```

```
      END IF

      IF (ECFMesh) THEN
C           123456789
        Output='ECF Mesh '//Output
      ELSE
        Output='PF Mesh  '//Output
      END IF

      IF (Debugging) THEN
C           123456
        Output='DEBUG '//Output
      ELSE
        Output='      '//Output
      END IF

      WRITE(*,210) Output,NoOfDim,NoOfEqu
210   FORMAT(X,A50,I2,' dimensions, ',I3,' equations')

C 123456789a123456789b123456789c123456789d123456789e
C DEBUGGING   CONVECTION   EXTERNAL 1ST GUESS      ii dimensions, iii equations

      WRITE(*,220) BasisDeg,Tolerance,NoOfGaussPts,MaxNoOfIter
220   FORMAT(X,'Degree of Basis Functions: ',I3,
     & '   Convergence Tolerance: ',G14.7,/,
     & X,'  Number of Gauss Points: ',I3,6X,
     & 'Maximum No. NR Iter: ',I3)

C 123                    1234
C Degree of Basis Functions: iii   Convergence Tolerance: ddddddddddddd
C   Number of Gauss Points: iii     Maximum No. NR Iter: iii
C                       123456

      RETURN
      END

C ************************************************************************ ReadBC

      SUBROUTINE ReadBC(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFbc.INC'
      INTEGER I,J
      CHARACTER*(*) FileName

C The following statements read the boundary conditions

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      DO J=1,NoOfEqu
        READ(1,*) NoOfCorners(J)
        DO I=1,NoOfCorners(J)
          READ(1,*) Corners(J,I),BCs(J,I),Kinetics(I),BCPar(J,I)
        END DO
      END DO

      CLOSE(1)
```

```
      RETURN
      END

C ******************************************************************** ReadFEMUser

      SUBROUTINE READFEMUser

C This subroutine read the file FEM.IN created by WriteFEMUser

      INCLUDE 'ECFNam.INC'

      CALL ReadMeshHdr(MeshHdrFile)
      CALL ReadMesh(MeshFile)
      CALL ReadBC(BCFile)
      CALL ReadRes(ResFile)
      CALL ReadPar(ParFile)

      RETURN
      END

C ******************************************************************** ReadMesh

      SUBROUTINE ReadMesh(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'
      CHARACTER*(*) FileName
      INTEGER I,J,K

C The following statements read the node locations

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      DO I=1,NoOfNodes
        DO J=1,NoOfDim
          READ(1,*) Nodes(I,J)
        END DO
      END DO

C The following statements read the NLOC matrix for finding the
C node numbers for various element locations.

      DO J=1,NoOfElems
        READ(1,*) ElemType(J)
        DO K=1,(BasisDeg+1)**NoOfDim
          READ(1,*) NLoc(J,K)
        END DO
      END DO

C The following statements read the number of the nodes on the boundary.

      DO I=1,NoOfBDNodes
        READ(1,*) BDNodes(I)
      END DO

      CLOSE(1)

      RETURN
```

```
      END

C ********************************************************************** ReadMeshHdr

      SUBROUTINE ReadMeshHdr(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'

      CHARACTER*(*) FileName
      LOGICAL FileExist

C The following statements read the degree of the basis functions, tolerance
C No of elements, nodes, boundary nodes, coefficients
C and debugging or convection flags, as well as file names.

      INQUIRE(FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) THEN
        WRITE(*,*) 'Error: File ',FileName,', does not exist.'
        STOP 'In Routine: ReadMeshHdr (ECFSUBS)'
      END IF

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      READ(1,*) BasisDeg,Tolerance,NoOfGaussPts,MaxNoOfIter
      READ(1,*) NoOfEqu,NoOfDim
      READ(1,*) NoOfElems,NoOfNodes,NoOfBDNodes
      READ(1,*) NoOfCoef,Nlc,Nuc
      READ(1,*) Debugging,Convection,PolyFlow,Guess,Nonnegative
      READ(1,*) ECFMesh,Binary,Supporting

      CLOSE(1)

      RETURN
      END

C ********************************************************************** ReadNames

      SUBROUTINE ReadNames(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFNam.INC'

      CHARACTER*(*) FileName

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      READ(1,*) MeshHdrFile
      READ(1,*) MeshFile
      READ(1,*) MeshSrcFile
      READ(1,*) BCFile
      READ(1,*) ParFile
      READ(1,*) ResFile
      READ(1,*) PlotFile
      READ(1,*) SlopeFile

      READ(1,*) PFMeshFile
      READ(1,*) PFCoefFile
```

```
      CLOSE(1)

      RETURN
      END

C ******************************************************************** ReadPFCoef

      SUBROUTINE ReadPFCoef(FileName)

      INCLUDE 'POLYCOEF.INC'

      INTEGER io1/1/,J
      CHARACTER*(*) FileName

      OPEN(UNIT=IO1,FILE=FILENAME,STATUS='OLD',FORM='UNFORMATTED')

      READ(io1) PolyNVers
      READ(Io1) PolyNTot,PolyNComp,PolyNNodeR,PolyVerTr
      READ(io1) PolyTitle
      READ(io1) (PolyFil1(J),J=1,PolyNComp)
      READ(io1) (PolyFil2(J),J=1,PolyNComp)

      IF (PolyNVers.NE.20) THEN
        WRITE(*,*) 'Error: Expecting to read data from the version'
        WRITE(*,*) '       20 Polyflow.'
        WRITE(*,*) '       NVers=',PolyNVers
        STOP 'In Routine: ReadPFCoef'
      END IF

      READ(io1) PolyCase,PolyAx,PolyFluid,PolyStres
      READ(io1) PolyTestRH,PolyTestZ

      IF (PolyCase.NE.1) THEN
        WRITE(*,*) 'Error: Expecting to read data for PolyFlow'
        WRITE(*,*) '       for ICASE=1'
        WRITE(*,*) '       ICASE=',PolyCase
        STOP 'In Routine: ReadPFCoef'
      END IF

      READ (io1) PolyFac,PolyTNat,PolyExpO,PolyRO

      IF (PolyNTot.GT.PolyMaxNoOfZ) THEN
        WRITE(*,*) 'Error: Too many coefficients!'
        WRITE(*,*) '           PolyNTot=',PolyNTot
        WRITE(*,*) '       PolyMaxNoOfZ=',PolyMaxNoOfZ
        STOP 'In Routine: ReadPFCoef'
      END IF

      READ (io1) (PolyZ(J),J=1,PolyNTot)

      CLOSE(IO1)

      RETURN
      END

C ******************************************************************** ReadPFMesh

      SUBROUTINE ReadPFMesh(FileName)
```

```
      INCLUDE 'PolyMesh.INC'
      CHARACTER*(*) FileName
      INTEGER I,J,Io1/1/,K
      LOGICAL Error,FileExists

      Error=.FALSE.
      INQUIRE(File=FileName,Exist=FileExists)
      IF (.NOT.FileExists) THEN
        WRITE(*,*) 'Error: The filename doesn''t exist.'
        WRITE(*,*) '       FileName=',Filename
        STOP 'In Routine: ReadPFMesh'
      END IF

      OPEN(UNIT=io1,FILE=Filename,STATUS='OLD',FORM='FORMATTED')

      READ (io1,100) PolyNVert,PolyNNode,PolyNbd,PolyNElem

      IF (PolyNElem.GT.PolyMaxNElem) THEN
        WRITE(*,*) 'Error: PolyNElem is larger than PolyMaxNElem'
        WRITE(*,*) '          PolyNElem=',PolyNelem
        WRITE(*,*) '       PolyMaxNElem=',PolyMaxNElem
        Error=.TRUE.
      END IF

      IF (PolyNBd.GT.PolyMaxNBd) THEN
        WRITE(*,*) 'Error: PolyNBd is larger than PolyMaxNBd'
        WRITE(*,*) '          PolyNBd=',PolyNBd
        WRITE(*,*) '       PolyMaxNBd=',PolyMaxNBd
        ERROR=.TRUE.
      END IF

      IF (PolyNNode.GT.PolyMaxNNode) THEN
        WRITE(*,*) 'Error: PolyNNode is larger than PolyMaxNNode'
        WRITE(*,*) '          PolyNNode=',PolyNNode
        WRITE(*,*) '       PolyMaxNNode=',PolyMaxNNode
        ERROR=.TRUE.
      END IF

      IF (Error) STOP 'In Routine: ReadPFMesh'

      DO I=1,PolyNElem
        READ(io1,100) (PolyNod(J,I),J=1,9)
100     FORMAT(20I4)
      END DO

      READ(io1,100) (PolyBDNod(K),K=1,PolyNBd)
      DO I=1,PolyNNode
        READ(io1,200) PolyX(I),PolyY(I)
200     FORMAT(2E14.7)
      END DO

      CLOSE(io1)

      RETURN
      END

C ********************************************************************** ReadPar
```

```
      SUBROUTINE ReadPar(FileName)

      INCLUDE 'ECFPar.INC'
      CHARACTER*(*) FileName

      OPEN(UNIT=1,STATUS='OLD',FORM='FORMATTED',FILE=FileName)

      READ(1,*) WagnerLin,aA,aC
      READ(1,*) Peclet
      READ(1,*) BinaryK2,SupportingK3

      CLOSE(1)

      RETURN
      END

C ************************************************************************ ReadRes

      SUBROUTINE ReadRes(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFRes.INC'
      INTEGER I,J,K
      CHARACTER*(*) FileName

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      DO I=1,NoOfCoef
        READ(1,*) C(I)
      END DO

      DO I=1,NoOfEqu
        DO J=1,NoOfElems
          DO K=1,(BasisDeg+1)**NoOfDim
            READ(1,*) CLoc(I,J,K)
          END DO
        END DO
      END DO

      CLOSE(1)

      RETURN
      END

C ************************************************************** SearchForPFElem

      SUBROUTINE SearchForPFElem(X,PFElem,Triangular)

C The following subroutine searches the Polyflow mesh element by element.
C The polyflow element containing the point X is returned.

C X()   We wish to find the polyflow element containing this point.
C PFElem        The polyflow element number containing X, or 0 if no
C       element found.
C Triangular    Set to .TRUE. if the polyflow element is triangular

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'POLYMesh.INC'
```

```
      REAL*8 X(MaxNoOfDim),ThetaSum,p1x,p1y,p2x,p2y,
     & Pi/3.141592/

      INTEGER PFElem,Nv,I,J,Jp1
      LOGICAL Triangular

      DO I=1,PolyNElem

        Triangular=(PolyNod(8,I).EQ.0).AND.(PolyNod(9,I).EQ.0)
        ThetaSum=0.0

        IF (Triangular) THEN
          Nv=3
        ELSE
          Nv=4
        END IF

C The following statements compute the angles

        DO J=1,Nv

          Jp1=J+1
          IF (Jp1.GT.Nv) Jp1=1

          p1X=PolyX(PolyNod(J,I))-X(1)
          p1Y=PolyY(PolyNod(J,I))-X(2)
          p2X=PolyX(PolyNod(Jp1,I))-X(1)
          p2Y=PolyY(PolyNod(Jp1,I))-X(2)

          IF (((P1X.EQ.0.0).AND.(P1Y.EQ.0.0)).OR.
     & ((P2X.EQ.0.0).AND.(P2Y.EQ.0.0))) THEN
            PFElem=I
            RETURN
          END IF

          ThetaSum=ThetaSum+ACOS((p1X*p2X+p1Y*P2Y)/
     & SQRT((P1X**2+P1Y**2)*(P2X**2+P2Y**2)))

        END DO

        IF (ThetaSum.GE.2.0*Pi) THEN
          PFElem=I
          RETURN
        END IF

      END DO
      PFElem=0

      RETURN
      END

C ********************************************************************** Velocity

      SUBROUTINE Velocity(X,Vlcty)

C The subroutine computes the velocity Vlcty() calculated by PolyFlow at the
C location X().
```

```
      REAL*8 X(*),Vlcty(*)
      LOGICAL Found

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'PolyMesh.INC'
      INCLUDE 'PolyCoef.INC'

      PARAMETER Np=9

      INTEGER Elem,PFElem,Indx(Np),NoOfPhis,I,J
      REAL PtMat(Np,Np),PhiCoef(Np,Np),Col(Np),Phis(Np)
      LOGICAL Triangular

      IF (.NOT.PolyFlow) THEN
        WRITE(*,*) 'Error: No external subroutine linked into ECCOMPUTE.FOR'
        WRITE(*,*) '       for computing velocity'
        STOP 'In Routine: Velocity (internal)'
      END IF

      DO I=1,NoOfDim
        Vlcty(I)=0.0
      END DO

      CALL SearchForPFElem(X,PFElem,Triangular)

C PFElem is zero when no polyflow element could be found. The following
C IF-THEN block tests for this and if true, tells the user and stops the
C program.

      IF (PFElem.EQ.0) THEN

        IF (PolyFlow) THEN
          WRITE(*,*) 'Error: point X() not found',
     & ' within polyflow mesh'
        ELSE
          WRITE(*,*) 'Error: point X() not found',
     & ' within external subroutine Velocity'
        END IF

        DO I=1,NoOfDim
          WRITE(*,*) '       X(',I,')=',X(I)
        END DO
        STOP 'In Routine: Velocity (internal)'
      END IF

C The following do-loop makes the inverse of the matrix that will be used to
C compute phis for PolyFlow elements

      IF (Triangular) THEN
        NoOfPhis=6
      ELSE
        NoOfPhis=9
      END IF

      DO I=1,NoOfPhis

        PtMat(I,1)=1.0
        PtMat(I,2)=PolyX(PolyNod(I,PFElem))
```

```
          PtMat(I,3)=PolyY(PolyNod(I,PFElem))
          PtMat(I,4)=PolyX(PolyNod(I,PFElem))*PolyY(PolyNod(I,PFElem))
          PtMat(I,5)=PolyX(PolyNod(I,PFElem))*PolyX(PolyNod(I,PFElem))
          PtMat(I,6)=PolyY(PolyNod(I,PFElem))*PolyY(PolyNod(I,PFElem))

          IF (.NOT.Triangular) THEN
            PtMat(I,7)=PolyX(PolyNod(I,PFElem))
     & *PolyY(PolyNod(I,PFElem))*PolyY(PolyNod(I,PFElem))
            PtMat(I,8)=PolyX(PolyNod(I,PFElem))
     & *PolyX(PolyNod(I,PFElem))*PolyY(PolyNod(I,PFElem))
            PtMat(I,9)=PolyX(PolyNod(I,PFElem))
     & *PolyX(PolyNod(I,PFElem))*PolyY(PolyNod(I,PFElem))
     & *PolyY(PolyNod(I,PFElem))
          END IF

        END DO

C The following CALL finds the inverse of the matrix

        CALL MatInv(PtMat,PhiCoef,Indx,Col,NoOfPhis,Np)

C The following Statements compute the position vector Col
C for use in computing the Phis at the point X()

        Col(1)=1
        Col(2)=X(1)
        Col(3)=X(2)
        Col(4)=X(1)*X(2)
        Col(5)=X(1)*X(1)
        Col(6)=X(2)*X(2)

        IF (.NOT.Triangular) THEN
          Col(7)=X(1)*X(2)*X(2)
          Col(8)=X(1)*X(1)*X(2)
          Col(9)=X(1)*X(1)*X(2)*X(2)
        END IF

C The following DO-Loop computes the PHis for the PolyFlow Element

        DO I=1,NoOfPhis
          Phis(I)=0.0

          DO J=1,NoOfPhis
            Phis(I)=Phis(I)+PhiCoef(J,I)*Col(J)
          END DO
        END DO

C The following DO-loop computes the velocity in x and y directions.

        DO I=1,NoOfDim
          Vlcty(I)=0.0

          DO J=1,NoOfPhis
            Vlcty(I)=Vlcty(I)+Phis(J)*
     & PolyZ(PolyNod(J,PFElem)+(I-1)*PolyNNode)
          END DO
        END DO
```

```
      RETURN
      END

C ********************************************************************** WriteBC

      SUBROUTINE WriteBC(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFbc.INC'
      INTEGER I,J
      CHARACTER*(*) FileName

C The following statements Write the boundary conditions

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')

      DO J=1,NoOfEqu
        WRITE(1,*) NoOfCorners(J)
        DO I=1,NoOfCorners(J)
          WRITE(1,*) Corners(J,I),',''',BCs(J,I),''',''',
     & Kinetics(I),''',',BCPar(J,I)
        END DO
      END DO

      CLOSE(1)

      RETURN
      END

C ******************************************************************** WriteFEMUser

      SUBROUTINE WriteFEMUser

      INCLUDE 'ECFNam.INC'
      INCLUDE 'ECFHdr.INC'

      CALL WriteMeshHdr(MeshHdrFile)
      CALL WriteMesh(MeshFile)
      CALL WriteBC(BCFile)
      CALL WriteRes(ResFile)
      CALL WritePar(ParFile)

      RETURN
      END

C ********************************************************************** WriteMesh

      SUBROUTINE WriteMesh(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFMsh.INC'
      CHARACTER*(*) FileName
      INTEGER I,J,K

C The following statements Write the node locations

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')
```

```
      DO I=1,NoOfNodes
        DO J=1,NoOfDim
          Write(1,*) Nodes(I,J)
        END DO
      END DO

C The following statements Write the NLOC matrix for finding the
C node numbers for various element locations.

      DO J=1,NoOfElems
        WRITE(1,*) '''',ElemType(J),''''
        DO K=1,(BasisDeg+1)**NoOfDim
          Write(1,*) NLoc(J,K)
        END DO
      END DO

C The following statements Write the number of the nodes on the boundary.

      DO I=1,NoOfBDNodes
        Write(1,*) BDNodes(I)
      END DO

      CLOSE(1)

      RETURN
      END

C ******************************************************************** WriteMeshHdr

      SUBROUTINE WriteMeshHdr(FileName)

      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFGauss.INC'

      CHARACTER*(*) FileName

C The following statements Write the degree of the basis functions, tolerance
C No of elements, nodes, boundary nodes, coefficients
C and debugging or convection flags, as well as file names.

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')

      Write(1,*) BasisDeg,',',Tolerance,',',NoOfGaussPts,',',MaxNoOfIter
      Write(1,*) NoOfEqu,',',NoOfDim
      Write(1,*) NoOfElems,',',NoOfNodes,',',NoOfBDNodes
      Write(1,*) NoOfCoef,Nlc,Nuc
      Write(1,*) Debugging,',',Convection,',',PolyFlow,',',Guess,
     & ',',Nonnegative
      WRITE(1,*) ECFMesh,',',Binary,',',Supporting

      CLOSE(1)

      RETURN
      END

C ******************************************************************** WriteNames

      SUBROUTINE WriteNames(FileName)
```

```
      INCLUDE 'ECFHdr.INC'
      INCLUDE 'ECFNam.INC'

      CHARACTER*(*) FileName
      INTEGER Length

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')

      CALL StringLength(MeshHdrFile,Length)
      Write(1,*) '''',MeshHdrFile(1:Length),''''
      CALL StringLength(MeshFile,Length)
      Write(1,*) '''',MeshFile(1:Length),''''
      CALL StringLength(MeshSrcFile,Length)
      Write(1,*) '''',MeshSrcFile(1:Length),''''
      CALL StringLength(BCFile,Length)
      Write(1,*) '''',BCFile(1:Length),''''
      CALL StringLength(ParFile,Length)
      Write(1,*) '''',ParFile(1:Length),''''
      CALL StringLength(ResFile,Length)
      Write(1,*) '''',ResFile(1:Length),''''

      CALL StringLength(PlotFile,Length)
      WRITE(1,*) '''',PlotFile(1:Length),''''
      CALL StringLength(SlopeFile,Length)
      WRITE(1,*) '''',SlopeFile(1:Length),''''

      CALL StringLength(PFMeshFile,Length)
      IF (Length.LE.0) Length=1
      Write(1,*) '''',PFMeshFile(1:Length),''''
      CALL StringLength(PFCoefFile,Length)
      IF (Length.LE.0) Length=1
      Write(1,*) '''',PFCoefFile(1:Length),''''

      CLOSE(1)

      RETURN
      END

C ********************************************************************** WritePar

      SUBROUTINE WritePar(FileName)

      INCLUDE 'ECFPar.INC'
      CHARACTER*(*) FileName

      OPEN(UNIT=1,STATUS='NEW',FORM='FORMATTED',FILE=FileName)

      Write(1,*) WagnerLin,',',aA,',',aC
      Write(1,*) Peclet
      WRITE(1,*) BinaryK2,SupportingK3

      CLOSE(1)

      RETURN
      END

C ********************************************************************** WriteRes
```

```
SUBROUTINE WriteRes(FileName)

INCLUDE 'ECFHdr.INC'
INCLUDE 'ECFRes.INC'

INTEGER I,J,K
CHARACTER*(*) FileName

OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')

DO I=1,NoOfCoef
  WRITE(1,*) C(I)
END DO

DO I=1,NoOfEqu
  DO J=1,NoOfElems
    DO K=1,(BasisDeg+1)**NoOfDim
      WRITE(1,*) CLoc(I,J,K)
    END DO
  END DO
END DO

CLOSE(1)

RETURN
END
```

```
C Include File: ECFbc.INC
C Written by: Ken Jordan                    Date: 8/3/87

C This INCLUDE file defines and dimensions the variables in COMMON block
C FEMbc.  The variables are described in program FEM2d.FOR

C ------------------------------------------------------------------ Parameters

C MaxNoOfCorners      The maximum number of boundary condition changes
C                     allowed.  (i.e. the maximum number of insulators, electrodes,
C                     ...)

C ------------------------------------------------------------------- Variables

C BCPar(j,i)          The voltage to apply to side i for equation j.
C                     This may also be a parameter used for computing
C                     boundary values for side i and equation j.
C BCs(,)              The type of boundary condition to apply to each
C                     side (BCs(,i) applies to Nodes(Corners(,i))-Nodes(Corners(,i+1))
C                     The first index is the equation number.
C Corners(,)          The node numbers for the corners.  The first index is the
C                     equation number, the second index is the corner number,
C                     the contents are the boundary node number for the corner.
C Kinetics()          The type of kinetics to use for each side.
C NoOfCorners         The number of corner points

C ------------------------------------------------------------------- Declarations

      PARAMETER MaxNoOfCorners=4
      COMMON /ECFbc/ Corners,NoOfCorners,BCs,Kinetics,BCPar

      INTEGER Corners(MaxNoOfEqu,MaxNoOfCorners),
     & NoOfCorners(MaxNoOfEqu)
      REAL*8 BCPar(MaxNoOfEqu,MaxNoOfCorners)
      CHARACTER*1 BCs(MaxNoOfEqu,MaxNoOfCorners)
      CHARACTER*2 Kinetics(MaxNoOfCorners)
```

```
C Include File: ECFCal.INC
C Written By: Ken Jordan                                  Date: 8/3/87

C This INCLUDE file defines and dimensions the variables in COMMON block
C FEMCalcs. The variables are described in program FEM2d.FOR
C Note: Assumes FEMMesh.INC precedes it.

C ------------------------------------------------------------------ Variables

C R()            The Residual vector for the last guess.
C RJacobian(,)   The jacobian (dR(i)/dcoefficient(j))

C ------------------------------------------------------------------ Declarations

      COMMON /ECFCal/R,RJacobian

      REAL*8 R(MaxNoOfCoef),RJacobian(MaxBandWidth,MaxNoOfCoef)
```

```
C Include File: ECFEqu.INC
C Written By: Ken Jordan

C This file is only used when integrating concentrations (e.g. in binary,
C supporting or complex problems to get k's)

C Equ    The equation number being solved.  Should be 1 for binary and
C        supporting

C Species        The species number being integrated
C        1=depositing species
C        2=depositing species anion
C        3=supporting cation

         INTEGER Equ,Species
         COMMON /ECFEQU/Equ,Species
```

```
C Include File: ECFGauss
C Written By: Ken Jordan                                    7/10/90

C This file contains declarations for gaussian integration points.

C NoOfGaussPts          The number of points used in the numerical integration.
C Pts(NoOfGaussPts)     The points to be used in the evaluation of the
C                       integral.
C Wts(NoOfGaussPts)     The weights to be used at each Gauss point.

      PARAMETER MaxNoOfGaussPts=5

      COMMON /ECFGauss/ Pts,Wts,NoOfGaussPts

      INTEGER NoOfGaussPts
      DOUBLE PRECISION Pts(MaxNoOfGaussPts),Wts(MaxNoOfGaussPts)
```

```
C Include File: ECFHdr.INC
C Written by: Ken Jordan                    Date: 8/3/87

C This include file defines and dimensions variables in COMMON block ECFHdr

C ------------------------------------------------------------------ Parameters

C MaxNoOfNodes        The maximum number of nodes allowed (A Parameter)
C MaxBandWidth        The maximum bandwidth for the problem
C MaxNoOfEqu The maximum number of equations the ECF program can handle
C MaxNoOfDim The maximum number of dimensions
C MaxNoOfBasisDeg     The maximum degree of basis functions.
C MaxNoOfCoef The maximum number of coefficients

C ------------------------------------------------------------------ Variables

C BasisDeg      The number      1 for linear Lagrange Basis Functions
C                               2 for quadratic Lagrange Basis Functions
C                               3 for cubic Hermite Basis Functions
C Convection    .True. if the user wishes to include convective terms
C               (program computes velocity in subroutine
C Binary        .TRUE. if the user is solving a binary electrolyte problem.
C               (program uses kappa for diffusivity, kinetics have conc. dep.)
C Supporting    .TRUE. if the user is solving a tertiary electrolyte problem.
C Debugging     .TRUE. if want CLoc, RJAcobian, ... printed out between
C               iterations
C Guess         .TRUE. if the user wants the first guess computed in
C               program ECDEFINE.
C MaxNoOfIter   The maximum number of iterations allowed before NewtonRaphson
C               routine stops
C Nlc           The number of lower codiagonals for the RJacobian Matrix
C NoOfBDNodes         The number of nodes on the boundary
C NoOfDim       The number of dimensions.
C NoOfElems     The number of elements
C NoOfEqu       The number of equations being solved
C NoOfNodes     The number of distinct nodes
C Nonnegative   .TRUE. if the coefficients are not allowed to be negative.
C Nuc           The number of upper codiagonals for the RJacobian Matrix
C PolyFlow      .TRUE. if the program is to use POLYFLOW results for Convection.
C Tolerance     The norm of the C(N+1)-C(N) must be less than this for
C               solution to be considered converged.
C ECFMesh       .TRUE. if mesh was generated by ECFMESH (.FALSE. if mesh
c               was generated by POLYMESH)

C ------------------------------------------------------------------ Declarations

      PARAMETER MaxNoOfNodes=2000
      PARAMETER MaxBandWidth=100
      PARAMETER MaxNoOfEqu=1
      PARAMETER MaxNoOfDim=2
      PARAMETER MaxNoOfBasisDeg=3
      PARAMETER MaxNoOfCoef=MaxNoOfEqu*MaxNoOfNodes

      COMMON /ECFHdr/ BasisDeg,Tolerance,MaxNoOfIter,
     & NoOfDim,NoOfEqu,
     & NoOfElems,NoOfNodes,NoOfBDNodes,
     & NoOfCoef,Nlc,Nuc,
     & Debugging,Convection,Guess,PolyFlow,Nonnegative,
```

```
& ECFMesh,Binary,Supporting

      INTEGER BasisDeg,NoOfElems,NoOfNodes,NoOfBDNodes,
& NoOfCoef,NoOfDim,NoOfEqu,Nlc,Nuc,MaxNoOfIter
      LOGICAL Debugging,Convection,PolyFlow,Guess,Nonnegative
      LOGICAL ECFMesh,Binary,Supporting
      REAL*8 Tolerance
```

```
C Include File: ECFHdr2.INC
C Written by: Ken Jordan                    Date: 8/3/87

C This include file defines and dimensions variables in COMMON block ECFHdr2

C ------------------------------------------------------------------ Variables

C BasisDeg2     The number      1 for linear Lagrange Basis Functions
C                               2 for quadratic Lagrange Basis Functions
C                               3 for cubic Hermite Basis Functions
C NoOfDim2      The number of dimensions.
C NoOfElems2    The number of elements
C NoOfEqu2      The number of equations being solved
C NoOfNodes2    The number of distinct nodes

C ------------------------------------------------------------------ Declarations

      COMMON /ECFHdr2/ BasisDeg2,NoOfDim2,NoOfEqu2,NoOfElems2,NoOfNodes2,
     & NoOfCoef2

      INTEGER BasisDeg2,NoOfElems2,NoOfNodes2,
     & NoOfCoef2,NoOfDim2,NoOfEqu2
```

```
C Include File: ECFMAC.INC
C Written By: Ken Jordan

C Parameters

C MaxNoOfMacVert
C MaxNoOfMacElem
C MaxNoOfMicVert

C Variables

C NoOfMacVert The number of macrovertices.
C MacVertX()   X location of macrovertices
C MAcVertY()   Y location of macrovertices

C NoOfMacElem          The number of macro elements
C MacElemType()                'S' for square elements, 'T' for triangular.
C MacElemVert(,4)      Macrovertex numbers that make up the element
C                      (Macro element number, vertex number)
C                      The following conventions apply:

C                              Square macro elements
C                              4-3              +3+
C                              ||<- Vert. No.   4 2 <- Side No.
C                              1-2              +1+

C                              Triangular Macro Elements
C                              2                *
C                              |               |C                    *-* <- Vert. No.    2-1
C                              ||              |C                    3_*_1               *-3-*

C NoOfMicElm(,4)       (Mac Elem No, Side), number of microelements to split
C                      the side up into.
C SpaType(,4)          (Mac Elem No, Side), spacing to use on the side.
C                      'LIN' for linear spacing (currently the only type
C                      known
C SpaFac(,4)           (Mac Elem No, Side), factor for spacing on the side.

c NoOfMacBDNodes       The number of macrovertices on the boundary
c MacBDNodeNo(MaxNoOfMacVert)
C                      The micro node number of the macro vertex. Used
C                      in printing reports on mesh, and in setting up B.C.
c MacBDNodes(MaxNoOfMacVert)
C                      The macrovertex numbers in the order that make up the
C                      boundary.

c MacNodes(MaxNoOfMicVert+1,MaxNoOfMicVert+1)
C                      (MicNode X, Mic Node Y)
C                      The node numbers of the microvertices in the whole
C                      macroelement. Two cases.

C                      Square macro element.
C
c                      1,2--2,2--3,2  etc.
c                      |   |   |
c                      1,1--2,1--3,1

C                      Triangular Macro Element
```

```
C                    1,3
C                    |c                    | C                    1,2--2,2
c                    |  c                  |  | c                 1,1--2,1--3,1

c SharedNodes(MAxNoOfMacElem,4,MaxNoOfMicVert+1)
C                    (Mac Elem No, Side, Micro Vert. No)
C                    The node numbers of micro vertices on the sides of each
C                    macro element.

      PARAMETER MaxNoOfMacVert=10
      PARAMETER MaxNoOfMacElem=10
      PARAMETER MaxNoOFMicVert=50

      INTEGER NoOfMacVert,NoOfMacElem,
     & MacElemVert(MaxNoOfMacElem,4),NoOfMicElm(MaxNoOfMacElem,4)
      INTEGER SharedNodes(MAxNoOfMacElem,4,MaxNoOfMicVert+1)
      INTEGER MacNodes(MaxNoOfMicVert+1,MaxNoOfMicVert+1)
      INTEGER MacBDNodes(MaxNoOfMacVert),NoOfMacBDNodes
      INTEGER MacBDNodeNo(MaxNoOfMacVert)
      REAL*8 MacVertX(MaxNoOfMacVert),MacVertY(MaxNoOfMacVert),
     & SpaFac(MaxNoOfMacElem,4)
      CHARACTER*4 SpaType(MaxNoOfMacElem,4)
      CHARACTER*1 MacElemType(MaxNoOfMacElem)

      COMMON /ECFMAC/NoOfMacVert,NoOfMacElem,MacElemVert,NoOfMicElm,
     & MacVertX,MacVertY,SpaFac,SpaType,SharedNodes,MacNodes,
     & NoOfMacBDNodes,MacBDNodes,MacBDNodeNo,MacElemType
```

```
C Include File: ECFMsh.INC
C Written by: Ken Jordan                    Date: 8/3/87

C This include file defines and dimensions variables in COMMON block
C ECFMsh.  Read to and written from ReadMesh, WriteMesh()

C ------------------------------------------------------------------ Variables

C BdNodes()     The vertex number of nodes on the boundary.  (In order
C               around the boundary.)

C NLoc(,)       The first index is the global element number (numbered
C               from the lower left corner and up)
C               The second index is the local node number
C               (1-> (BasisDeg+1)^(NoOfDim)
C               The contents are the Node location in matrix NODES.

C Nodes(,)      The location of the nodes.
C               The first index is the node number (1-> NoOfNodes)
C               The second index is coordinate number (1=x, 2=y)

C ElemType()    The type of element 'S' for square and 'T' for triangular.

C ------------------------------------------------------------------ Declarations

      COMMON /ECFMsh/ Nodes,NLoc,BDNodes,ElemType

      INTEGER NLoc(MaxNoOfNodes,(MaxNoOfBasisDeg+1)**MaxNoOfDim),
     & BDNodes(MaxNoOfNodes)
      REAL*8 Nodes(MaxNoOfNodes,MaxNoOfDim)
      CHARACTER*1 ElemType(MaxNoOfNodes)
```

```
C Include File: ECFMsh2.INC
C Written by: Ken Jordan                    Date: 8/3/87

C This include file defines and dimensions variables in COMMON block
C ECFMsh.  Read to and written from ReadMesh, WriteMesh()

C ------------------------------------------------------------------ Variables

C NLoc2(,)      The first index is the global element number (numbered
C               from the lower left corner and up)
C               The second index is the local node number
C               (1-> (BasisDeg+1)^(NoOfDim)
C               The contents are the Node location in matrix NODES.

C Nodes2(,)     The location of the nodes.
C               The first index is the node number (1-> NoOfNodes)
C               The second index is coordinate number (1=x, 2=y)

C ElemType2()   The type of element 'S' for square and 'T' for triangular.

C ------------------------------------------------------------------ Declarations

      COMMON /ECFMsh2/ Nodes2,NLoc2,ElemType2

      INTEGER NLoc2(MaxNoOfNodes,(MaxNoOfBasisDeg+1)**MaxNoOfDim)
      REAL*8 Nodes2(MaxNoOfNodes,MaxNoOfDim)
      CHARACTER*1 ElemType2(MaxNoOfNodes)
```

```
C Include File: ECFNam.INC
C Written by: Ken Jordan                    Date: 8/11/87

C ------------------------------------------------------------------ Variables

C BCFile        The name of the *.BC file.
C MeshFile      The filename for the *.MSH file
C MeshHdrFile   The filename for the *.HDR file
C MeshSrcFile   The filename of the polyflow mesh file that serves as the
C               source for the *.MSH file
C ParFile       The name of the *.PAR file.
C PFMeshFile    The filename for the hydrodynamic flow mesh. (from POLYFLOW)
C PFCoefFile    The filename for the hydrodynamic coefficients(from POLYFLOW)
C PlotFile      The name of the *.ISO file containing line segments
C               of constant value.
C ResName       The filename for the results file (also used for
C               first guess.)
C SlopeFile     The name of the *.SLP file containing the slope of coefficients
C               along the boundary.

C ------------------------------------------------------------------ Declarations

      CHARACTER*72 MeshHdrFile,MeshFile,MeshSrcFile,BCFile,ParFile,
     & PlotFile
      CHARACTER*72 PFMeshFile,PFCoefFile,ResFile,SlopeFile

      COMMON /ECFNam/MeshHdrFile,MeshFile,BCFile,ParFile,
     & ResFile,PFMeshFile,PFCoefFile,PlotFile,SlopeFile,
     & MeshSrcFile
```

```
C Include File: ECFPar.INC
C Written by: Ken Jordan                    Date: 8/3/87

C This INCLUDE file defines and dimensions the variables in COMMON block
C ECFPar. The variables are described here. They
C represent parameters used to define kinetic and convection conditions.

C ------------------------------------------------------------------ Variables

C Kinetic Variables

C WagnerKLin  Linear wagner number for kinetics to ohmic
C             kappa R T / L F i0
C aA          The anodic transfer Coefficient
C aC          The cathodic transfer Coefficient
C BinaryK2    Integration constant when considering binary electrolyte
C SupportingK3 Integration constant when considering supporting electrolyte.

C Convection Variables

C PecletThe peclet number for convection problems

C ------------------------------------------------------------------ Declarations

      COMMON /ECFPar/ WagnerLin,aA,aC,Peclet,
     & BinaryK2,SupportingK3

      REAL*8 aA,aC,WagnerLin
      REAL*8 Peclet,BinaryK2,SupportingK3
```

```
C Include File: ECFPlot.INC
C Written By: Ken Jordan

C This file contains variables used internally to program ecfplot and
C subroutines. This file just saves retyping all of this stuff in each routine
C and in each call pass list.

C XBase()       The starting point for the current segment
C dX()          The total step across the segment (if straight)
C Distance      The total distance moved along the bondary so far.
C X,Y           The current X,Y location plotted.
C XLast,YLast   The last x,y locatio plotted.
C Normal()      The outward pointing unit normal to the surface.

      REAL*8 Distance,X,Y,XLast,YLast
      REAL*8 XBase(MaxNoOfDim),dX(MaxNoOfDim),
     & Normal(MaxNoOfDim)

      COMMON /ECFPlot/Distance,X,Y,XLast,YLast,XBase,dX,Normal
```

```
C Include File: ECFRes.INC
C Written by: Ken Jordan                    Date: 8/11/87

C This file contains the dimensions and common block necessary for accessing
C the ECF results.

C ------------------------------------------------------------------ Variables

C C()          The coefficients for the basis functions.
C CLoc(,,)     The first index is the equation number (1=potential)
C              The second index is the global element number (numbered
C              from the lower left corner and up)
C              The third index is the local node number (1-> (BasisDeg+1)^2)
C              The contents are the Global Coef location.
C              (Some people call this the NOP array)

C ------------------------------------------------------------------ Declarations

      COMMON /ECFRes/C,CLoc

      INTEGER CLoc(MaxNoOfEqu,MaxNoOfNodes,
     & (MaxNoOfBasisDeg+1)**(MaxNoOfDim))
      REAL*8 C(MaxNoOfCoef)
```

```
C Include File: ECFRes2.INC
C Written by: Ken Jordan                    Date: 8/11/87

C This file contains the dimensions and common block necessary for accessing
C the ECF results.

C ------------------------------------------------------------------ Variables

C C2()          The coefficients for the basis functions.
C CLoc2(,,)     The first index is the equation number (1=potential)
C               The second index is the global element number (numbered
C               from the lower left corner and up)
C               The third index is the local node number (1-> (BasisDeg+1)^2)
C               The contents are the Global Coef location.
C               (Some people call this the NOP array)

C ------------------------------------------------------------------ Declarations

      COMMON /ECFRes2/C2,CLoc2

      INTEGER CLoc2(MaxNoOfEqu,MaxNoOfNodes,
     & (MaxNoOfBasisDeg+1)**(MaxNoOfDim))
      REAL*8 C2(MaxNoOfCoef)
```

```
C Include File: ECFTriPts.INC
C Written By: Ken Jordan

C This file contains integration points to use for triangular elements.
C Currently, only NoOfTriPts=4 is allowed.  See Burnett, p 596 for others

      COMMON /ECFTriPts/NoOfTriPts,TriPts,TriWts

      INTEGER NoOfTriPts
      REAL*8 TriPts(10,2),TriWts(10)
```

```
C Include File: PolyCoef.INC
C Written By: Ken Jordan                                    Date: 8/5/87

C This include file contains the declarations and common block necessary
C to access the polyflow coefficient data.

C ------------------------------------------------------------------ Variables

C PolyNVers     The polyflow version number that made the coefficient file.
C PolyNtot      The total number of coefficients.
C PolyNComp     The number of ??
C PolyNNodeR    ?
C PolyVerTr     ?
C PolyTitle     The title
C PolyFil1(*)
C PolyFil2(*)
C PolyCase      The type of problem that POLYFLOW solved
C PolyAx        ?
C PolyFluid     ?
C PolyStres     ?
C PolyTestRH    ?
C PolyTestZ     ?
C PolyFac       ?
C PolyTNat      ?
C PolyExpO      ?
C PolyRO
C PolyZ(*)      The Coefficient Values at each node location
C               Stored in the form U1, U2, U3, ... UNNode,
C                                  V1, ...
C                                      P1, ...
C                                      Psi1, ...

C ------------------------------------------------------------------ Declarations

      PARAMETER PolyMaxNoOfZ=8000

      INTEGER PolyNVers,PolyNTot,PolyNComp,PolyNNodeR,PolyVerTr
      INTEGER PolyTitle(19)
      INTEGER PolyFil1(20),PolyFil2(20),PolyCase,PolyAx,PolyFluid
      INTEGER PolyStres
      REAL*8 PolyTestRH,PolyTestZ,PolyFac,PolyTNat,PolyExpO,PolyRO
      REAL*8 PolyZ(PolyMaxNoOfZ)

      COMMON /PolyCoef/PolyNVers,PolyNTot,PolyNComp,PolyNNodeR,
     & PolyVerTr,PolyTitle,PolyFil1,PolyFil2,PolyCase,PolyAx,
     & PolyFluid,PolyStres,PolyTestRH,PolyTestZ,PolyFac,PolyTNat,
     & PolyExpO,PolyRo,PolyZ
```

```
C Include File: PolyMesh.INC
C Written by: Ken Jordan                    Date: 8/5/87

C This file contains the declarations and common block required to
C read the polyflow mesh.

C ------------------------------------------------------------------ Variables

C PolyNVert      The number of vertices
C PolyNNode      The number of nodes
C PolyNbd        The number of points on the boundary
C PolyNElem      The number of elements
C PolyNod(9,*)   The nodes locations for each element
C PolyBDNod(*)The ?? Boundary nodes
C PolyX(*)       The X points of the mesh
C PolyY(*)       The Y Points of the mesh

C ------------------------------------------------------------------ Declarations

      PARAMETER PolyMaxNElem=1300
      PARAMETER PolyMaxNBd=360
      PARAMETER PolyMaxNNode=5400

      INTEGER PolyNVert,PolyNNode,PolyNBd,PolyNElem
      INTEGER PolyNod(9,PolyMaxNElem)
      INTEGER PolyBDNod(PolyMaxNBd)

      REAL PolyX(PolyMaxNNode),PolyY(PolyMaxNNode)

      COMMON /PolyMesh/PolyNVert,PolyNNode,PolyNBd,PolyNElem,
     & PolyNod,PolyBDNod,PolyX,PolyY
```

```
      PROGRAM ECBPAR

C This program generates the *.PAR file

      CHARACTER*80 FileName
      INTEGER FilLen

      CALL ChInp('Dimensional File? (No .DIM)',FileName)

      CALL StringLength(FileName,FilLen)
      CALL ReadUserParams(FileName(1:FilLen)//'.DIM')
      CALL PrintUserParams(FileName(1:FilLen)//'.DIM')
      CALL WritePar(FileName(1:FilLen)//'.PAR')

      STOP 'Normal End'
      END

C ************************************************************** PrintUserParams

      SUBROUTINE PrintUserParams(FileName)

      CHARACTER*(*) FileName

      INCLUDE 'ECBPAR.INC'
      INCLUDE 'ECBPar2.INC'

      INTEGER FilLen

10    FORMAT(X,55('*'),' Constants & Parameters')

C 123456789a123456789b123456789c123456789d123456789e12345
C ******************************************************** Constants & Parameters

20    FORMAT(X,'  Fundamental:',7X,'F =',F12.0,' C/mol')

C             1234567   123456789a12
C   Fundamental:      F = ddddd.de+dd C/mol

30    FORMAT(X,'     Additive:  C Bulk =',G10.3,' mol/l',/
     & X,19X,'   D =',G10.3,' cm^2/s')

35    FORMAT(X,'Current Revrsl:  CRRatio=',G10.3,
     & ' i cathodic/i anodic',
     & /,X,     '              WaLinPCR=',G10.3)

C 123         12         123456789
C      Additive:  C Bulk = d.ddde+dd mol/l
C 123456789a123456789
C                      D = d.dddc+dd cm^2/s

40    FORMAT(X,4X,'  Kinetic: alpha A =',F6.3,
     & 11X,'     i0 =',F7.2,' mA/cm^2',
     & /,X,13X,'  alpha C =',F6.3)

C 1234                 12345123456789a1     123456
C      Kinetic: alpha A = d.ddd          i0 = ddd.dd mA/cm^2
C 123456789a123                 123456789
C              alpha C = d.ddd
```

```
50      FORMAT(X,'  Electrolyte:   kappa =',F6.3,' mho/cm')

C   Electrolyte:   kappa = d.ddd mho/cm

60      FORMAT(X,'Characteristic:   xstar =',G12.4,' microns',
   & /,X,      '                phistar =',G12.4,' V',
   & /,X,      '                  istar =',G12.4,' mA/cm^2',
   & 5X,'idep =',G12.4,' mA/cm^2',
   & /,X,      '                          ',12X,  '        ',
   & 5X,'BC 4 =',G12.4,
   & /,X,      '                  tstar =',G12.4,' s')

C                     123456
C Characteristic:   xstar = ddd.d microns
C                 istar = ddd.ddd mA/cm^2
C                     12345678
C               phistar = ddd.ddd V

70      FORMAT(X,78('*'))

C        1        2        3        4        5        6        7
C 123456789a123456789b123456789c123456789d123456789e123456789f123456789g12345678
C ******************************************************************************

        WRITE(*,10)
        CALL StringLength(FileName,FilLen)
        WRITE(*,*) 'File=',FileName(1:FilLen)
        WRITE(*,*) ' '
        WRITE(*,20) F
        WRITE(*,*) ' '
        IF (CorLev) THEN
          WRITE(*,30) CaBulk,Da
          WRITE(*,*) ' '
        END IF
        IF (PerRev) THEN
          WRITE(*,35) CRRatio,WaLinPCR
        END IF
        WRITE(*,40) aA,i0,aC
        WRITE(*,*) ' '
        WRITE(*,50) kappa
        WRITE(*,*) ' '
        WRITE(*,60) xstar,phistar,istar,idep,(idep/istar),tstar
        WRITE(*,*) ' '
        WRITE(*,70)
        WRITE(*,*) ' '
        WRITE(*,80)
        WRITE(*,*) ' '
        IF (CorLev) THEN
          WRITE(*,90) WagLin,Corros,WagLin/(idep/i0),idep/i0
        ELSE
          IF (PerRev) THEN
            WRITE(*,*) '      A N O D I C'
            WRITE(*,*) ' '
            WRITE(*,91) WagLin,WagLin/(idep/i0),idep/i0
            WRITE(*,*) ' '
            WRITE(*,*) '      C A T H O D I C'
            WRITE(*,*) ' '
            WRITE(*,91) WagLin*WaLinPCR,
```

```
     & WagLin*WaLinPCR/(idep*CRRatio/i0),idep*CRRatio/i0
          ELSE
            WRITE(*,91) WagLin,WagLin/(idep/i0),idep/i0
          END IF
        END IF
        WRITE(*,*) ' '
        WRITE(*,70)
        WRITE(*,*) ' '

80      FORMAT(X,57('*'),' Dimensionless Groups')

C 123456789a123456789b123456789c123456789d123456789e1234567
C ********************************************************** Dimensionless Groups

90      FORMAT(X,6X,'WagLin =',G11.4,31X,'Corrosion =',G11.4,
     & /,     X,6X,'WagTaf =',G11.4,
     & /,X,   'Nonlinearity =',G11.4)
91      FORMAT(X,6X,'WagLin =',G11.4,
     & /,     X,6X,'WagTaf =',G11.4,
     & /,X,   'Nonlinearity =',G11.4)

C 123456        123456789a1123456789a123456789b123456789c1
C       WagLin = d.dddde+dd                          Corrosion = d.dddde+dd
C Nonlinearity = d.dddde+dd

        RETURN
        END

C ************************************************************** ReadUserParams

        SUBROUTINE ReadUserParams(FileName)

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBPar.INC'
        INCLUDE 'ECBPar2.INC'
        INCLUDE 'ECGaus.INC'
        INCLUDE 'ECBMAC.INC'

        CHARACTER*(*) FileName

C Read the data in the kinetics file (usually called corros.dat, filename
C is set in *.PAR file for the problem)

        OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

        READ(1,*) F
c             96487
        READ(1,*) CaBulk,Da,    kappa
c             mol/l, cm^2/s,/ohm-cm
        READ(1,*) nelec,aA,aC,i0
c                       mA/cm^2
        READ(1,*) xstar, PhiStar
c             um,    V
        READ(1,*) rho,  MW
C             g/cc, g/mol
        READ(1,*) idep
c                 mA/cm^2
```

```
      READ(1,*) Axisymmetric,CorLev,Spline3D,PerRev
      READ(1,*) MaxIter,SrcFactor,Tolerance
      READ(1,*) KineticsFile
      READ(1,*) NoOfDim,BasisDeg,NoOfGaussPts
      READ(1,*) CRRatio,WaLinPCR
      IF (PerRev) THEN
        READ(1,*) NoOfPCR
        READ(1,*) PCRMax
      END IF
      CLOSE(1)

C Compute characteristic quantities

      istar = (kappa * PhiStar / xstar ) * 10**7
C             mA/cm2

      tstar = rho * nelec * F * xstar / ( MW * istar ) / 10
C             s

C Compute Dimensionless Groups

      WagLin = istar / i0
      Corros = ( Da * nelec * F * CaBulk / Idep / Xstar ) * 10**4

c Note: Corros must be multiplied by (Xstar/delta)(istar/idep)
C       to get the correct physical meaning. (>1=corrosion, <1=net dep)

      RETURN
      END

C ********************************************************************** WritePar

      SUBROUTINE WritePar(FileName)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBPAR2.INC'
      INCLUDE 'ECGaus.INC'

      CHARACTER*(*) FileName
      LOGICAL FileExist
      INTEGER Length,I,Version,KinLen

      Version=6

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')
      WRITE(1,*) Version

      WRITE(1,*) NoOfDim,',',BasisDeg,',',NoOfGaussPts
      WRITE(1,*) Spline3D,',',Axisymmetric
      WRITE(1,*) CorLev,',',PerRev
      WRITE(1,*) Corros
      WRITE(1,*) NoOfPCR,',',CRRatio,',',WaLinPCR,',',PCRMax
      WRITE(1,*) SrcFactor,',',Tolerance,',',MaxIter
      WRITE(1,*) WagLin
      WRITE(1,*) aA,',',aC
      CALL StringLength(KineticsFile,KinLen)
      IF (KinLen.GT.0) THEN
```

```
  WRITE(1,*) '''',KineticsFile(1:KinLen),''''
ELSE
  WRITE(1,*) ''''''
END IF
WRITE(1,*) tstar,istar

WRITE(1,*) ' '
WRITE(1,*) 'Dim, BasisDeg, GaussPts'
WRITE(1,*) 'Spline 3 d?, Axisymmetric ?'
WRITE(1,*) 'Corrosive Levelling?, Periodic Current Reversal?'
WRITE(1,*) 'Corrosion #'
WRITE(1,*) 'NoOfPCR, CRRatio, WaLinPCR, PCRMax'
WRITE(1,*) 'Src Factor, Tolerance, Max Iter'
WRITE(1,*) 'Wagner Number for Linear Kinetics'
WRITE(1,*) 'alpha A, alpha C'
WRITE(1,*) 'Kinetics Filename'
WRITE(1,*) ' t star, i star'
CLOSE(1)

RETURN
END
```

```
      PROGRAM ECBPrePro

      CHARACTER*80 FileName
      INTEGER Length,I,Lgth
      LOGICAL User

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMac.INC'

      COMMON /WORKSP/  RWKSP
      REAL RWKSP(250000)
      CALL IWKIN(250000)

C This program read macro vertices and bc's and makes the *.MSH & *.SRC
C file

      CALL Titles
      CALL ChInp('Filename? (No extension)',FileName)
      CALL UpperCaseCh(FileName)
      CALL StringLength(FileName,Length)

      User=(FileName(1:Length).EQ.'USER')
      IF (.NOT.User) THEN
        IF ((FileName(Length:Length).EQ.'A').OR.
     & (FileName(Length:Length).EQ.'B')) THEN
          CALL ReadPar(FileName(1:Length-1)//'.PAR')
        ELSE
          CALL ReadPar(FileName(1:Length)//'.PAR')
        END IF
      ELSE
        BasisDeg=1
      END IF
      CALL ReadMac(FileName(1:Length)//'.MAC')

      IF (NoOfMacElems.GT.1) CALL LocateTrans
      CALL MakeNodeMap
      CALL WriteMac(FileName(1:Length)//'.MAC')

      CALL MakeMeshPts(.FALSE.,.FALSE.)
      CALL PrintMacroElems(FileName)

      IF (.NOT.User) THEN
        CALL WriteMesh(FileName(1:Length)//'.MSH')
        WRITE(*,*) FileName(1:Length),'.MSH has been created'

        CALL MakeSrcPts
        CALL WriteSrc(FileName(1:Length)//'.SRC')
        WRITE(*,*) FileName(1:Length),'.SRC has been created.'

      ELSE

        OPEN(UNIT=1,FILE='USER.ORIG',STATUS='NEW',FORM='FORMATTED')
        WRITE(1,*) NoOfCoords
        DO I=1,NoOfCoords-1
          WRITE(1,*) Coords(I,1),Coords(I,2)
        END DO
```

```
        WRITE(1,*) MacVertX(MacIndx(NoOfMacElems,2)),
     & MacVertY(MacIndx(NoOfMacElems,2))
        CLOSE(1)
        WRITE(*,*) 'USER.ORIG has been created.'

      END IF

      STOP 'ECBPrePro: Normal Completion'
      END

C ******************************************************************** CopyMactoTMac

      SUBROUTINE CopyMactoTMac(I,Offset)

      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBTMac.INC'
      INTEGER I,J,Offset

      TMacBCType(I+Offset)=MacBCType(I)
      TMacBCVal(I+Offset)=MacBCVal(I)
      TNoOfMicElems(I+Offset)=NoOfMicElems(I)
      DO J=1,2
        TMacIndx(I+Offset,J)=MacIndx(I,J)
      END DO
      DO J=1,3
        TMacPar(I+Offset,J)=MacPar(I,J)
      END DO
      TMacRatio(I+Offset)=MacRatio(I)
      TMacShape(I+Offset)=MacShape(I)
      TMacSpac(I+Offset)=MacSpac(I)

      RETURN
      END

C ******************************************************************** CopyTMactoMac

      SUBROUTINE CopyTMactoMac

      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBTMac.INC'
      INTEGER I,J

      DO I=1,NoOfMacElems
        MacBCTypc(I)=TMacBCTypc(I)
        MacBCVal(I)=TMacBCVal(I)
        NoOfMicElems(I)=TNoOfMicElems(I)
        DO J=1,2
          MacIndx(I,J)=TMacIndx(I,J)
        END DO
        DO J=1,3
          MacPar(I,J)=TMacPar(I,J)
        END DO
        MacRatio(I)=TMacRatio(I)
        MacShape(I)=TMacShape(I)
        MacSpac(I)=TMacSpac(I)

      END DO
```

```
      RETURN
      END

C ************************************************************** CreateMacTrans

      SUBROUTINE CreateMacTrans(I,IPlus1,Offset)

      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBTMac.INC'
      INTEGER I,J,Offset,IPlus1

      TMacBCType(I+Offset)=MacBCType(I)
      TNoOfMicElems(I+Offset)=1

      TMacRatio(I+Offset)=1.0
      TMacShape(I+Offset)='TRNS'
      TMacSpac(I+Offset)='EQUL'
      DO J=1,3
        TMacPar(I+Offset,J)=0
      END DO

      MacVertX(NoOfMacVert+1)=
     & 0.95*(MacVertX(MacIndx(IPlus1,1)) - MacVertX(MacIndx(IPlus1,2)))
     & + MacVertX(MacIndx(IPlus1,2))
      MacVertY(NoOfMacVert+1)=
     & 0.95*(MacVertY(MacIndx(IPlus1,1)) - MacVertY(MacIndx(IPlus1,2)))
     & + MacVertY(MacIndx(IPlus1,2))

      MacVertX(TMacIndx(I+Offset-1,2))=
     & 0.95*(MacVertX(TMacIndx(I+Offset-1,2)) -
     & MacVertX(TMacIndx(I+Offset-1,1)))
     & + MacVertX(TMacIndx(I+Offset-1,1))
      MacVertY(TMacIndx(I+Offset-1,2))=
     & 0.95*(MacVertY(TMacIndx(I+Offset-1,2)) -
     & MacVertY(TMacIndx(I+Offset-1,1)))
     & + MacVertY(TMacIndx(I+Offset-1,1))

      MacIndx(IPlus1,1)=NoOfMacVert+1
      TMacIndx(I+Offset,1)=TMacIndx(I+Offset-1,2)
      TMacIndx(I+Offset,2)=MacIndx(IPlus1,1)

      NoOfMacVert=NoOfMacVert+1

      RETURN
      END

C ******************************************************************** LocateTrans

      SUBROUTINE LocateTrans

C This subroutine locates 'trans'-elements. Those elements that must span
C between two regions with different boundary conditions, but the same BC
C type (e.g. NAT, 0, to NAT, 1)

      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBTMac.INC'

      INTEGER I,Offset,IPlus1
```

```
      LOGICAL MacNat
      EXTERNAL MacNat

      Offset=0
      DO I=1,NoOfMacElems
        CALL CopyMactoTMac(I,Offset)
        CALL IncrementI(I,IPlus1,1,NoOfMacElems)
        IF ( ( MacNat(MacBCType(I))
     & .EQ. MacNat(MacBCType(IPlus1)) ) .AND.
     & ( MacBCVal(I) .NE. MacBCVal(IPlus1) ) .AND.
     & ( (MacShape(IPlus1) .NE. 'TRNS').AND.
     & (MacShape(I).NE.'TRNS') )) THEN
          Offset=Offset+1
          CALL CreateMacTrans(I,IPlus1,Offset)
        END IF
      END DO
      NoOfMacElems=NoOfMacElems+Offset

      CALL CopyTMactoMac

      RETURN
      END

C ************************************************************** PrintMacroElems

      SUBROUTINE PrintMacroElems(FileName)

      INCLUDE 'ECBMac.INC'

      INTEGER LOutput/6/,I,Length,MSFLen
      CHARACTER*(*) FileName
      CHARACTER*20 FileN

      FileN=Filename
      CALL StringLength(FileN,Length)

      WRITE(LOutput,100)
100   FORMAT(X,80('*'))

C                    123456789a123456789b
      WRITE(LOutput,250) 'Macro Element Report',FileN
250   FORMAT(X,30X,A20,<30-Length>X,A<Length>)

      WRITE(LOutput,100)

      WRITE(LOutput,500)
500   FORMAT(X,/,X,18X,'- - - - M A C R O V E R T I C E S - - - -'/,
     & X,18X,'Point',10X,'X',18X,'Y',/)

C 123456789a123456
c   - - - - M A C R O V E R T I C E S - - - -
C 123     123456789a 123456789a12345678
c   Point        X                Y

510   FORMAT(X,19X,I3,2(5X,G14.5))
C 1234  12345            12345
c    iii    gggggggggggggg    gggggggggggggg
```

```
      DO I=1,NoOfMacVert
        WRITE(LOutput,510) I,MacVertX(I),MacVertY(I)
      END DO

      WRITE(LOutput,600)
600   FORMAT(X,/,/,X,15X,'Micro',6X,'Boundary',27X,'MicroNodes',/
     & X,'Element  Type Elements  Type   Value',10X,
     & 'MacroVertices  Start  End',/)
C 123456789a12345     123456        123456789a123456789b1234567
C              Micro      Boundary                        MicroNodes
C       12    1        12    123     123456789a1           12     12
C Element  Type Elements  Type   Value          MacroVertices  Start  End

610   FORMAT(X,2X,I3,4X,A4,3X,I3,5X,A4,3X,G14.5,X,2(X,I4),
     & 5X,I4,2X,I4)
611   FORMAT(X,9X,A)
615   FORMAT(X,9X,A4,:,3X,'Spacing Factor:',G14.5)
616   FORMAT(X,9X,A4,:,3X,'  Spacing File:',A)
C                     123456789a12345
C 12   1234    123   12345    123              12         12345    12
C  iii    aaaa   iii   xxaaaax  gggggggggggggg  iiii iiii     iiii  iiii
c                          (transition)
620   FORMAT(X,2X,I3,4X,A4,3X,I3,5X,A4,3X,' (transition) ',
     & X,2(X,I4),5X,I4,2X,I4)

      DO I=1,NoOfMacElems
        IF (MacShape(I).NE.'TRNS') THEN

          WRITE(LOutput,610) I,MacShape(I),NoOfMicElems(I),
     & MacBCType(I),MacBCVal(I),MacIndx(I,1),MacIndx(I,2),
     & MacStart(I),MacEnd(I)

          IF (MacShape(I).EQ.'USER') THEN
            CALL StringLength(MacBdryFile,MSFLen)
            WRITE(LOutput,611) MacBdryFile(1:MSFLen)
          END IF

          IF (MacSpac(I).EQ.'USER') THEN
            CALL StringLength(MacSpaFile(I),MSFLen)
            WRITE(LOutput,616) MacSpac(I),MacSpaFile(I)(1:MSFLen)
          ELSE IF (MacSpac(I).EQ.'LOG') THEN
            WRITE(LOutput,615) MacSpac(I),MacRatio(I)
          ELSE
            WRITE(LOutput,615) MacSpac(I)
          END IF

        ELSE

          WRITE(LOutput,620) I,MacShape(I),NoOfMicElems(I),
     & MacBCType(I),MacIndx(I,1),MacIndx(I,2),
     & MacStart(I),MacEnd(I)

        END IF

        IF ((MacIndx(I,1).GT.NoOfMacVert).OR.
     & (MacIndx(I,2).GT.NoOfMacVert)) THEN
          WRITE(*,*) 'Error: MacIndx is out of range.  Mac Elem=',I
          WRITE(*,*) '       MacIndx=',MacIndx(I,1),MacIndx(I,2)
```

```
        STOP 'In Routine: PrintMacroElems'
      END IF
    END DO

    RETURN
    END

C ************************************************************************ Titles

    SUBROUTINE Titles

    WRITE(*,*) '   PROGRAM: ECBPrePro                    2/23/88'
    WRITE(*,*) 'Written by: Ken Jordan                Version 2'
    WRITE(*,*) ' '
    WRITE(*,*) 'This program reads input from the data file'
    WRITE(*,*) '*.MAC and generates output files *.MSH & *.SRC'
    WRITE(*,*) ' '
    WRITE(*,*) 'These files are used as input for program ECB'
    WRITE(*,*) ' '
    WRITE(*,*) 'This program generates the BEM Mesh from the Macro'
    WRITE(*,*) 'mesh stored in *.MAC'
    WRITE(*,*) ' '

    RETURN
    END
```

```
      program ECBM

C This program will solve for the current distribution and move boundaries
C for various levelling problems.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBSrc.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECGLOG.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMOVE.INC'
      INCLUDE 'ECBTime.INC'

      CHARACTER*80 FileName
      INTEGER FilLen,I

C Declarations required by IMSL

      COMMON /WORKSP/ RWKSP
      REAL RWKSP(136973)
      CALL IWKIN(136973)

C ------------------------------------------------------------ Initialize Problem

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following CALLS obtain from the user the name of the problem he is
C solving and time stepping criteria.

      CALL ChInp('Filename?',FileName)
      WRITE(*,*) ' '

      CALL DPInp('  Start Time (in seconds)',Time)
      CALL DPInp('     End Time',EndTime)
      CALL DPInp('Desired dTime',DesdTime)
      WRITE(*,*) ' '

      CALL DPInp('Desired Maximum dX per time step (unitless)',DesdX)
      WRITE(*,*) ' '

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following CALLS read data from files describing the BEM problem to solve.

      CALL StringLength(FileName,FilLen)
      CALL ReadPar(Filename(1:FilLen)//'.PAR')
      IF (CorLev) THEN
        CALL ReadMesh(Filename(1:FilLen)//'A.Msh')
        CALL ReadMac(FileName(1:FilLen)//'B.MAC')
      ELSE
        CALL ReadMesh(Filename(1:FilLen)//'.Msh')
        CALL ReadMac(FileName(1:FilLen)//'.MAC')
      END IF

      IF (PerRev) THEN
```

```
C Save time step for use in periodically reversing the current

          DeltaTime=Endtime
        END IF

C Note, the following two calls can be replaced by IMSL routines.

        CALL GauLeg(NoOfGaussPts,GaussPts,GaussWts)
        NoOfGauLogPts=NoOfGaussPts
        CALL GauLog(NoOfGauLogPts,GauLogPts,GauLogWts)

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C Make dimensional quantities dimensionless

        Time=Time/tstar
        EndTime=EndTime/tstar
        DesdTime=DesdTime/tstar
        MaxTimeCounter=1.5*(EndTime-Time)/DesdTime
        LastTime=.FALSE.
        TimeCounter=1

        OPEN(UNIT=3,FILE=Filename(1:FilLen)//'.TD',STATUS='NEW',
     & FORM='FORMATTED')
        WRITE(3,*) 1000

        IF (PerRev) THEN

          IF (PCRMax) THEN
            CALL PerCurRevMax(FileName)
          ELSE
            CALL PerCurRev(FileName)
          END IF

        ELSE

          CALL TimeLoop(FileName)

        END IF

        CLOSE(3)

        STOP 'Normal End'
        END

C ******************************************************************** AdjustForPr

        SUBROUTINE AdjustForPr(St,En)

        INTEGER St,En

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMac.INC'
        INCLUDE 'ECBPar.INC'
        INCLUDE 'ECBBC.inc'
        INTEGER I

        DO I=MacStart(St),MacEnd(En)
D         WRITE(*,*) 'Old BC Value:',I,BCValues(I)
```

```
          BCValues(I)= - BCValues(I)*CRRatio
D         WRITE(*,*) '        New:',I,BCValues(I)
        END DO

        DO I=St,En
          MacBCVal(I)= - MacBCVal(I)*CRRatio
        END DO

        RETURN
        END

C ************************************************************************** Bem

        SUBROUTINE Bem

C This subroutine solves the BEM problem posed in the include files.

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBPar.INC'

        REAL*8 dBCValues(MaxNoOfCoords),MagDiff,MagBC,MagdBC
        INTEGER NoOfdBCValues,Iteration
        LOGICAL Convergence,SolvedOnce

        REAL*8 VecNormDP
        EXTERNAL VecNormDP

C Make set of source points

        CALL MakeSrcPts

C Assemble G & H Matrices, and re-arrange into A & C

        CALL AssembleGH
        CALL AssembleAC

        Iteration=0
        SolvedOnce=.FALSE.
        Convergence=.FALSE.

        DO WHILE(.NOT.Convergence)

C Solve the system BemG.SolValues= (BemH.BCValues) for SolValues

          CALL BemMatSol(SolvedOnce)

C Iterate to solve kinetic BCValues

          CALL NewtonRaphson(Iteration,
     & NoOfdBCValues,dBCValues,MagDiff,MagBC)
          IF (NoOfdBCValues.GT.0) THEN

C Test for convergence on kinetic boundary condition

            MagdBC=VecNormDP(NoOfdBCValues,dBCValues)
            Convergence=(MagdBC.LT.Tolerance)
            Convergence=Convergence.AND.(MagDiff.LT.Tolerance)
            IF (MagBC.NE.0) Convergence=Convergence.AND.
```

```
     & (MagdBC/MagBC.LT.Tolerance)

C Increment Number of Iterations, and print iteration status report

              Iteration=Iteration+1
              IF (MagBC.EQ.0) THEN
                WRITE(*,10) Iteration,MagDiff,MagdBC
              ELSE
                WRITE(*,10) Iteration,MagDiff,MagdBC,MagdBC/MagBC
              END IF
10            FORMAT(X,'Iter ',I3,2X,'|k-Ip|=',G13.6,2X,
     & '|dBC|=',G13.6,:,2X,'|dBC/BC|=',G13.6)
cIter iii |k-Ip|=gggggggggggggg |dBC|=gggggggggggggg |dBC/BC|=gggggggggggggg

C Check to see that we haven't iterated too many times.

              IF (Iteration.GE.MaxIter)
     & CALL ErrMaxIter(NoOfdBCValues,dBCValues)

              CALL UpdateBCValues(NoOfdBCValues,dBCValues)

            ELSE
              Convergence=.TRUE.
            END IF

          END DO

C If we have been iterating on a kinetic bc, re-solve the BEM equations
C before returning.

          IF (NoOfdBCValues.GT.0) CALL BemMatSol(SolvedOnce)

          RETURN
          END

C ******************************************************************** BemMatSol

          SUBROUTINE BemMatSol(SolvedOnce)

          LOGICAL SolvedOnce

          INCLUDE 'ECBHdr.INC'
          INCLUDE 'ECBMsh.INC'
          INCLUDE 'ECBCal.INC'
          INCLUDE 'ECBBC.INC'

          REAL*8 VecB(MaxNoOfCoords)

          CALL MatVecMultDP(MaxNoOfCoords,NoOfCoords,BemH,
     & BCValues,VecB)
          IF (SolvedOnce) THEN
            CALL SolveRelEq(VecB,SolValues)
          ELSE
            CALL SolveSimEq(BemG,VecB,NoOfCoords,SolValues,
     & MaxNoOfCoords)
            SolvedOnce=.TRUE.
          END IF
```

```
      RETURN
      END

C ******************************************************************* CopyInsPoints

      SUBROUTINE CopyInsPoints(Ntc,NodeMap,Elem,Coords)

C Passed Variables

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER Ntc,Elem,
     & NodeMap(MaxNoOfCoords,MaxBasisDeg+1)
      REAL*8 Coords(MaxNoOfCoords,MaxNoOfDim)

C Local Variables

      INTEGER Loc

      DO Loc=BasisDeg+1,2,-1
        CALL CopyPoint(NodeMap(Elem,Loc),Coords,NoOfIns,Ins)

        Ntc=Ntc+1
        IF (Ntc.EQ.1) First=NoOfIns
        IF (Ntc.EQ.2) Second=NoOfIns

        NextToLast=NoOfIns

      END DO

      RETURN
      END

C ******************************************************************* CurReversal

      SUBROUTINE CurReversal

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBTime.INC'

C The following CALLs adjust the current bc's for Current Reversal

      CALL AdjustForPr(3,5)
      CALL AdjustForPr(1,1)

      WagLin=WagLin*WaLinPCR
      DeltaTime=DeltaTime/CRRatio
      DesdTime=DesdTime/CRRatio

      EndTime=Time+DeltaTime/tstar
      MaxTimeCounter=1.5*(EndTime-Time)/DesdTime+TimeCounter
      LastTime=.FALSE.

C The following lines place the current reversal message into the output file.

      WRITE(*,*) ' '
```

```
      WRITE(*,*) 'Reversing Current! ',
   & 'Current Ratio Final/Initial=',CRRatio
      WRITE(*,*) '                    ',
   & '              Wa(lin) PCR =',WaLinPCR
      WRITE(*,*) '     Delta Time=',DeltaTime
      WRITE(*,*) ' '

      RETURN
      END

C ******************************************************************** ErrMaxIter

      SUBROUTINE ErrMaxIter(NoOfdBCValues,dBCValues)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBBC.INC'

      INTEGER J,I,NoOfdBCValues
      REAL*8 dBCValues(*),Factor,VecNormDP,MagdBC
      LOGICAL KineticBC
      EXTERNAL KineticBC,VecNormDP

      MagdBC=VecNormDP(NoOfdBCValues,dBCValues)
      IF (MagdBC.GT.1.0) THEN
        Factor=MAX(MIN(0.9D0,2.0D0/MagdBC),0.2D0)
      ELSE
        Factor=1.0
      END IF

      WRITE(*,*) 'Error: Maximum number of iterations reached'
      WRITE(*,*) 'ECBM: (In Routine: Bem) Abnormal End'

      WRITE(*,5)
5     FORMAT(/,X,'     dBC Value     BC Value      Sol Value',/)
10    FORMAT(X,I3,3X,G11.4,3X,G11.4,3X,G11.4)

C     dBC Value     BC Value      Sol Value
C iii ggggggggggg   ggggggggggg   ggggggggggg
C     md.ddddemdd
c     12345678901

      J=1
      DO I=1,NoOfCoords
        IF (KineticBC(BCType(I))) THEN
          WRITE(*,10) I,dBCValues(J)*Factor,
   & BCValues(I)-dBCValues(J)*FActor,SolValues(I)
          J=J+1
        END IF
      END DO

      STOP 'In Routine: Bem'
      END

C ******************************************************************** IntegrateBnddX

      REAL*8 FUNCTION IntegrateBnddX(Dummy)
```

```
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMove.INC'

      INTEGER NoOfX,I
      LOGICAL Dummy
      REAL*8 X(MaxNoOfCoords),Y(MaxNoOfCoords),
     & Break(MaxNoOfCoords),CSCoef(4,MaxNoOfCoords)
      REAL*8 LowerLim,UpperLim,DCSITG
      EXTERNAL DCSITG

      IF (.NOT.Dummy) THEN

        NoOfX=NoOfMicElems(1)*BasisDeg+1

        DO I=1,NoOfX
          X(I)=Coords(I,1)
          Y(I)=Coords(I,2)
D         WRITE(*,*) I,' X=',X(I),' Y=',Y(I)
        END DO

        LowerLim=Coords(1,1)
        UpperLim=Coords(NoOfX,1)

      ELSE

        NoOfX=NoOfMoved
        DO I=1,NoOfX
          X(I)=Moved(I,1)
          Y(I)=Moved(I,2)
D         WRITE(*,*) I,' X=',X(I),' Y=',Y(I)
        END DO

        LowerLim=Moved(1,1)
        UpperLim=Moved(NoOfX,1)

      END IF

C Compute Integral

      CALL DCSAKM(NoOfX,X,Y,Break,CSCoef)

D     WRITE(*,*) 'Lower Limit=',LowerLim
D     WRITE(*,*) 'Upper Limit=',UpperLim

      IntegrateBnddX=DCSITG(LowerLim,UpperLim,NoOfX-1,Break,CSCoef)

      RETURN
      END

C ********************************************************************** MakeIns

      SUBROUTINE MakeIns(ElemStart,ElemEnd,NodeMap,Coords,NoOfElems,
     & FileName)

C Passed Variables
```

```
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER Coord,ElemStart,ElemEnd,
     & NodeMap(MaxNoOfCoords,MaxBasisDeg+1),NoOfElems
      REAL*8 Coords(MaxNoOfCoords,MaxNoOfDim)
      CHARACTER*(*) FileName

C Local Variables

      INTEGER Elem,Loc,Ntc
      LOGICAL FirstExec/.TRUE./,InsExt/.FALSE./
      REAL*8 BegPtX,BegPtY,EndPtX,EndPtY

      COMMON /MakeInsVar/InsExt,FirstExec

      IF (FirstExec) CALL ReadInsExt(BegPtX,BegPtY,EndPtX,EndPtY,
     & InsExt,FileName)

      IF (InsExt) THEN
        NoOfIns=1
        Ins(1,1)=BegPtX
        Ins(1,2)=BegPtY
      ELSE
        NoOfIns=0
      END IF

C Forces the 1st point on the boundary to be an endpoint for an element

      IF ((ElemEnd+1).LE.(ElemStart-1)) THEN

        Ntc=0
        DO Elem=ElemStart-1,ElemEnd+1,-1
          CALL CopyInsPoints(Ntc,NodeMap,Elem,Coords)
        END DO

      ELSE

        Ntc=0
        DO Elem=ElemStart-1,1,-1
          CALL CopyInsPoints(Ntc,NodeMap,Elem,Coords)
        END DO

        DO Elem=NoOfElems,ElemEnd+1,-1
          CALL CopyInsPoints(Ntc,NodeMap,Elem,Coords)
        END DO

      END IF

      CALL CopyPoint(NodeMap(ElemEnd+1,1),Coords,NoOfIns,Ins)
      Last=NoOfIns

      IF (InsExt) THEN

        NoOfIns=NoOfIns+1

        Ins(NoOfIns,1)=EndPtX
        Ins(NoOfIns,2)=EndPtY
```

```
      END IF

      IF (dXdN(1).GT.0) THEN
        IF (InsExt) THEN
          StaRev=.TRUE.
          Second=1
        ELSE
          WRITE(*,*) 'Error: Moving boundary off of insulator!'
          WRITE(*,*) '       This error could be corrected by creating'
          WRITE(*,*) '       a *.EXT file.'
          STOP 'In Routine: MakeIns'
        END IF
      ELSE
        StaRev=.FALSE.
      END IF

      IF (dXdN(NoOfMoved).GT.0) THEN
        IF (InsExt) THEN
          EndRev=.TRUE.
          NextToLast=NoOfIns
        ELSE
          WRITE(*,*) 'Error: Moving boundary off of insulator!'
          STOP 'In Routine: MakeIns'
        END IF
      ELSE
        EndRev=.FALSE.
      END IF

      CALL AcuteAngles

      FirstExec=.FALSE.

      RETURN
      END

C ********************************************************************** MakedXdN

      SUBROUTINE MakedXdN(ElemStart,ElemEnd)

C Passed Variables

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBBC2.INC'
      INCLUDE 'ECBPAR.INC'
      INCLUDE 'ECBMOVE.INC'
      INCLUDE 'ECBTIME.INC'

      INTEGER ElemStart,ElemEnd

C Internal Variables

      INTEGER NoOfSmooth,Offset,Elem,Loc,I,Coord,LastC

      LOGICAL FirstTime/.TRUE./

      REAL*8 Im(MaxNoOfCoords),Ia(MaxNoOfCoords)
```

```
      REAL*8 SmoothX(MaxNoOfCoords),SmoothF(MaxNoOfCoords)
      REAL*8 OrigX(MaxNoOfCoords),LastOrig
      REAL*8 Breaka(MaxNoOfCoords),CSCoefa(4,MaxNoOfCoords)
      REAL*8 Breakm(MaxNoOfCoords),CSCoefm(4,MaxNoOfCoords)
      REAL*8 Ratio,Factor

C External Functions

      REAL*8 DCSVAL,SlopeVal,DCSITG
      EXTERNAL DCSVAL,SlopeVal,DCSITG

C The following DO-loop computes the local metal and corrosive agent flux.

      NoOfdXdN=0

      LastOrig=0.0
      LastC=NodeMap(ElemStart,1)

      DO Elem=ElemStart,ElemEnd
        DO Loc=1,BasisDeg+1
          Coord=NodeMap(Elem,Loc)
          NoOfdXdN=NoOfdXdN+1

          Im(NoOfdXdN)=
     & SlopeVal(Natural(Coord),BCValues(Coord),SolValues(Coord))

          IF (CorLev) THEN

C Compute Flux of Additive at point and equivalent corrosion current density
C from additive.

            Ia(NoOfdXdN)=
     & SlopeVal(Natural2(Coord),BCValues2(Coord),SolValues2(Coord))

          ELSE
            Ia(NoOfdXdN)=0
          END IF

C Calculate distances used to smooth c.d.

          OrigX(NoOfdXdN)=SQRT(
     & (Coords(Coord,1)-Coords(LastC,1))
     &  *(Coords(Coord,1)-Coords(LastC,1))
     & +(Coords(Coord,2)-Coords(LastC,1))
     &  *(Coords(Coord,2)-Coords(LastC,1)) )+LastOrig
          SmoothX(NoOfdXdN)=OrigX(NoOfdXdN)

          LastC=Coord
          LastOrig=OrigX(NoOfdXdN)

        END DO
      END DO

C Smooth dXdN *) Remove Duplicate X Values.

      NoOfSmooth=NoOfdXdN
      Offset=0
      DO I=1,NoOfSmooth-1
```

```
          IF (SmoothX(I-Offset).EQ.SmoothX(I+1)) THEN
            Offset=Offset+1
          END IF
          SmoothX(I-Offset+1)=SmoothX(I+1)
          IF (CorLev) Ia(I-Offset+1)=Ia(I+1)
          Im(I-Offset+1)=Im(I+1)
        END DO
        NoOfSmooth=NoOfSmooth-Offset

C Smooth dXdN *) Call IMSL Smoothing routine.

        IF (CorLev) CALL DCSSCV(NoOfSmooth,SmoothX,Ia,2,Breaka,CSCoefa)
        CALL DCSSCV(NoOfSmooth,SmoothX,Im,2,Breakm,CSCoefm)

        IF (CorLev) THEN

C Ratio= additive / current

          Ratio=DCSITG(SmoothX(1),SmoothX(NoOfSmooth),
     & NoOfSmooth-1,Breaka,CSCoefa)
     & / DCSITG(SmoothX(1),SmoothX(NoOfSmooth),
     & NoOfSmooth-1,Breakm,CSCoefm)

C First time through the loop, we must compute Factor = Corros <i> / <Na>

          IF (FirstTime) Factor=Corros/Ratio
          WRITE(*,*) '     Corros=',Ratio*Factor

        ELSE IF (PerRev) THEN

          WRITE(*,*) '     Im=',DCSITG(SmoothX(1),SmoothX(NoOfSmooth),
     & NoOfSmooth-1,Breakm,CSCoefm),'  Time=',Time*tstar

        END IF

        NoOfdXdN=0
        DO Elem=ElemStart,ElemEnd
          DO Loc=1,BasisDeg+1
            NoOfdXdN=NoOfdXdN+1

            Im(NoOfdXdN)=DCSVAL(OrigX(NoOfdXdN),NoOfSmooth-1,
     & Breakm,CSCoefm)
            IF (CorLev) THEN
              Ia(NoOfdXdN)=DCSVAL(OrigX(NoOfdXdN),NoOfSmooth-1,
     & Breaka,CSCoefa)
            ELSE
              Ia(NoOfdXdN)=0.0
            END IF

            dXdN(NoOfdXdN)=Im(NoOfdXdN)-Factor*Ia(NoOfdXdN)

D           WRITE(*,*) 'NoOfdXdN ',NoOfdXdN,'  X',SmoothX(NoOfDxDn)
          END DO
        END DO

        FirstTime=.FALSE.

        RETURN
```

```
      END

C ******************************************************************** PerCurRev

      SUBROUTINE PerCurRev(FileName)

      CHARACTER*80 FileName
      INTEGER PCRCounter

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMac.INC'

      IF (NoOfMacElems.LT.6) THEN
        WRITE(*,*) 'Error: Improper configuration for ',
     & 'solution of periodic'
        WRITE(*,*) '       current reversal problem.'
        WRITE(*,*) '       Use 4 sides w/ 2 transition elements'
        STOP 'ECBM: In Subroutine: PerCurRev'
      END IF

      IF (MacBCVal(4).GE.0) THEN
        WRITE(*,*) 'Current Initially Cathodic=',
     & MacBCVal(4)*istar,'mA/cm2'
      ELSE
        WRITE(*,*) 'Current Initially Anodic  =',
     & MacBCVal(4)*istar,'mA/cm2'
      END IF

      DO PCRCounter=1,NoOfPCR

        CALL TimeLoop(FileName)

C The following call updates the boundary conditions

        CALL CurReversal

        CALL TimeLoop(FileName)

        IF (PCRCounter.LT.NoOfPCR) THEN

          CRRatio=1/CRRatio
          WaLinPCR=1/WaLinPCR

          CALL CurReversal

          CRRatio=1/CRRatio
          WaLinPCR=1/WaLinPCR

        END IF

      END DO

      RETURN
      END

C ******************************************************************** PerCurRevMax
```

```
      SUBROUTINE PerCurRevMax(FileName)

      CHARACTER*80 FileName
      INTEGER PCRCounter,FilLen

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMOVE.INC'
      INCLUDE 'ECBTime.INC'

      INTEGER ElemStart,ElemEnd,Elem,Loc,I
      LOGICAL BdryFileExist,Iterating
      REAL*8 IntOrig,IntCurr,IntegrateBnddX,dAdis,dAdep,dh,h
      REAL*8 FirstX,FirstY,LastX,LastY,BigdTime
      REAL*8 BigdMaxTC
      EXTERNAL IntegrateBnddX

      CALL StringLength(FileName,FilLen)

      IF (NoOfMacElems.LT.4) THEN
        WRITE(*,*) 'Error: Improper configuration for ',
     & 'solution of periodic'
        WRITE(*,*) '       current reversal problem with',
     & ' maximum levelling.'
        WRITE(*,*) '       Use 4 sides (ESS,NAT,ESS,NAT)'
        STOP 'ECBM: In Subroutine: PerCurRevMax'
      END IF

      IF (Time.GT.0) THEN
        WRITE(*,*) 'Warning: Start time reset to zero.'
      END IF
      Time=0.0
      BigdTime=EndTime-Time
      BigdMaxTC=MaxTimeCounter

      WRITE(*,*) 'Current Initially Anodic: Maximum Levelling Case.'

C Moving boundary is always the first macro element.

C (Note: NoOfMicElems(1) is the number of microelements on the moving part
C of the boundary. The program assumes that parts A & B have the same moving
C part of the boundary, with identical discretization.)

      ElemStart=1
      ElemEnd=NoOfMicElems(1)

      IntOrig=IntegrateBnddX(.FALSE.)

      DO PCRCounter=1,NoOfPCR

        CALL TimeLoop(FileName)

        IntCurr=IntegrateBnddX(.FALSE.)
        dAdis=ABS(IntOrig-IntCurr)
        IntOrig=IntCurr
```

```
      h=dAdis/(ABS(
     & MacVertX(MacIndx(1,1))
     & - MacVertX(MacIndx(1,2)) ))

      WRITE(*,*) 'Reversing Current! '

C ------------------------------------------------------------ Boundary Movement

C Store coordinates in mesh into temporary storage (Moved), then compute
C the normal to every point on the surface (Nx), and current density at those
C points (dXdN)

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following DO-LOOP stores the 'current density' at each point of the moving
C boundary; the result is stored in matrix dXdN.

      NoOfdXdN=0
      DO Elem=ElemStart,ElemEnd
        DO Loc=1,BasisDeg+1
          NoOfdXdN=NoOfdXdN+1
          dXdN(NoOfdXdN)=-1.0
        END DO
      END DO

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

C The following CALL computes the 'normal' at each point of the moving
C boundary; the result is stored in matrix Nx.

      CALL MakeMoved(ElemStart,ElemEnd,NodeMap,Coords)
      CALL ComputeNx(ElemEnd-ElemStart+1)

C The following DO-loop iterates on h until dAdep=dAdis - - - - - - - - - -

      dh=100*Tolerance
      Iterating=.TRUE.
      DO WHILE (Iterating)

C Moved;1
D         CALL WriteMoved

C Move the points.

          CALL Move(h)
C Moved;2
D         CALL WriteMoved

C Store coordinates not being moved in Ins

          CALL MakeIns(ElemStart,ElemEnd,NodeMap,Coords,NoOfElems,
     & Filename)

C Search for intersections, and recompute moved boundary w/o intersections

          CALL RemoveLocIntersects
C Moved;3
D         CALL WriteMoved
```

```
c           CALL RemoveBdryIntersects
cC Moved;4
cD          CALL WriteMoved

            CALL RemoveInsIntersects
C Moved;5
D           CALL WriteMoved

C Compute next change in h.

            IntCurr = IntegrateBnddX(.TRUE.)
            dAdep = ABS(IntOrig-IntCurr)
            dh = (h/dAdep)*(dAdis-dAdep)
            WRITE(*,100) h,dh,dAdep,dAdis
100         FORMAT(X,'h=',G14.7,'  dh=',G14.7,'  dAdep=',G14.7,
   & '  dAdis=',G14.7)
            h = h+dh
            Iterating=(dh/h.GT.Tolerance)

          END DO
          IntOrig=IntCurr

          CALL WriteMBF(MacBdryFile)

          FirstX=Moved(1,1)
          FirstY=Moved(1,2)
          LastX=Moved(NoOfMoved,1)
          LastY=Moved(NoOfMoved,2)

          CALL ReMakeMesh(FileName(1:FilLen),FirstX,FirstY,
   & LastX,LastY,.FALSE.)

          CALL WriteBND(1,FILENAME(1:FilLen)//'.BND',
   & ElemStart,ElemEnd,NodeMap,Coords,Time*tstar)

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following IF-THEN block resets time constants for the
C next anodic dissolution step.

          IF (PCRCounter.LT.NoOfPCR) THEN
            WRITE(*,*) 'Reversing Current! '
            EndTime=Time+BigdTime
            LastTime=.FALSE.
            MaxTimeCounter=TimeCounter+BigdMaxTC
            CALL MakeMoved(ElemStart,ElemEnd,NodeMap,Coords)
          END IF

        END DO
        CALL WriteMesh(FileName(1:FilLen)//'.MSH')

        RETURN
        END

C ******************************************************************** ReMakeMesh

        SUBROUTINE ReMakeMesh(FileName,X1,Y1,X2,Y2,Smooth)

C X1,Y1, and X2,Y2 are the starting and ending points for the moved element
```

```
C of the macro mesh.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBPar.INC'

      CHARACTER*(*) FileName
      REAL*8 X1,Y1,X2,Y2
      LOGICAL SaveKinetic/.TRUE./,Smooth

C Alter macro info

      IF (CorLev) CALL ReadMac(FileName//'.MAC')

      MacVertX(1)=X1
      MacVertY(1)=Y1

      MacVertX(2)=X2
      MacVertY(2)=Y2

      MacShape(1)='USR'
      CALL MakeNodeMap
      CALL MakeMeshPts(SaveKinetic,Smooth)

      RETURN
      END

C ******************************************************************** ReadInsExt

      SUBROUTINE ReadInsExt(BegPtX,BegPtY,EndPtX,EndPtY,InsExt,FileName)

      LOGICAL InsExt
      REAL*8 BegPtX,BegPtY,EndPtX,EndPtY
      CHARACTER*(*) FileName
      INTEGER FilLen

      CALL StringLength(FileName,FilLen)
      INQUIRE(FILE=FileName(1:FilLen)//'.EXT',EXIST=InsExt)
      IF (InsExt) THEN

        OPEN(UNIT=1,FILE=FileName(1:FilLen)//'.EXT',
     & STATUS='OLD',FORM='FORMATTED')
        READ(1,*) BegPtX,BegPtY
        READ(1,*) EndPtX,EndPtY
        CLOSE(1)

        WRITE(*,10)
        WRITE(*,20) Filename(1:FilLen)//'.EXT'
        WRITE(*,30) BegPtX,BegPtY
        WRITE(*,40) EndPtX,EndPtY
        WRITE(*,50)

      END IF
      RETURN

C The following comments document the extended insulator report.

C 123456789a123456789b123456789c13456789d123456789e123456789
```

```
C ------------------------------------------------------Insulator Extension
C 123456789a123
C           >> Insulator extended beyond solution domain  <<
C 123456789a123456789b1234        123456789a
C                     Start: ( d.ddde+dd, d.ddde+dd)
C                    Finish: ( d.ddde+dd, d.ddde+dd)
C 123456789a123456789b123456789c123456789d123456789e123456789f123456789g12345678
C ---------------------------------------------------------------------------

10      FORMAT(X,59('*'),' Insulator Extension')
20      FORMAT(X,13X,'>>  Insulator Extended Beyond Solution Domain: ',
   & A,'  <<')
30      FORMAT(X,24X,'Start: (',G10.3,',',G10.3,')')
40      FORMAT(X,23X,'Finish: (',G10.3,',',G10.3,')')
50      FORMAT(X,78('*'))

        END

C ******************************************************************** SaveECBbc

        SUBROUTINE SaveECBbc

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMsh.INC'
        INCLUDE 'ECBbc.INC'
        INCLUDE 'ECBbc2.INC'

        INTEGER I

        DO I=1,NoOfCoords
          BCValues2(I)=BCValues(I)
          SolValues2(I)=SolValues(I)
          Natural2(I)=Natural(I)
        END DO

        RETURN
        END

C ******************************************************************** SlopeVal

        REAL*8 FUNCTION SlopeVal(Nat,BCV,SolV)
        LOGICAL Nat
        REAL*8 BCV,SolV

        IF (Nat) THEN
          SlopeVal=BCV
        ELSE
          SlopeVal=SolV
        END IF

        RETURN
        END

C ******************************************************************** TimeLoop

        SUBROUTINE TimeLoop(FileName)

C The subroutine performs the Time Iteration Loop
```

```
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBSrc.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECGLOG.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMOVE.INC'
      INCLUDE 'ECBTIME.INC'

      CHARACTER*(*) FileName

      INTEGER ElemStart,ElemEnd,Elem,Loc,I,FilLen
      LOGICAL BdryFileExist
      REAL*8 FirstX,FirstY,LastX,LastY
      REAL*8 MaxdXdN,MindXdN

      CALL StringLength(FileName,FilLen)

      DO WHILE(.NOT.LastTime)

        IF (CorLev) THEN

C ------------------------------------- Compute the concentration distribution
C Set up for DifEq A

D           WRITE(*,*) 'Solving Differential Equation A--Concentration'
            CALL Bem

C Save what we need from Dif Eq A for Dif Eq B

            CALL SaveECBbc
          END IF

C ------------------------------------------- Compute the current distribution
C Set up for Dif Eq B

C Read the macro file for part B of the problem (the part that computes the
C current distribution)

          IF (CorLev) THEN

            CALL ReadMac(FileName(1:FilLen)//'B.MAC')

C Read the mesh file for part B.

            CALL ReadMesh(Filename(1:FilLen)//'B.MSH')
D           WRITE(*,*) 'Solving Differential Equation B--Potential'

          ELSE

D           WRITE(*,*) 'Solving Differential Equation----Potential'

          END IF

C Solve current distribution problem
```

```
      CALL Bem

C ------------------------------------------------------------ Boundary Movement

C Moving boundary is always the first macro element.

C (Note: NoOfMicElems(1) is the number of microelements on the moving part
C of the boundary. The program assumes that parts A & B have the same moving
C part of the boundary, with identical discretization.)

      ElemStart=1
      ElemEnd=NoOfMicElems(1)

C Store coordinates in mesh into temporary storage (Moved), then compute
C the normal to every point on the surface (Nx), and current density at those
C points (dXdN)

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following CALL stores the moving boundary into a temporary location,
C array Moved.

      CALL MakeMoved(ElemStart,ElemEnd,NodeMap,Coords)
C Moved;1
D     CALL WriteMoved

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following CALL computes the 'current density' at each point of the moving
C boundary; the result is stored in matrix dXdN.

      CALL MakedXdN(ElemStart,ElemEnd)

C The maximum motion of the boundary determines the maximum allowable time
C step. For the time step, we choose the minimum of desired time step, and
C the largest time step allowed by the maximum current density on the boundary.

      CALL VecExtremaDP(dXdN,NoOfMoved,MaxdXdN,MindXdN)
      MaxdXdN=MAX(ABS(MaxdXdN),ABS(MindXdN))

      MotdTime=DesdX/MaxdXdN
      dTime=MIN(DesdTime,MotdTime)

      IF (dTime.LE.0) THEN
        WRITE(*,*) 'Error: dTime < 0.0!'
        STOP 'In Program :ECBM'
      END IF

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following IF-THEN-ELSE block determines if this is the last time we
C run the Time-Iteration Loop

      IF (Time+dTime.GT.EndTime) THEN
        dTime=EndTime-Time
        LastTime=.TRUE.
      ELSE
        LastTime=.FALSE.
      END IF

      IF (TimeCounter.GT.MaxTimeCounter) LastTime=.TRUE.
```

```
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following CALL computes the 'normal' at each point of the moving
C boundary; the result is stored in matrix Nx.

          CALL ComputeNx(ElemEnd-ElemStart+1)

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C Move the points.

          CALL Move(dTime)
C Moved;2
D         CALL WriteMoved

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C Store coordinates not being moved in Ins

          CALL MakeIns(ElemStart,ElemEnd,NodeMap,Coords,NoOfElems,
     & Filename)

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C Search for intersections, and recompute moved boundary w/o intersections

          CALL RemoveLocIntersects
C Moved;3
D         CALL WriteMoved

c         CALL RemoveBdryIntersects
cC Moved;4
cD        CALL WriteMoved

          CALL RemoveInsIntersects
C Moved;5
D         CALL WriteMoved

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C Save moved nodes for generating new mesh

          INQUIRE(FILE=MacBdryFile,EXIST=BdryFileExist)
          IF (BdryFileExist) THEN
            OPEN(UNIT=1,FILE=MacBdryFile,STATUS='OLD',FORM='FORMATTED')
          ELSE
            OPEN(UNIT=1,FILE=MacBdryFile,STATUS='NEW',FORM='FORMATTED')
            WRITE(*,*) 'Warning: Moving boundary must have shape type USER'
            STOP 'ECBM: Abnormal End'
          END IF

          WRITE(1,*) NoOfMoved
          DO I=1,NoOfMoved
            WRITE(1,*) Moved(I,1),Moved(I,2)
          END DO
          CLOSE(1)

          FirstX=Moved(1,1)
          FirstY=Moved(1,2)
          LastX=Moved(NoOfMoved,1)
          LastY=Moved(NoOfMoved,2)
```

```
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C Next, use the current distribution to compute the boundary movement

          Time=Time+dTime
          TimeCounter=Timecounter+1

          IF (.NOT.PerRev) THEN
            WRITE(*,20) TimeCounter,Time*tstar
20          FORMAT(X,'Boundary ;',I4,' corresponds to time=',G14.7,' secs')
          END IF

          WRITE(3,*) Time*tstar,Moved(NoOfMoved,2)-Moved(1,2)

C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C The following calls recompute the mesh based on the new boundary

          IF (CorLev) THEN
            CALL ReMakeMesh(FileName(1:FilLen)//'B',FirstX,FirstY,
     & LastX,LastY,.TRUE.)
            CALL WriteMesh(FileName(1:FilLen)//'B.MSH')

            CALL ReMakeMesh(FileName(1:FilLen)//'A',FirstX,FirstY,
     & LastX,LastY,.TRUE.)
            CALL WriteMesh(FileName(1:FilLen)//'A.MSH')
          ELSE
            CALL ReMakeMesh(FileName(1:FilLen),FirstX,FirstY,
     & LastX,LastY,.TRUE.)
            CALL WriteMesh(FileName(1:FilLen)//'.MSH')
          END IF

          IF (.NOT.PerRev) THEN
            CALL WriteBND(1,FILENAME(1:FilLen)//'.BND',
     & ElemStart,ElemEnd,NodeMap,Coords,Time*tstar)
          END IF

        END DO

        IF (PerRev) THEN
          CALL WriteBND(1,FILENAME(1:FilLen)//'.BND',
     & ElemStart,ElemEnd,NodeMap,Coords,Time*tstar)
          WRITE(*,*) FILENAME(1:FilLen)//'.BND',' written for ',Time*tstar
        END IF

        RETURN
        END

C ******************************************************************** WriteArray

        SUBROUTINE WriteArray(NoOfElem,Array)

        REAL*8 Array(*)
        INTEGER NoOfElem,I

        OPEN(UNIT=1,FILE='ARRAY.DAT',STATUS='NEW',FORM='FORMATTED')
        WRITE(1,*) NoOfElem

        DO I=1,NoOfElem
          WRITE(1,*) I,Array(I)
```

```
      END DO

      CLOSE(1)

      RETURN
      END

C ******************************************************************** WriteBND

      SUBROUTINE WriteBND(LunO,Name,ElemStart,ElemEnd,NodeMap,Coords,
     & Time)

      INCLUDE 'ECBHdr.INC'
      REAL*8 Coords(MaxNoOfCoords,MaxNoOfDim),Time
      INTEGER NoOfOrig,LunO,I,K,NodeMap(MaxNoOfCoords,MaxBasisDeg+1)
      INTEGER Elem,Loc,ElemStart,ElemEnd
      CHARACTER*(*) Name

      IF (.NOT.((LunO.EQ.6).OR.(LunO.EQ.5))) THEN
        OPEN(UNIT=LunO,FILE=Name,FORM='FORMATTED',STATUS='NEW')
      END IF

      WRITE(LunO,*) (ElemEnd-ElemStart+1)*BasisDeg+1
      DO Elem=ElemStart,ElemEnd
        DO Loc=1,BasisDeg
          WRITE(LunO,*) (Coords(NodeMap(Elem,Loc),K),K=1,NoOfDim)
        END DO
      END DO
      WRITE(LunO,*) (Coords(NodeMap(ElemEnd,BasisDeg+1),K),K=1,NoOfDim)
      WRITE(LunO,*) Time
      IF (.NOT.((LunO.EQ.6).OR.(LunO.EQ.5))) THEN
        CLOSE(LunO)
      END IF

      RETURN
      END

C ******************************************************************** WriteMBF

      SUBROUTINE WriteMBF(MacBdryFile)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMove.INC'

      CHARACTER*(*) MacBdryFile
      INTEGER I
      LOGICAL BdryFileExist

C Save moved nodes for generating new mesh

      INQUIRE(FILE=MacBdryFile,EXIST=BdryFileExist)
      IF (BdryFileExist) THEN
        OPEN(UNIT=1,FILE=MacBdryFile,STATUS='OLD',FORM='FORMATTED')
      ELSE
        OPEN(UNIT=1,FILE=MacBdryFile,STATUS='NEW',FORM='FORMATTED')
        WRITE(*,*) 'Warning: Moving boundary must have shape type USER'
        STOP 'ECBM: (Subroutine WriteMBF) Abnormal End'
      END IF
```

```
      WRITE(1,*) NoOfMoved
      DO I=1,NoOfMoved
        WRITE(1,*) Moved(I,1),Moved(I,2)
      END DO
      CLOSE(1)

      RETURN
      END

C ******************************************************************** WriteMoved

      SUBROUTINE WriteMoved

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER I

      OPEN(UNIT=1,FILE='MOVED.DAT',STATUS='NEW',FORM='FORMATTED')

      WRITE(1,*) NoOfMoved
      DO I=1,NoOfMoved
        WRITE(1,*) Moved(I,1),Moved(I,2)
      END DO

      CLOSE(1)
      RETURN
      END

c *
```

```
      PROGRAM ECBINT

C This program will be used to compute potential at internal points,
C after a current distribution problem has been solved.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBSrc.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECGLOG.INC'

      INTEGER Length
      CHARACTER*80 File/'ECB'/

      CALL Titles
      CALL EditCh('Filename? (No extension)',File)
      CALL StringLength(File,Length)

      CALL ReadPar(FILE(1:Length)//'.PAR')
      CALL ReadMesh(FILE(1:Length)//'.Msh')

      CALL GauLeg(NoOfGaussPts,GaussPts,GaussWts)
      NoOfGauLogPts=NoOfGaussPts
      CALL GauLog(NoOfGauLogPts,GauLogPts,GauLogWts)

      CALL ReadInt(File(1:Length)//'.INT')
      CALL ComputeInt
      CALL WriteInt(File(1:Length)//'.INT')

      STOP 'ECB: Normal Completion'
      END

C ********************************************************************** Titles

      SUBROUTINE Titles

      WRITE(*,*) '   PROGRAM: ECBINT                    10/20/89'
      WRITE(*,*) 'Written by: Ken Jordan                Version 1'
      WRITE(*,*) ' '
      WRITE(*,*) 'This program reads input from the data files'
      WRITE(*,*) '*.PAR, *.MSH, and *.INT; generates output files'
      WRITE(*,*) '*.INT'
      WRITE(*,*) ' '
      WRITE(*,*) 'This program solves for the potential at the internal'
      WRITE(*,*) 'points specified in the *.INT input files.'
      WRITE(*,*) ' '

      RETURN
      END
```

```
      PROGRAM ECBPlot

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBSrc.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBElm.INC'
      INCLUDE 'ECBINT.INC'
      INCLUDE '[KJ.DATA]PLOTLABL.INC'
      INCLUDE 'ECBPAR.INC'

      CHARACTER*80 FileName
      CHARACTER*1 SubProb
      INTEGER Start,End,FLen
      LOGICAL IntFile

      CALL GetFileName(FileName,FLen)

C Read and process the data for later use.

      CALL ReadPar(FileName(1:FLen)//'.PAR')
      IF (CorLev) THEN
        DO WHILE ((SubProb.NE.'A').AND.(SubProb.NE.'B'))
          CALL ChInp('Sub Problem A or B? (A or B)?',SubProb)
        END DO
        FileName(FLen+1:)=SubProb
        FLen=FLen+1
      END IF

      CALL ReadMac(FileName(1:FLen)//'.MAC')
      CALL ReadMesh(FileName(1:FLen)//'.MSH')
      CALL ReadSrc(FileName(1:FLen)//'.SRC')

      INQUIRE(FILE=FileName(1:FLen)//'.INT',EXIST=Intfile)
      IF (Intfile) THEN
        CALL ReadInt(FileName(1:FLen)//'.INT')
      END IF

C Put Source Points into a plottable file

      CALL PlotSrc

C Put Node Points into a plottable file

      CALL PlotNod

C Read, Process & Plot internal points

      IF (IntFile) CALL ProcessInt

C Process set and solved for values.

      CALL MeshReport

      CALL StartEnd(Start,End)

      IF (Start.GT.0) CALL ProcessSolSet(Start,End)
```

```
C Tell user where plotted data is.

      WRITE(*,*) ' '
      WRITE(*,*) ' ECBPLOT.NOD    Node Points (x,y)'
      WRITE(*,*) ' ECBPLOT.SRC    Source Points (x,y)'
      WRITE(*,*) ' '

      STOP 'ECBPlot: Normal End'
      END

C ******************************************************************** GetFileName

      SUBROUTINE GetFileName(FileName,FLen)

C Get filename.

      CHARACTER*(*) FileName
      LOGICAL FileExist
      INTEGER FLen

      FileExist=.FALSE.
      DO WHILE(.NOT.FileExist)
        CALL EditCh('Enter Problem Name?',FileName)
        CALL StringLength(FileName,FLen)
        IF (FLen.LE.0) STOP 'ECBPlot (GetFileName): No name Entered'
        INQUIRE(FILE=FileName(1:FLen)//'.PAR',EXIST=FileExist)
        IF (.NOT.FileExist) THEN
          WRITE(*,*) 'Error: File '//FileName(1:FLen)//'.PAR '
          WRITE(*,*) '       doesn''t exist.'
        END IF
      END DO

      RETURN
      END

C ******************************************************************** MeshReport

      SUBROUTINE MeshReport

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBMac.INC'

      WRITE(*,*) ' '
      WRITE(*,*) ' Preparing to Plot BC and Solution Values'
      WRITE(*,*) ' '

      WRITE(*,*) 'Minimum Element Number=',1
      WRITE(*,*) 'Maximum Element Number=',NoOfElems

      RETURN
      END

C ********************************************************************** PlotInt

      SUBROUTINE PlotInt(Pot,Con)

      INCLUDE 'ECBHDR.INc'
```

```
      INCLUDE 'ECBINT.INC'

      INTEGER J,K,Loc,NoOfPts,I,Con
      LOGICAL Between
      REAL PtsX(MaxNoOfCoords),PtsY(MaxNoOfCoords)
      REAL*8 XPt,YPt,Pot
      CHARACTER*3 Ext(10)/'CN1','CN2','CN3','CN4','CN5','CN6','CN7',
     & 'CN8','CN9','CNA'/

      WRITE(*,*) 'Plotting Internal Points ',Pot

      NoOfPts=0
      DO J=1,NoOfSpines
        DO K=1,IntPerSpine-1
          Loc=(J-1)*IntPerSpine+K

          Between= (( (InternalPot(Loc).LE.Pot) .AND.
     & (Pot.LE.InternalPot(Loc+1)) ) .OR.
     & ( (InternalPot(Loc).GE.Pot) .AND.
     & (Pot.GE.InternalPot(Loc+1)) ))
          IF (Between) THEN
            NoOfPts=NoOfPts+1
            CALL Interp1d(InternalPot(Loc),InternalPts(Loc,1),
     & InternalPot(Loc+1),InternalPts(Loc+1,1),
     & Pot,XPt)
            PtsX(NoOfPts)=XPt

            CALL Interp1d(InternalPot(Loc),InternalPts(Loc,2),
     & InternalPot(Loc+1),InternalPts(Loc+1,2),
     & Pot,YPt)
            PtsY(NoOfPts)=YPt
            WRITE(*,10) XPt,YPt
10          FORMAT(X,10X,G12.4,2X,G12.4)
          END IF
        END DO
      END DO

      IF (NoOfPts.GT.0) THEN
        OPEN(UNIT=1,FILE='ECBPLOT.'//EXT(Con),
     & STATUS='NEW',FORM='FORMATTED')
        WRITE(1,*) NoOfPts, Pot,',''Potential'''
        DO I=1,NoOfPts
          WRITE(1,*) PtsX(I),PtsY(I)
        END DO
        CLOSE(1)
      END IF

      RETURN
      END

C ********************************************************************** PlotNod

      SUBROUTINE PlotNod

      INTEGER I
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
```

```
      OPEN(UNIT=1,FILE='ECBPLOT.NOD',STATUS='NEW',FORM='FORMATTED')
      WRITE(1,*) NoOfCoords
      DO I=1,NoOfCoords
        WRITE(1,*) Coords(I,1),Coords(I,2)
      END DO
      CLOSE(1)

      RETURN
      END

C ******************************************************************** PlotSolSet

      SUBROUTINE PlotSolSet(NoOfSC,TotDist,Dist,BCPtsY,SolPtsY)

      INTEGER NoOfSC,J
      INCLUDE 'ECBHdr.INC'
      REAL Dist(MaxNoOfCoords),BCPtsY(MaxNoOfCoords),
     & SolPtsY(MaxNoOfCoords)
      REAL*8 TotDist

      OPEN (UNIT=1,FILE='ECBPLOT.SOL',STATUS='NEW',FORM='FORMATTED')
      OPEN (UNIT=2,FILE='ECBPLOT.SET',STATUS='NEW',FORM='FORMATTED')

      WRITE(1,*) NoOfSC
      WRITE(2,*) NoOfSC

      DO J=1,NoOfSC
        WRITE(1,*) Dist(J)/TotDist,SolPtsY(J)
        WRITE(2,*) Dist(J)/TotDist,BCPtsY(J)
      END DO

      CLOSE(1)
      CLOSE(2)

      RETURN
      END

C ********************************************************************** PlotSrc

      SUBROUTINE PlotSrc

      INTEGER I
      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBSRC.INC'

C Put Source Points into a plottable file

      OPEN(UNIT=1,FILE='ECBPLOT.SRC',STATUS='NEW',FORM='FORMATTED')
      WRITE(1,*) NoOfSrc
      DO I=1,NoOfSrc
        WRITE(1,*) Src(I,1),Src(I,2)
      END DO
      CLOSE(1)

      RETURN
      END

C ******************************************************************** ProcessData
```

```
      SUBROUTINE ProcessData(I,Elem,Loc,IntegrandX,Integrand,TotDist,Dist,
     & BCPtsY,SolPtsY)

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBElm.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBBC.INC'

      INTEGER I,Elem,Loc
      REAL Dist(MaxNoOfCoords),BCPtsY(MaxNoOfCoords),
     & SolPtsY(MaxNoOfCoords)
      REAL*8 TotDist
      REAL*8 Integrand(MaxNoOfCoords),IntegrandX(MaxNoOfCoords)

      IntegrandX(I)=TotDist
      IF (ElemNat(Elem)) THEN
        Integrand(I)=BCValues(NodeMap(Elem,Loc))
      ELSE
        Integrand(I)=SolValues(NodeMap(Elem,Loc))
      END IF
      IF (Axisymmetric) Integrand(I)=Integrand(I)*ElemR(1,Loc)

      SolPtsY(I)=SolValues(NodeMap(Elem,Loc))

      Dist(I)=TotDist
      BCPtsY(I)=BCValues(NodeMap(Elem,Loc))

      RETURN
      END

C ******************************************************************** ProcessInt

      SUBROUTINE ProcessInt

      INTEGER NoOfPot,I
      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBINT.INC'

      REAL*8 Pot,PotMax,PotMin

C Read, Process & Plot internal points

      CALL MatExtremaDP(InternalPot,NoOfInternal,PotMax,PotMin)

      OPEN(UNIT=3,FILE='POT.INT',STATUS='OLD',FORM='FORMATTED')
      READ(3,*) NoOfPot
      DO I=1,NoOfPot
        READ(3,*) Pot
        CALL PlotInt(Pot,I)
      END DO
      CLOSE(3)

      RETURN
      END

C ******************************************************************** ProcessSolSet
```

```
      SUBROUTINE ProcessSolSet(Start,End)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBElm.INC'

      INTEGER I,Start,End,Elem,Loc
      REAL Dist(MaxNoOfCoords),BCPtsY(MaxNoOfCoords),
     & SolPtsY(MaxNoOfCoords)
      REAL*8 TotDist,Seg
      REAL*8 Integrand(MaxNoOfCoords),IntegrandX(MaxNoOfCoords)
      REAL*8 Break(MaxNoOfCoords),CSCoef(4,MaxNoOfCoords)

      REAL*8 DCSITG
      EXTERNAL DCSITG

      TotDist=0.0

      I=1
      DO Elem=Start,End

        CALL GetElemR(Elem)

        DO Loc=1,BasisDeg

          Seg=SQRT((ElemR(1,Loc+1)-ElemR(1,Loc))**2+
     & (ElemR(2,Loc+1)-ElemR(2,Loc))**2)

          CALL ProcessData(I,Elem,Loc,IntegrandX,Integrand,TotDist,Dist,
     & BCPtsY,SolPtsY)

          TotDist=TotDist+Seg
          I=I+1

        END DO

      END DO

      CALL ProcessData(I,End,BasisDeg+1,IntegrandX,Integrand,
     & TotDist,Dist,BCPtsY,SolPtsY)

C Store Data in Plot Files

      CALL PlotSolSet(I,TotDist,Dist,BCPtsY,SolPtsY)

C Carry out integration

      CALL DCSAKM(I,integrandx,Integrand,Break,CSCoef)
      WRITE(*,*) '1/Integral=',1/DCSITG(0.0,TotDist,I-1,Break,CSCoef)

      WRITE(*,*) ' '
      WRITE(*,*) ' ECBPLOT.SOL    ',
     & 'Solution Values, Relative TotDistance'
      WRITE(*,*) ' ECBPLOT.SET    ',
     & 'Proscribed Values, Relative TotDistance'
      WRITE(*,*) ' '

      RETURN
      END
```

```
C ********************************************************************** StartEnd

      SUBROUTINE StartEnd(Start,End)

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBMsh.INC'
      INTEGER Start,End,T

C Get the Start and End Element

      CALL EditI('Starting Elem Number? (<=0 to quit)',Start)
      IF (Start.GT.0) THEN

        CALL EditI('Ending Elem Number?',End)

        IF (Start.GT.End) THEN
          T=Start
          Start=End
          End=T
        END IF

        Start=MAX(1,Start)
        Start=MIN(NoOfElems,Start)

        End=MIN(End,NoOfElems)
        End=MAX(End,1)

      ELSE
        End=0
      END IF

      RETURN
      END

c *
```

```
C File: ECBSUBS.FOR

C Boundary Movement subroutines.

C       REAL*8 FUNCTION CrossProduct2D(Ax,Ay,Bx,By)
C       REAL*8 FUNCTION DotProduct(Ax,Ay,Az,Bx,By,Bz)
C       REAL*8 FUNCTION GeomSeries(Ratio,NoOfTerms)
C       LOGICAL FUNCTION MacNat(BCType)

c       SUBROUTINE AddOne(NoOfM,M,X,Y)
c       SUBROUTINE ChkIsoPara(Mac)
c       SUBROUTINE ChkIsoParaErr(Dim,Mic,MajorError)
c       SUBROUTINE ComputeBdry(Moved,NoOfMoved,
cglobal SUBROUTINE ComputeNx(Moved,NoOfElems,Nx)
c       SUBROUTINE ComputePrevElem(Elem,PrevElems)
c       SUBROUTINE ComputeRelativeSpacing(...
cglobal SUBROUTINE CopyPoint(Coord,Orig,NoOfMoved,Moved)
c       SUBROUTINE CopyPoint2(Coord,Orig,NoOfMoved,Moved)
c       SUBROUTINE LocateIntersects(Sign,NoOfMoved,Moved,
c       SUBROUTINE MakeCoord(X,Y,MicNod,MacElm,SaveKinetic)
c       SUBROUTINE MakeElemBC
c       SUBROUTINE MakeMacElem(Mac,SaveKinetic,Smooth)
c       SUBROUTINE MakeMeshPts(SaveKinetic,Smooth)
cglobal SUBROUTINE MakeMoved(ElemStart,ElemEnd,NodeMap,Orig,
cglobal SUBROUTINE MakeNodeMap
cglobal SUBROUTINE Move(NoOfMoved,Moved,dXdN,Nx,dt)
c       SUBROUTINE ObtAngAtBdryBeg(Ins,NoOfIns,...
c       SUBROUTINE ObtAngAtBdryEnd(Ins,NoOfIns,...
c       SUBROUTINE ObtAngErr
cglobal SUBROUTINE ReadMac(FileName)
cglobal SUBROUTINE RemoveBdryIntersects(NoOfMoved,Moved)
c       SUBROUTINE RemoveBdryAfter(NoOfMoved,Moved,I,XInt,YInt)
c       SUBROUTINE RemoveBdryBefore(NoOfMoved,Moved,I,XInt,YInt)
cglobal SUBROUTINE RemoveInsIntersects(Orig,Ins,NoOfIns,
cglobal SUBROUTINE RemoveLocIntersects(NoOfMoved,Moved)

c       SUBROUTINE ShpCir(MacElm,SaveKinetic)
c       SUBROUTINE ShpFlat(MacElm,SaveKinetic)
c       SUBROUTINE ShpFlatHiPt(MacElm,MaxY)
c       SUBROUTINE ShpLine(MacElm,SaveKinetic)
c       SUBROUTINE ShpPrev(MacElm,SaveKinetic)
c       SUBROUTINE ShpSine(MacElm,SaveKinetic)
c       SUBROUTINE ShpTRNS(MacElm,SaveKinetic)
c       SUBROUTINE ShpUser(MacElm,SaveKinetic,Smooth)

c       SUBROUTINE WriteMac(FileName)
c       SUBROUTINE XY2T(X,Y,T)

C Read/Write subroutines

c       SUBROUTINE ErrorFilename(Filename,RoutineName)
c       SUBROUTINE ErrorVersion(Version,FileName,RoutineName)
c       SUBROUTINE ReadCal(FileName)
c       SUBROUTINE ReadMesh(FileName)
c       SUBROUTINE ReadPar(FileName)
c       SUBROUTINE ReadSrc(FileName)
c       SUBROUTINE WriteCal(FileName)
c       SUBROUTINE WriteMesh(FileName)
```

```
c       SUBROUTINE WriteSrc(FileName)

C Boundary Element Routines.

c       SUBROUTINE AssembleAC
c       SUBROUTINE AssembleGH
c       SUBROUTINE Integrate(Nodes)

C Kinetic Boundary Condition routines for BEM method.

c       SUBROUTINE NewtonRaphson(NoOfdBCValues,dBCValues,MagDiff)
c       SUBROUTINE UpdateBCValues(dBCValues)
c       SUBROUTINE ECBKinetics(KinTyp,Slope,Pot,Coord)
c       SUBROUTINE Kinetics(Slope,Pot,Elem,X,Y)

C Miscellaneous I/O routines for BEM.

c       SUBROUTINE GetElemR(Elem)
c       SUBROUTINE GetSrcPoint(Source)
c       SUBROUTINE TestForSingularity(Singularity)
c       SUBROUTINE PrintMesh(Lun)
c       SUBROUTINE PrintPar
c       SUBROUTINE PrintSrc(Lun)

C ****************************************************************** AcuteAngles

        SUBROUTINE AcuteAngles

C The following subroutine determine if the angles between the moving
C boundary and the insulator on either side are acute.

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMOVE.INC'

        REAL*8 NCross

C External Functions

        REAL*8 CrossProduct2D
        EXTERNAL CrossProduct2D

D       WRITE(*,10) 'Moved',Moved(1,1),Moved(1,2),1
D       WRITE(*,10) 'Ins First',Ins(First,1),Ins(First,2),First
D       WRITE(*,10) 'Ins Second',Ins(Second,1),Ins(Second,2),Second
D       WRITE(*,*) ' '
D       WRITE(*,10) 'Ins NxtLst',Ins(NextToLAst,1),Ins(NextToLast,2),
D     & NextToLast
D       WRITE(*,10) 'Ins Last',Ins(Last,1),Ins(Last,2),Last
D       WRITE(*,10) 'Moved',Moved(NoOfMoved,1),Moved(NoOfMoved,2),
D     & NoOfMoved
10      FORMAT(X,A15,G10.3,2X,G10.3,:,2X,I4)

C The following lines check the beginning of the moving boundary

        NCross=CrossProduct2D(Moved(1,1)-Ins(First,1),
     & Moved(1,2)-Ins(First,2),
     & Ins(Second,1)-Ins(First,1),
     & Ins(Second,2)-Ins(First,2))
```

```
D       IF (StaRev) THEN
D         WRITE(*,*) 'Bdry Start Moving Backwards'
D       ELSE
D         WRITE(*,*) 'Bdry Start Moving Forwards'
D       END IF

D       WRITE(*,*) 'Cross Product=',NCross
        AcuteAngleBefore=(((NCross.GT.0.0).AND.(StaRev)).OR.
     & ((NCross.LT.0.0).AND.(.NOT.StaRev)))

        NoAngleBefore=(NCross.EQ.0.0)

D       IF (AcuteAngleBefore) THEN
D         WRITE(*,*) 'At Moving Bdry Start: Acute Angle'
D       ELSE
D         WRITE(*,*) 'At Moving Bdry Start: Obtuse Angle'
D       END IF

C The following lines determine if the angle between the end of the boundary
C and the insulator is acute.

D       WRITE(*,*) ' '
        NCross=CrossProduct2D(Moved(NoOfMoved,1)-Ins(Last,1),
     & Moved(NoOfMoved,2)-Ins(Last,2),
     & Ins(NextToLast,1)-Ins(Last,1),
     & Ins(NextToLast,2)-Ins(Last,2))

D       IF (EndRev) THEN
D         WRITE(*,*) 'Bdry End Moving Backwards'
D       ELSE
D         WRITE(*,*) 'Bdry End Moving Forwards'
D       END IF

D       WRITE(*,*) 'Cross Product=',NCross

        AcuteAngleAfter=(((NCross.GT.0.0).AND.(.NOT.EndRev)).OR.
     & ((NCross.LT.0.0).AND.(EndRev)))
        NoAngleAfter=(NCross.EQ.0.0)

D       IF (AcuteAngleAfter) THEN
D         WRITE(*,*) '   At Moving Bdry End: Acute Angle'
D       ELSE
D         WRITE(*,*) '   At Moving Bdry End: Obtuse Angle'
D       END IF

        RETURN
        END

C ************************************************************************** AddOne

        SUBROUTINE AddOne(NoOfM,M,X,Y)

C This subroutine adds the coordinates x,y to the list of coordinates
C stored in M.

        INTEGER NoOfM
        REAL*8 X,Y
        INCLUDE 'ECBHdr.INC'
```

```
      REAL*8 M(MaxNoOfCoords,MaxNoOfDim)

      NoOfM=NoOfM+1
      M(NoOfM,1)=X
      M(NoOfM,2)=Y

      RETURN
      END

C ******************************************************************** AssembleAC

      SUBROUTINE AssembleAC

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBBC.INC'

      INTEGER Row,Col,Elem,LastCol
      REAL*8 Temp

C Re-arrange matrices G and H so that G will multiply the vector of unknowns.
C Before this subroutine is executed, G multiplies
C the natural conditions, so we must exchange the Column with its counterpart
C in H.  After this routine is executed, G multiplies the unknowns; H, the
C knowns.

      DO Col=1,NoOfCoords
        IF (Natural(Col)) THEN
          DO Row=1,NoOfCoords
              Temp        =  BemH(Row,Col)
              BemH(Row,Col)= -BemG(Row,Col)
              BemG(Row,Col)= -Temp
          END DO
        END IF
      END DO

      RETURN
      END

C ******************************************************************** AssembleGH

      SUBROUTINE AssembleGH

C This subroutine
C       1) formulates G (stored in BemG)
C       2) formulates H (stored in BEMH)
C Note: the following is set up for 2nd order basis functions

C Common Variables

      INCLUDE 'ECBPar.INc'
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECBG.INC'
      INCLUDE 'ECBCal.INC'
```

```
C Internal Variables

      INTEGER Elem,Row,Col,Loc
      INCLUDE 'EcbElm.INC'
      REAL*8 Sum

C Clear the BemG and BemH Matrices

      CALL MatZeroDP(MaxNoOfCoords,NoOfCoords,NoOfCoords,BemG)
      CALL MatZeroDP(MaxNoOfCoords,NoOfCoords,NoOfCoords,BemH)

C The following DO-Loop performs the integration by Element, computing
C the components of the G and H matrices

      DO Elem=1,NoOfElems
        CALL PrepInt(Elem)
        CALL Integrate(Elem)
      END DO

C Obtain diagonal elements of H.

      DO Row=1,NoOfCoords
        BEMH(Row,Row)=0.0
        Sum=0.0
        DO Col=1,NoOfCoords
          Sum=Sum+BEMH(Row,Col)
        END DO
        BemH(Row,Row)=-Sum
      END DO

      RETURN
      END

C ************************************************************************** Axi2B3

      REAL*8 FUNCTION Axi2B3(GaussPt)

C For an Axisymmetric Problem, with 2nd Order basis functions,
C this function returns the bounded portion of the integrand for
C loc=3

      INCLUDE 'ECBHDR.INC'
      REAL*8 GaussPt

      REAL*8 Den,A,B,UstarB,KB,RBig,Eta
      REAL*8 R(MaxNoOfDim), IsoJacob
      REAL*8 Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=1-GaussPt
      CALL AxiSub(Eta,3,A,B,Den,R,IsoJacob,Phi,RBig)

      KB=A+B*LOG(Den)-B*LOG(RBig)
      UStarB=2*KB/SQRT(Den)
      Axi2B3 = UStarB*IsoJacob*r(1)*Phi(3)

      RETURN
```

```
      END
C ********************************************************************** Axi2B2
      REAL*8 FUNCTION Axi2B2(GaussPt)

C For an Axisymmetric Problem, with 2nd Order basis functions,
C this function returns the bounded portion of the integrand
C for loc=2

      REAL*8 GaussPt,Eta

      INCLUDE 'ECBHDR.INC'
      REAL*8 Den,A,B,UstarB,KB,RBig
      REAL*8 R(MaxNoOfDim), IsoJacob
      REAL*8 Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=(1.0-GaussPt)/2.0
      CALL AxiSub(Eta,2,A,B,Den,R,IsoJacob,Phi,RBig)

      KB=A+B*LOG(Den)-B*LOG(RBig)
      UStarB=KB/SQRT(Den)
      Axi2B2 = UStarB*IsoJacob*r(1)*Phi(2)

      Eta=(1.0+GaussPt)/2.0
      CALL AxiSub(Eta,2,A,B,Den,R,IsoJacob,Phi,RBig)

      KB=A+B*LOG(Den)-B*LOG(RBig)
      UStarB=KB/SQRT(Den)
      Axi2B2 = Axi2B2 + UStarB*IsoJacob*r(1)*Phi(2)

      RETURN
      END
C ********************************************************************** Axi2B1
      REAL*8 FUNCTION Axi2B1(GaussPt)

C For an Axisymmetric Problem, with 2nd Order basis functions,
C this function returns the bounded portion of the integrand for
C loc=1

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 Den,A,B,UstarB,KB,RBig
      REAL*8 R(MaxNoOfDim), IsoJacob
      REAL*8 Phi(MaxBasisDeg+1)

C From Common ElemR

      CALL AxiSub(GaussPt,1,A,B,Den,R,IsoJacob,Phi,RBig)

      KB=A+B*LOG(Den)-B*LOG(RBig)
      UStarB=2*KB/SQRT(Den)
      Axi2B1 = UStarB*IsoJacob*r(1)*Phi(1)
```

```
      RETURN
      END

C ************************************************************************ Axi2U3

      REAL*8 FUNCTION Axi2U3(GaussPt)

C For an Axisymmetric Problem, with 2nd Order basis functions,
C this function returns the unbounded portion of the integrand for
C loc=3 (Note: the unbounded LN(1/GaussPt) term is NOT included.  This
C is accounted for in the integration routine points and weights)

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 Den,A,B,UstarU,KU,RBig,Eta
      REAL*8 R(MaxNoOfDim), IsoJacob
      REAL*8 Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=1-GaussPt
      CALL AxiSub(Eta,3,A,B,Den,R,IsoJacob,Phi,RBig)

      KU=2*B
      UStarU=2*KU/SQRT(Den)

      Axi2U3 = UStarU*IsoJacob*r(1)*Phi(3)

      RETURN
      END

C ************************************************************************ Axi2U2

      REAL*8 FUNCTION Axi2U2(GaussPt)

C For an Axisymmetric Problem, with 2nd Order basis functions,
C this function returns the unbounded portion of the integrand for
C loc=2 (Note: the unbounded LN(1/GaussPt) term is NOT included.  This
C is accounted for in the integration routine points and weights)

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 Den,A,B,UstarU,KU,Eta,RBig
      REAL*8 R(MaxNoOfDim), IsoJacob
      REAL*8 Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=(1.0-GaussPt)/2.0
      CALL AxiSub(Eta,2,A,B,Den,R,IsoJacob,Phi,RBig)

      KU=B
      UStarU=2*KU/SQRT(Den)

      Axi2U2 = UStarU*IsoJacob*r(1)*Phi(2)
```

```
      Eta=(1.0+GaussPt)/2.0
      CALL AxiSub(Eta,2,A,B,Den,R,IsoJacob,Phi,RBig)

      KU=B
      UStarU=2*KU/SQRT(Den)

      Axi2U2 = Axi2U2 + UStarU*IsoJacob*r(1)*Phi(2)

      RETURN
      END

C ********************************************************************** Axi2U1

      REAL*8 FUNCTION Axi2U1(GaussPt)

C For an Axisymmetric Problem, with 2nd Order basis functions,
C this function returns the unbounded portion of the integrand for
C loc=1 (Note: the unbounded LN(1/GaussPt) term is NOT included.  This
C is accounted for in the integration routine points and weights)

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 Den,A,B,UstarU,KU,RBig
      REAL*8 R(MaxNoOfDim), IsoJacob
      REAL*8 Phi(MaxBasisDeg+1)

C From Common ElemR

      CALL AxiSub(GaussPt,1,A,B,Den,R,IsoJacob,Phi,RBig)

      KU=2*B
      UStarU=2*KU/SQRT(Den)

      Axi2U1 = UStarU*IsoJacob*r(1)*Phi(1)

      RETURN
      END

C ********************************************************************** AxiSub

      SUBROUTINE AxiSub(GaussPt,Loc,A,B,Den,R,IsoJacob,Phi,RBig)

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBPAR.INC'
      INCLUDE 'ECBELM.INC'

      INTEGER Loc
      REAL*8 GaussPt,A,B,Den,R(*),IsoJacob,Phi(*),RBig

      REAL*8 M,M1,dRdZeta(MaxNoOfDim),PhiZeta(MaxBasisDeg+1)

C From Common ElemR, SourceR

C Functions

      REAL*8 IsoJacobian
      EXTERNAL IsoJacobian
```

```
C From Common BasisDeg, ElemR, SourceR

      CALL Compute1DPhis(BasisDeg,GaussPt,Phi,PhiZeta)
      CALL ComputeRZ(Phi,R)

      Den = (R(1)+SourceR(1))*(R(1)+SourceR(1)) +
     & (r(2)-SourceR(2))*(r(2)-SourceR(2))
      M=4*r(1)*SourceR(1)/Den

      IF (M.GE.1) THEN
        WRITE(*,*) '      M=',M,'    Den=',Den
        WRITE(*,*) '      R=',R(1),R(2)
        WRITE(*,*) ' Source=',SourceR(1),SourceR(2)
        WRITE(*,*) 'GaussPt=',GaussPt,'    Loc=',Loc
        WRITE(*,*) ' '
      END IF

      CALL KelpCoef(M,M1,A,B)

      IsoJacob=IsoJacobian(PhiZeta,dRdZeta)

      CALL ComputeRBig(GaussPt,Loc,RBig)

      RETURN
      END

C ************************************************************************ ChkIsoPara

      SUBROUTINE ChkIsoPara(Mac)

      INTEGER MAc
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBElm.INC'
      INCLUDE 'ECBMac.INC'

      INTEGER PrevElems,Mic,Dim
      REAL*8 Test

      LOGICAL MajorError

      CALL ComputePrevElem(Mac,PrevElems)
      DO Mic=PrevElems+1,PrevElems+NoOfMicElems(Mac)

        MajorError=.FALSE.
        CALL GetElemR(Mic)

        DO Dim=1,NoOfDim

          Test=ElemR(Dim,1) + ElemR(Dim,3) - 2.0 * ElemR(Dim,2)

          IF (Test.NE.0.0) THEN

              Test= (ElemR(Dim,1) - ElemR(Dim,3))/Test
              IF (ABS(Test).LE.2.0) CALL ChkIsoParaErr(Dim,Mic,MajorError)

          END IF
        END DO
      END DO
```

```
      RETURN
      END

C ************************************************************** ChkIsoParaErr

      SUBROUTINE ChkIsoParaErr(Dim,Mic,MajorError)

      INTEGER Dim,Mic
      LOGICAL MajorError

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBElm.INC'
      INCLUDE 'ECBMsh.INC'

      IF (.NOT.MajorError) THEN
        MajorError=.TRUE.
      ELSE
        WRITE(*,*) ' '
        WRITE(*,*) ' Error: isoparametric mapping from global ',
     & 'element to local element is definitely not 1-1'
        WRITE(*,*) ' '
        WRITE(*,*) '       Element # ',Mic
        WRITE(*,*) ' '
        WRITE(*,*) '       ElemR(,1)',ElemR(1,1),ElemR(2,1)
        WRITE(*,*) '       ElemR(,2)',ElemR(1,2),ElemR(2,2)
        WRITE(*,*) '       ElemR(,3)',ElemR(1,3),ElemR(2,3)
        WRITE(*,*) ' '
        WRITE(*,*) '       NodeMap(,1..3)',NodeMap(Mic,1),
     & NodeMap(Mic,2),NodeMap(Mic,3)

        CALL WriteMesh('ISOPARA.ERR')

        STOP 'In Routine: ChkIsoPara (ChkIsoParaErr)'
      END IF

      RETURN
      END

c ************************************************************** CirLinNtrsxn

      SUBROUTINE CirLinNtrsxn(X1,Y1,X2,Y2,Xc,Yc,R,
     & XInt1,YInt1,XInt2,YInt2,Error)

C Input Variables ------------------------------------------------------------

      REAL*8 X1,Y1,X2,Y2,Xc,Yc,R

C X1,Y1        Starting point of line segment
C X2,Y2        Ending point of line segment

C Xc,YcCenter of Circle
C R     Radius of Circle

C Output Variables -----------------------------------------------------------

      REAL*8 XInt1,YInt1,XInt2,YInt2
      INTEGER Error
```

```
C Xint1,YInt1    First intersection between circle and line segment
C XInt2,YInt2    Second intersection between circle and line segment

C Error 0 if only 1 intersection 1 is on the line segment
C       1 if two intersections are on the line segment
C       2 if no intersections are on the line segment, but are on the line
C       3 if no intersections are on the line.

C Internal Variables ------------------------------------------------------

      REAL*8 Xhat,Yhat,dX,dY,b_2a_y,c_a_y,b_2a_x,c_a_x,DiscX,DiscY
      REAL*8 Temp
      LOGICAL Tangent,One,Two

C Xhat,Yhat     x1-xc, y1-yc
C dx,dy         x1-x2,y1-y2
C b_2a_y        dx(yhat dx - xhat dy)/(dx dx + dy dy)
C c_a_y         (r r dy dy - ( xhat dy - yhat dx )^2)/(dx dx + dy dy)
C b_2a_x        dy(xhat dy ...
C c_a_y         (r r dx dx ...

      xhat=x1-xc
      yhat=y1-yc

      dx=x1-x2
      dy=y1-y2

      b_2a_y = dx*(yhat*dx - xhat*dy) / (dx*dx + dy*dy)
      b_2a_x = dy*(xhat*dy - yhat*dx) / (dx*dx + dy*dy)

      c_a_y = ( r*r*dy*dy -
     & (xhat*dy - yhat*dx)*(xhat*dy - yhat*dx))
     & / (dx*dx + dy*dy)
      c_a_x = ( r*r*dx*dx -
     & (xhat*dy - yhat*dx)*(xhat*dy - yhat*dx))
     & / (dx*dx + dy*dy)

      DiscX = b_2a_x*b_2a_x + c_a_x
      DiscY = b_2a_y*b_2a_y + c_a_y

      Tangent=.FALSE.
      IF (DiscY.LT.0) THEN

        Error=3
        RETURN

      ELSE IF ((DiscY.EQ.0).AND.(DiscX.EQ.0)) THEN

        XInt1=b_2a_x
        YInt1=b_2a_y

        XInt2=b_2a_x
        YInt2=b_2a_y

        Tangent=.TRUE.

      ELSE IF ((DiscY.EQ.0).AND.(DiscX.GT.0)) THEN
```

```
      XInt1 = b_2a_x + SQRT(DiscX)
      YInt1 = b_2a_y

      XInt2 = b_2a_x - SQRT(DiscX)
      YInt2 = b_2a_y

    ELSE IF ((DiscY.GT.0).AND.(DiscX.EQ.0)) THEN

      XInt1 = b_2a_x
      YInt1 = b_2a_y + SQRT(DiscY)

      XInt2 = b_2a_x
      YInt2 = b_2a_y - SQRT(DiscY)

    ELSE IF ((DiscY.GT.0).AND.(DiscX.GT.0)) THEN

      XInt1 = b_2a_x + SQRT(DiscX)
      YInt1 = dy/dx * ( xInt1 - xhat) + yhat

      XInt2 = b_2a_x - SQRT(DiscX)
      YInt2 = dy/dx * ( xInt2 - xhat) + yhat

    END IF

    XInt1=XInt1 + xc
    YInt1=YInt1 + yc

    XInt2=XInt2 + xc
    YInt2=YInt2 + yc

    One = ( ((MIN(X1,X2).LE.XInt1).AND.(XInt1.LE.MAX(X1,X2)))
&  .AND.
& ((MIN(Y1,Y2).LE.YInt1).AND.(YInt1.LE.MAX(Y2,Y1))) )

    Two = ( ((MIN(X1,X2).LE.XInt2).AND.(XInt2.LE.MAX(X1,X2)))
&  .AND.
& ((MIN(Y1,Y2).LE.YInt2).AND.(YInt2.LE.MAX(Y2,Y1))) )

    IF (Tangent) THEN

      IF (One) THEN
        Error=0
      ELSE
        Error=2
      END IF

    ELSE

      IF (One.AND.Two) THEN
        Error=1
      ELSE IF (One) THEN
        Error=0
      ELSE IF (Two) THEN
        Error=0

        Temp=XInt1
        XInt1=XInt2
        XInt2=Temp
```

```
            Temp=YInt1
            YInt1=YInt2
            YInt2=Temp

          ELSE
            Error=2
          END IF
        END IF

        RETURN
        END

C ************************************************************************ CirPts

        SUBROUTINE CirPts(Xc,Yc,X1,Y1,X2,Y2,NPts,PtsX,PtsY,Clockwise)

C This subroutine generates points on a circular arc
C (centered at xc,yc) starting with (x1,y1) and ending with (x2,y2)

C Input Variables ------------------------------------------------------------

        REAL*8 Xc,Yc,X1,Y1,X2,Y2
        LOGICAL Clockwise

C Xc,YcCenter of the Circle
C X1,Y1          First Point on Circular Arc
C X2,Y2 Last Point on Circular Arc
C Clockwise      .TRUE. for a clockwise circle

C Output Variables ------------------------------------------------------------

        INTEGER NPts
        REAL*8 PtsX(*),PtsY(*)

C NPts  The number of points on the arc
C PtsX(*)        X coordinate locations
C PtsY(*)        Y coordinate locations

C Internal Variables ----------------------------------------------------------

        REAL*8 T1,T2,TStp,T,R
        INTEGER I
        INCLUDE '[kj.for.util]pi.inc'

C External Functions ----------------------------------------------------------

        REAL*8 Mag
        EXTERNAL Mag

        R=(MAG(X1-Xc,Y1-Yc)+MAG(X2-Xc,Y2-Yc))/2

        CALL XY2T(X1-Xc,Y1-Yc,T1)
        CALL XY2T(X2-Xc,Y2-Yc,T2)

        IF ((ABS(T1-T2).LT.0.01)
     & .OR.(ABS(T1-T2).GT.(2.0*Pi-0.01))) THEN
          NPts=2
```

```
      ELSE

        IF (Clockwise) THEN
          DO WHILE(T1.LE.T2)
            T1=T1+2.0*Pi
          END DO
        ELSE
          DO WHILE(T1.GE.T2)
            T1=T1-2.0*Pi
          END DO
        END IF

        NPts=ABS((T2-T1)*18.0/Pi)+2
        TStp=(T2-T1)/(NPts-1)

        T=T1+TStp
        DO I=2,NPts-1
          PtsX(I)=Xc+R*COS(T)
          PtsY(I)=Yc+R*SIN(T)
          T=T+TStp
        END DO

      END IF

      PtsX(1)=X1
      PtsY(1)=Y1
      PtsX(Npts)=X2
      PtsY(NPts)=Y2

      RETURN
      END

C ******************************************************************** ComputeBdry

      SUBROUTINE ComputeBdry(Spa,NoOfSpa,Smooth,Spl3D)

C This subroutine computes the new y coordinates based on Spa.  An IMSL Cubic
C Spline routine is used to generate the new boundary points from moved points.

C Input Variables ---------------------------------------------------------

C UserPts()      Locations of X and Y Coordinates to use in Spline
C NoOfUserPTs    Number of X,Y Coordinates to use spline with.
C Spa()          Location of X coordinates to find Y coordinates (using Spline)
C NoOfSpa        Number of X locs at which we wish to find Y locs.
C Smooth         .TRUE. if smoothing spline is to be used.

C Output Variables --------------------------------------------------------

C UserPts()      Desired Locations of X and Y Coordinates

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMove.INC'

      REAL*8 PtsX(MaxNoOfCoords),PtsY(MaxNoOfCoords)
      REAL*8 Spa(*),OrigSpa(MaxNoOfCoords)
      REAL*8 YBreak(MaxNoOfCoords),YCSCoef(4,MaxNoOfCoords)
```

```
      REAL*8 XBreak(MaxNoOfCoords),XCSCoef(4,MaxNoOfCoords)
      INTEGER I,J,NoOfSpa,Offset

      LOGICAL Smooth,Spl3D

      REAL*8 DCSVAL
      EXTERNAL DCSVAL

      DO I=1,NoOfMoved
        PtsX(I)=Moved(I,1)
        PtsY(I)=Moved(I,2)
      END DO

      IF (Spl3D) THEN

c       IF (Smooth) THEN
c         WRITE(*,*) 'Error: Smoothing Not implemented in this version'
c       END IF

D       WRITE(*,*) 'ComputeBdry: Calling ComputeRelativeSpacing'
        CALL ComputeRelativeSpacing(Moved,NoOfMoved,OrigSpa)

        Offset=0
        DO I=1,NoOfMoved-1
          IF (ABS(OrigSpa(I-Offset)-OrigSpa(I+1))
     & .LE.Tolerance/100.0) THEN
            Offset=Offset+1
          END IF
          OrigSpa(I-Offset+1)=OrigSpa(I+1)
          PtsX(I-Offset+1)=PtsX(I+1)
          PtsY(I-Offset+1)=PtsY(I+1)
        END DO
        NoOfMoved=NoOfMoved-Offset

        IF (Smooth) THEN

C Use Shape smoothing spline

D         WRITE(*,*) 'Calling DCSSCV from ComputeBdry:',
D    & ' x(1..',NoOfMoved,')'
          CALL DCSSCV(NoOfMoved,OrigSpa,PtsX,2,XBreak,XCSCoef)
D         WRITE(*,*) 'Calling DCSSCV from ComputeBdry:',
D    & ' y(1..',NoOfMoved,')'
          CALL DCSSCV(NoOfMoved,OrigSpa,PtsY,2,YBreak,YCSCoef)

        ELSE

C Use Akima Spline

D         WRITE(*,*) 'Calling DCSAKM from ComputeBdry:',
D    & ' x(1..',NoOfMoved,')'
          CALL DCSAKM(NoOfMoved,OrigSpa,PtsX,XBreak,XCSCoef)
D         WRITE(*,*) 'Calling DCSSCV from ComputeBdry:',
D    & ' y(1..',NoOfMoved,')'
          CALL DCSSCV(NoOfMoved,OrigSpa,PtsY,YBreak,YCSCoef)

        END IF
```

```
C Generate the new points from the old X locations.

          DO J=1,NoOfSpa
            Moved(J,1)=DCSVAL(Spa(J),NoOfMoved-1,XBreak,XCSCoef)
            Moved(J,2)=DCSVAL(Spa(J),NoOfMoved-1,YBreak,YCSCoef)
          END DO

        ELSE

        Offset=0
        DO I=1,NoOfMoved-1
          IF (ABS(PtsX(I-Offset)-PtsX(I+1))
     & .LE.Tolerance/100) THEN
            Offset=Offset+1
          END IF
          PtsX(I-Offset+1)=PtsX(I+1)
          PtsY(I-Offset+1)=PtsY(I+1)
        END DO
        NoOfMoved=NoOfMoved-Offset

          IF (NoOfMoved.LE.3) THEN
            DO J=1,NoOfSpa
              Moved(J,1)=Spa(J)
              IF (PtsX(NoOfMoved).EQ.PtsX(1)) THEN
                WRITE(*,*) 'Error: X points are not varying.'
                STOP 'In Routine: ComputeBdry'
              END IF
              Moved(J,2)=PtsY(NoOfMoved)-(PtsY(NoOfMoved)-PtsY(1))
     & / (PtsX(NoOfMoved)-PtsX(1)) * (PtsX(NoOfMoved)-Spa(J))
            END DO
          ELSE

            IF (Smooth) THEN

D             WRITE(*,*) 'Calling DCSSCV from ComputeBdry: ',
D    & 'y(x(1..',NoOfMoved,'))'
              CALL DCSSCV(NoOfMoved,PtsX,PtsY,2,YBreak,YCSCoef)

            ELSE

D             WRITE(*,*) 'Calling DCSAKM from ComputeBdry: ',
D    & 'y(x(1..',NoOfMoved,'))'
              CALL DCSAKM(NoOfMoved,PtsX,PtsY,YBreak,YCSCoef)

            END IF

            DO J=1,NoOfSpa
              Moved(J,1)=Spa(J)
              Moved(J,2)=DCSVAL(Spa(J),NoOfMoved-1,YBreak,YCSCoef)
            END DO
          END IF

        END IF

        NoOfMoved=NoOfSpa

        RETURN
        END
```

```
C ******************************************************************** ComputeInt

      SUBROUTINE ComputeInt

C This subroutine (for internal points)

C       0) stores the internal points into the source points
C       1) formulates G (stored in BemG)
C       2) formulates H (stored in BemH)
C       3) computes Pot for internal points

C Note: the following is set up for 2nd order basis functions
C Note: the source points ARE destroyed by this subroutine, so, it
C       should always be called last, after the BEM problem has been solved
C       and saved.

C Common Variables

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBINT.INC'
      INCLUDE 'ECBG.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBPar.INc'
      INCLUDE 'ECBSRC.INC'
      INCLUDE '[kj.for.util]pi.inc'

C Internal Variables

      INTEGER Elem,Row,Col,Loc,I,J
      REAL*8 U,Q
      INCLUDE 'EcbElm.INC'

C Clear the BemG and BemH Matrices

      CALL MatZeroDP(MaxNoOfCoords,NoOfCoords,NoOfCoords,BemG)
      CALL MatZeroDP(MaxNoOfCoords,NoOfCoords,NoOfCoords,BemH)

C Store the internal points into the source points

      NoOfSrc=NoOfInternal
      DO I=1,NoOfInternal
        DO J=1,NoOfDim
          Src(I,J)=InternalPts(I,J)
        END DO
      END DO

C The following DO-Loop performs the integration by Element, computing
C the components of the G and H matrices

      DO Elem=1,NoOfElems
        CALL PrepInt(Elem)
        CALL Integrate(Elem)
      END DO

C Compute the value of the potential at the internal points.
```

```
      DO I=1,NoOfInternal
        InternalPot(I)=0.0
        DO J=1,NoOfCoords

          IF (Natural(J)) THEN
            U=SolValues(J)
            Q=BCValues(J)
          ELSE
            U=BCValues(J)
            Q=SolValues(J)
          END IF

          InternalPot(I)=InternalPot(I)+
     & BemG(I,J)*Q-BemH(I,J)*U

        END DO
        InternalPot(I)=InternalPot(I)/(2.0*Pi)
      END DO

      RETURN
      END

C ********************************************************************** ComputeNx

      SUBROUTINE ComputeNx(NoOfElems)

C This subroutine computes the unit normals for all the boundary points.
C K is a counter variable.  Assumes moving boundary has been placed into MOVE

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBElm.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER Elem,Loc,K,Dim,NoOfElems
      REAL*8 dRdZeta(MaxNoOfDim),
     & Zeta,Norm,Phi(MaxBasisDeg+1),PhiZeta(MaxBasisDeg+1)
      REAL*8 DeltaX,DeltaY

      REAL*8 VecNormDP
      EXTERNAL VecNormDP

      K=0
      DO Elem=1,NoOfElems

C Transfer points from global Moved matrix to local ELEMR matrix

        DO Loc=1,BasisDeg+1
          K=K+1
          DO Dim=1,NoOfDim
            ElemR(Dim,Loc)=Moved(K,Dim)
          END DO
        END DO

C Compute the normal to each element at each node

D       WRITE(*,*) 'ComputeNx: ElemR'
D       WRITE(*,*) '  (,1)',ElemR(1,1),ElemR(2,1)
D       WRITE(*,*) '  (,2)',ElemR(1,2),ElemR(2,2)
```

```
D         IF (BasisDeg+1.GT.2) WRITE(*,*) '  (,3)',ElemR(1,3),ElemR(2,3)

          DeltaX=ElemR(1,2)-ElemR(1,1)
          DeltaY=ElemR(2,2)-ElemR(2,1)

          Norm=SQRT(DeltaX*DeltaX+DeltaY*DeltaY)
          Nx(K+1-BasisDeg-1,1) =-DeltaY/Norm
          Nx(K+1-BasisDeg-1,2) = DeltaX/Norm

D         WRITE(*,*) '    Nx(,1)=',-DeltaY/Norm,K+1-BasisDeg-1
D         WRITE(*,*) '    Nx(,2)=', DeltaX/Norm

          DO Loc=2,BasisDeg

            DeltaX=ElemR(1,Loc)-ElemR(1,Loc-1)
            DeltaY=ElemR(2,Loc)-ElemR(2,Loc-1)

            DeltaX=DeltaX+ElemR(1,Loc+1)-ElemR(1,Loc)
            DeltaY=DeltaY+ElemR(2,Loc+1)-ElemR(2,Loc)

            Norm=SQRT(DeltaX*DeltaX+DeltaY*DeltaY)
            Nx(K+Loc-BasisDeg-1,1) =-DeltaY/Norm
            Nx(K+Loc-BasisDeg-1,2) = DeltaX/Norm

D           WRITE(*,*) '    Nx(,1)=',-DeltaY/Norm,K+Loc-BasisDeg-1
D           WRITE(*,*) '    Nx(,2)=', DeltaX/Norm

          END DO

          Loc=BasisDeg+1
          DeltaX=ElemR(1,Loc)-ElemR(1,Loc-1)
          DeltaY=ElemR(2,Loc)-ElemR(2,Loc-1)

          Norm=SQRT(DeltaX*DeltaX+DeltaY*DeltaY)
          Nx(K+Loc-BasisDeg-1,1) =-DeltaY/Norm
          Nx(K+Loc-BasisDeg-1,2) = DeltaX/Norm
D         WRITE(*,*) '    Nx(,1)=',-DeltaY/Norm,K+Loc-BasisDeg-1
D         WRITE(*,*) '    Nx(,2)=', DeltaX/Norm

        END DO

        RETURN
        END

C ************************************************************** ComputePrevElem

        SUBROUTINE ComputePrevElem(Elem,PrevElems)

C Computes the previous number of micro-elements (PrevElem) from
C macro-element(1) to macro-element(Elem).

        INTEGER Elem,PrevElems
        INCLUDE 'ECBMAC.INC'
        INTEGER J

        PrevElems=0
        DO J=1,Elem-1
          PrevElems=PrevElems+NoOfMicElems(J)
```

```
      END DO

      RETURN
      END

C ******************************************************************** ComputeRBig

      SUBROUTINE ComputeRBig(GaussPt,Loc,RBig)

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBElm.INC'
      INTEGER Loc
      REAL*8 GaussPt,RBig

      REAL*8 RBigr,RBigz

      IF (Loc.EQ.1) THEN

        RBigr = ElemR(1,1)*(-3+2*GaussPt)
     & + ElemR(1,2)*4*(1-GaussPt)
     & + ElemR(1,3)*(2*GaussPt-1)
        RBigz = ElemR(2,1)*(-3+2*GaussPt)
     & + ElemR(2,2)*4*(1-GaussPt)
     & + ElemR(2,3)*(2*GaussPt-1)

      ELSE IF (Loc.EQ.2) THEN

        RBigr = ElemR(1,1)*(1-GaussPt)
     & - ElemR(1,2)*(1-2*GaussPt)
     & - ElemR(1,3)*GaussPt
        RBigz = ElemR(2,1)*(1-GaussPt)
     & - ElemR(2,2)*(1-2*GaussPt)
     & - ElemR(2,3)*GaussPt

      ELSE IF (Loc.EQ.3) THEN

        RBigr = ElemR(1,1)*(1-2*GaussPt)
     & + ElemR(1,2)*4*GaussPt
     & - ElemR(1,3)*(1+2*GaussPt)
        RBigz = ElemR(2,1)*(1-2*GaussPt)
     & + ElemR(2,2)*4*GaussPt
     & - ElemR(2,3)*(1+2*GaussPt)

      ELSE

        WRITE(*,*) 'Error: Information for computing RBig has'
        WRITE(*,*) '       not been entered for Loc=',Loc
        STOP 'In Routine: ComputeRBig'

      END IF

      RBig = RBigr*RBigR + RBigZ*RBigZ
      RETURN
      END

C ******************************************************************** ComputeRZ

      SUBROUTINE ComputeRZ(Phi,R)
```

```
C This subroutine computes the isoparametric transformation

C Input Variables

C Phi(1),Phi(2),Phi(3)      The basis functions evaluated at the gauss point.
C ElemR(MaxNoOfDim,*)       The three coordinates on the element.

C Output Variables

C R,Z   The coordinates of the current point on the integration path.

      INCLUDE 'ECBHdr.INC'

      INCLUDE 'ECBElm.INC'
      REAL*8 Phi(*),R(*)
      INTEGER Loc,J

      CALL VecZeroDP(NoOfDim,R)

      DO Loc=1,BasisDeg+1
        DO J=1,NoOfDim
          R(J)=R(J)+Phi(Loc)*ElemR(J,Loc)
        END DO
      END DO

      RETURN
      END

C ************************************************************** CrossProduct2D

      REAL*8 FUNCTION CrossProduct2D(Ax,Ay,Bx,By)

      REAL*8 Ax,Ay,Bx,By

C Compute | A x B |

C |     I      J      K |
C |     ax     ay     0 |
C |     bx     by     0 |

      CrossProduct2D=Ax*By-Bx*Ay

      RETURN
      END

C ******************************************************** ComputeRelativeSpacing

      SUBROUTINE ComputeRelativeSpacing(Moved,NoOfMoved,Spacing)

      INCLUDE 'ECBHdr.INC'

      REAL*8 Moved(MaxNoOfCoords,MaxNoOfDim),Spacing(*),IncLength
      INTEGER I,NoOfMoved
      REAL*8 Mag
      EXTERNAL Mag

      Spacing(1)=0.0
```

```
      DO I=2,NoOfMoved
        IncLength=MAG(Moved(I,1)-Moved(I-1,1),Moved(I,2)-Moved(I-1,2))
        Spacing(I)=Spacing(I-1)+IncLength
      END DO

      IF (Spacing(NoOfMoved).EQ.0) THEN
        WRITE(*,*) 'Error: Boundary has no length!'
        STOP 'In Routine: ComputeRelativeSpacing'
      END IF

      DO I=2,NoOfMoved-1
        Spacing(I)=Spacing(I)/Spacing(NoOfMoved)
      END DO
      Spacing(NoOfMoved)=1.0

      RETURN
      END

C ******************************************************************** CopyPoint

      SUBROUTINE CopyPoint(Coord,Orig,NoOfMoved,Moved)

      INCLUDE 'ECBHdr.INC'

      INTEGER NoOfMoved,Coord,J
      REAL*8 Moved(MaxNoOfCoords,MaxNoOfDim),
     & Orig(MaxNoOfCoords,MaxNoOfDim)

      NoOfMoved=NoOfMoved+1
      CALL CopyPoint2(Coord,Orig,NoOfMoved,Moved)

      RETURN
      END

C ******************************************************************** CopyPoint2

      SUBROUTINE CopyPoint2(Coord,Orig,NoOfMoved,Moved)

      INCLUDE 'ECBHdr.INC'

      INTEGER NoOfMoved,Coord,J
      REAL*8 Moved(MaxNoOfCoords,MaxNoOfDim),
     & Orig(MaxNoOfCoords,MaxNoOfDim)

      DO J=1,NoOfDim
        Moved(NoOfMoved,J)=Orig(Coord,J)
      END DO

      RETURN
      END

C ******************************************************************** DotProduct

      REAL*8 FUNCTION DotProduct(Ax,Ay,Az,Bx,By,Bz)

      REAL*8 Ax,Ay,Az,Bx,By,Bz

      DotProduct=Ax*Bx+Ay*By+Az*Bz
```

```
      RETURN
      END

C ******************************************************************** ECBKinetics

      SUBROUTINE ECBKinetics(Pot,KinTyp,Slope,SlopePot,Coord)

C Given the kinetics type, and the potential slope, this subroutine computes
C the potential.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER Coord,Elem
      CHARACTER*(*) KinTyp
      REAL*8 Pot,SlopePot,Slope
      LOGICAL Ok

      Ok=.TRUE.

      IF (KinTyp.EQ.'BV') THEN
        Slope    = - (EXP(aA*Pot)-EXP(-aC*Pot))       / WagLin
        SlopePot = - (aA*EXP(aA*Pot)+aC*EXP(-aC*Pot)) / WagLin
      ELSE IF (KinTyp.EQ.'LIN') THEN
        Slope    = - Pot*(aA+aC) / WagLin
        SlopePot = - (aA+aC)     / WagLin
      ELSE IF (KinTyp.EQ.'TAFA') THEN
        Slope    = - EXP(aA*Pot)    / WagLin
        SlopePot = - aA*EXP(aA*Pot) / WagLin
      ELSE IF (KinTyp.EQ.'TAFC') THEN
        Slope    = - EXP(-aC*Pot)  / WagLin
        SlopePot = aC*EXP(-aC*Pot) / WagLin
      ELSE IF (KinTyp.EQ.'USR') THEN
        CALL Kinetics(Pot,Coords(Coord,1),Coords(Coord,2),Coord,
     & Slope,SlopePot,Ok)
      END IF

      IF (.NOT.Ok) THEN
        WRITE(*,10) Coord,Coords(Coord,1),Coords(Coord,2)
10      FORMAT(X,'        Out of range at ',I3,
     & '(',G9.2,',',G9.2,')')
      END IF

      RETURN
      END

C ********************************************************************** ErrSing

      SUBROUTINE ErrSing

C This subroutine prints the error message for singularities that the program
C cannot handle.

      WRITE(*,*) 'Error: Current Program cannot compute for'
      WRITE(*,*) '       singularities. Use source points outside'
      WRITE(*,*) '       of domain of interest:'
      STOP 'In Routine: Integrate'
```

```
      END

C ************************************************************** ErrorFileName

      SUBROUTINE ErrorFilename(Filename,RoutineName)

      CHARACTER*(*) FileName,RoutineName
      INTEGER Length

      CALL StringLength(FileName,Length)
      WRITE(*,*) 'Error: Filename "',FileName(1:Length),
   & '" doesn''t exist.'
      WRITE(*,*) 'In Routine: '//RoutineName

      STOP
      END

C ************************************************************** ErrorVersion

      SUBROUTINE ErrorVersion(Version,FileName,RoutineName)

      CHARACTER*(*) fileName,RoutineName
      INTEGER Version

      WRITE(*,*) 'Error: unable to read this version.'
      WRITE(*,*) '       Version=',Version
      WRITE(*,*) '       Filename=',FileName
      WRITE(*,*) 'In Routine: '//RoutineName

      STOP
      END

C ******************************************************************** GaussInt

      SUBROUTINE GaussInt(BemG,BemH,Loc)

      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBG.INC'
      INCLUDE 'EcbElm.INC'

C Passed Variables

      INTEGER Loc
      REAL*8 BemG,BemH

C Internal Variables

      REAL*8 Dist2,Gc,Hc,UBJacob,Cor,Den,M
      INTEGER Gs

C Functions

      REAL*8 UStar,QStar
      EXTERNAL UStar,QStar

C Subroutine Code
```

```
      DO Gs=1,NoOfGaussPts

        IF (GPhi(Gs,Loc).NE.0.0) THEN
          IF (Axisymmetric) THEN
              Den=(GR(Gs,1)+SourceR(1))*(GR(Gs,1)+SourceR(1))
     & + (GR(Gs,2)-SourceR(2))*(GR(Gs,2)-SourceR(2))
              M=4.0D0*GR(Gs,1)*SourceR(1)/Den
              IF (M.GE.1.0) THEN
                WRITE(*,*) '      M=',M,'   Gs=',Gs
                WRITE(*,*) '      R=',Gr(Gs,1),Gr(Gs,2)
                WRITE(*,*) 'SourceR=',SourceR(1),SourceR(2)
                WRITE(*,*) '  ElemR=',ElemR(1,1),ElemR(2,1)
                WRITE(*,*) '        ',ElemR(1,2),ElemR(2,2)
                WRITE(*,*) '        ',ElemR(1,3),ElemR(2,3)
                WRITE(*,*) ' '
              END IF
          END IF

          Dist2=(GR(Gs,1)-SourceR(1))**2 + (GR(Gs,2)-SourceR(2))**2

          Gc=UStar(Dist2,Gs,Den,M)*GaussWts(Gs)*GIsoJacob(Gs)
          BEMG = BEMG + Gc*GPhi(Gs,Loc)

C Note: the Hc term does not need to be multiplied by the isojacobian here
C because this has already been done when computing the normal.

          Hc=QStar(Dist2,Gs,Den,M)*GaussWts(Gs)
          BEMH = BEMH + Hc*GPhi(Gs,Loc)

        END IF
      END DO

      RETURN
      END

C ********************************************************************** GeomSeries

      REAL*8 FUNCTION GeomSeries(Ratio,NoOfTerms)

C This function returns the value of the sum of the geometric series
C sum(J=0,NoOfTerms) of (Ratio**J)

      REAL*8 Ratio
      INTEGER NoOfTerms,J

      GeomSeries=0.0D0
      DO J=0,NoOfTerms
        GeomSeries=GeomSeries+Ratio**J
      END DO

      RETURN
      END

C ********************************************************************** GetElemR

      SUBROUTINE GetElemR(Elem)

      INCLUDE 'ECBHdr.INC'
```

```
      INCLUDE 'ECBElm.INC'
      INCLUDE 'ECBMsh.INC'
      INTEGER Elem,Loc,Dim

      DO Loc=1,BasisDeg+1
        DO Dim=1,NoOfDim
          ElemR(Dim,Loc)=Coords(NodeMap(Elem,Loc),Dim)
        END DO
      END DO

      RETURN
      END

C ********************************************************************** GetSpa

      SUBROUTINE GetSpa(MacElm,Spa,NoOfSpa)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMove.INC'
      INTEGER NoOfSpa,MacElm
      REAL*8 Spa(*)

      INTEGER I,J
      REAL*8 D,X

      IF (MacSpac(MacElm).EQ.'=CRV') THEN

D       WRITE(*,*) 'GetSpa: Calling ComputeRelativeSpacing'
        CALL ComputeRelativeSpacing(Moved,NoOfMoved,Spa)
        CALL ReMesh(NoOfMicElems(MacElm)+1,Spa,NoOfMoved,Moved)
        NoOfSpa=NoOfMicElems(MacElm)*BasisDeg
        Spa(NoOfSpa+1)=1.0
        DO I=NoOfMicElems(MacElm),1,-1
          Spa( (I-1)*BasisDeg+1 )=Spa(I)
          D=Spa( I*BasisDeg+1)-Spa( (I-1)*BasisDeg+1 )
          DO J=1,BasisDeg-1
            Spa(I*BasisDeg+1-J)=Spa((I-1)*BasisDeg+1)+
     & D*FLOAT(J)/FLOAT(BasisDeg)
          END DO
        END DO

      ELSE

C Calculate Spacing

        CALL ShpSpacing(MacElm,NoOfSpa,Spa)

      END IF

      RETURN
      END

C ******************************************************************** GetSrcPoint

      SUBROUTINE GetSrcPoint(Source)
```

```
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBSrc.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBElm.INC'

      INTEGER Source,I

      IF ((Source.LT.1).OR.(Source.GT.NoOfSrc)) THEN
        WRITE(*,*) 'Error: Source point is out of range!'
        WRITE(*,*) '       Source Point #  =',Source
        WRITE(*,*) '       Range=(1,',NoOfSrc,')'
        STOP 'In routine: GetSourcePoint'
      END IF

      DO I=1,NoOfDim
        SourceR(I)=Src(Source,I)
      END DO

      RETURN
      END

C ********************************************************************** IntAxis

      SUBROUTINE IntAxis(GiiB,GiiU,SingLoc)

      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECGLOG.INC'

      INTEGER SingLoc
      REAL*8 GiiB,GiiU

C Functions

      REAL*8 Axi2b3,Axi2b2,Axi2b1,Axi2u3,Axi2u2,Axi2u1
      EXTERNAL Axi2b3,Axi2b2,Axi2b1,Axi2u3,Axi2u2,Axi2u1

      IF (SingLoc.EQ.3) THEN
        CALL GaussQuad(NoOfGaussPts,GaussPts,GaussWts,
     & Axi2B3,GiiB)
        CALL GaussQuad(NoOfGauLogPts,GauLogPts,GauLogWts,
     & Axi2U3,GiiU)
      ELSE IF (SingLoc.EQ.2) THEN
        CALL GaussQuad(NoOfGaussPts,GaussPts,GaussWts,
     & Axi2B2,GiiB)
        CALL GaussQuad(NoOfGauLogPts,GauLogPts,GauLogWts,
     & Axi2U2,GiiU)
      ELSE IF (SingLoc.EQ.1) THEN
        CALL GaussQuad(NoOfGaussPts,GaussPts,GaussWts,
     & Axi2B1,GiiB)
        CALL GaussQuad(NoOfGauLogPts,GauLogPts,GauLogWts,
     & Axi2U1,GiiU)
      END IF

      RETURN
      END

C ********************************************************************** IntPlan
```

```
      SUBROUTINE IntPlan(GiiB,GiiU,SingLoc)

      REAL*8 GiiB,GiiU
      INTEGER SingLoc

      REAL*8 Pla2b3,Pla2b2,Pla2b1,Pla2u3,Pla2u2,Pla2u1
      EXTERNAL Pla2b3,Pla2b2,Pla2b1,Pla2u3,Pla2u2,Pla2u1

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBElm.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECGLOG.INC'

      REAL*8 RSeg

      IF (BasisDeg.EQ.1) THEN

C Gii for Planar 1st degree basis functions

        RSeg=SQRT(
     & (ElemR(1,2)-ElemR(1,1))*(ElemR(1,2)-ElemR(1,1)) +
     & (ElemR(2,2)-ElemR(2,1))*(ElemR(2,2)-ElemR(2,1)) )

        GiiB=RSeg/2*(1-LOG(RSeg))
        GiiU=0.0

      ELSE IF (BasisDeg.EQ.2) THEN

C Gii for Planar 2nd degree basis functions

        IF (SingLoc.EQ.3) THEN
          CALL GaussQuad(NoOfGaussPts,GaussPts,GaussWts,
     & Pla2B3,GiiB)
          CALL GaussQuad(NoOfGauLogPts,GauLogPts,GauLogWts,
     & Pla2U3,GiiU)
        ELSE IF (SingLoc.EQ.2) THEN
          CALL GaussQuad(NoOfGaussPts,GaussPts,GaussWts,
     & Pla2B2,GiiB)
          CALL GaussQuad(NoOfGauLogPts,GauLogPts,GauLogWts,
     & Pla2U2,GiiU)
        ELSE IF (SingLoc.EQ.1) THEN
          CALL GaussQuad(NoOfGaussPts,GaussPts,GaussWts,
     & Pla2B1,GiiB)
          CALL GaussQuad(NoOfGauLogPts,GauLogPts,GauLogWts,
     & Pla2U1,GiiU)
        END IF

      ELSE
        CALL ErrSing
      END IF

      RETURN
      END

C ********************************************************************** Integrate

      SUBROUTINE Integrate(Elem)
```

```
C This subroutine performs the integrations for all source points over an
C element.

      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECGLOG.INC'
      INCLUDE 'ECBPar.INC'

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBG.INC'
      INCLUDE 'ECBSRC.INC'

C Input Variables

      INTEGER Elem
      INCLUDE 'ECBELM.INC'

C Internal Variables

      LOGICAL Singularity
      INTEGER Source,Loc,I,SingLoc

      REAL*8 Dist2,DTemp,Small/1.0D-05/,Gc,Hc
      REAL*8 GiiU,GiiB

C Code Follows

C The following Do-Loop performs the integration along the element for each
C source point.

      DO Source=1,NoOfSrc

        CALL GetSrcPoint(Source)
        CALL TestForSingularity(Singularity,SingLoc)

        DO Loc=1,BasisDeg+1
          IF (SingLoc.NE.Loc) THEN

C Integrate over elements w/o singularities

              CALL GaussInt(BEMG(Source,NodeMap(Elem,Loc)),
     & BEMH(Source,NodeMap(Elem,Loc)),Loc)

          ELSE

C Integrate over elements w/ singularities.

              IF (Axisymmetric) THEN

C Axisymmetric Elements w/ singularities

              IF (BasisDeg.NE.2) CALL ErrSing
              CALL IntAxis(GiiB,GiiU,SingLoc)

              ELSE

C Planar elements w/ singularities
```

```
                CALL IntPlan(GiiB,GiiU,SingLoc)

                END IF

                BemG(Source,NodeMap(Elem,Loc)) =
     & BemG(Source,NodeMap(Elem,Loc)) + GiiB + GiiU

            END IF
          END DO
        END DO

        RETURN
        END

C ******************************************************************** IsoJacobian

        REAL*8 FUNCTION IsoJacobian(PhiZeta,dRdZeta)

C This subroutine computes the jacobian for the iso-parametric transformation.

        INCLUDE 'ECBHdr.INC'

        INCLUDE 'EcbElm.INC'
        REAL*8 PhiZeta(*),dRdZeta(MaxNoOfDim)
        REAL*8 VecNormDP
        EXTERNAL VecNormDP

        CALL ComputeRZ(PhiZeta,dRdZeta)
        IsoJacobian=VecNormDP(NoOfDim,dRdZeta)

        RETURN
        END

C ******************************************************************** Kinetics

        SUBROUTINE Kinetics(Pot,X,Y,Coord,Slope,SlopePot,Ok)

C Return the potential as a function of the slope in potential

        INTEGER Coord
        REAL*8 Pot,Slope,SlopePot,X,Y
        LOGICAL Ok

        WRITE(*,*) 'Error: Kinetics routine has not been written'
        Ok=.FALSE.

        RETURN
        END

C ******************************************************************** KineticBC

        LOGICAL FUNCTION KineticBC(BCType)

        CHARACTER*(*) BCType

        KineticBC=(BCType.EQ.'LIN').OR.(BCType.EQ.'TAFA')
     & .OR.(BCType.EQ.'TAFC').OR.(BCType.EQ.'BV').OR.
```

```
     & (BCType.EQ.'USR')

          RETURN
          END

C ************************************************************** LocateIntersects

          SUBROUTINE LocateIntersects(Sign,I,J,XInt,YInt,Error)

          INCLUDE 'ECBHdr.INC'
          INCLUDE 'ECBMOVE.INC'

          REAL*8 XInt,YInt
          INTEGER Sign,I,J,ErrorInt

          INTEGER ILimits(2),JLimits(2)
          LOGICAL Error

D         WRITE(*,*) '---------: LocateIntersects'

          Error=.FALSE.
          IF (Sign.GT.0) THEN
            ILimits(1)=1
            ILimits(2)=NoOfMoved-1
            JLimits(1)=1
            JLimits(2)=NoOfIns-1
          ELSE
            ILimits(1)=NoOfMoved
            ILimits(2)=2
            JLimits(1)=NoOfIns
            JLimits(2)=2
          END IF

C Outer loop, goes down moved boundary points

          DO I=ILimits(1),ILimits(2),Sign

D           WRITE(*,*) '        I=',I
D           WRITE(*,*) '       M1 ',Moved(I,1),Moved(I,2)
D           WRITE(*,*) '       M2 ',Moved(I+Sign,1),Moved(I+Sign,2)

C Inner loop, goes down insulator points.

            DO J=JLimits(1),JLimits(2),Sign
              CALL ComputeIntersection(Moved(I,1),Moved(I,2),
     & Moved(I+Sign,1),Moved(I+Sign,2),
     & Ins(J,1),Ins(J,2),
     & Ins(J+Sign,1),Ins(J+Sign,2),XInt,YInt,ErrorInt)

D             WRITE(*,*) 'J=',J,'   ErrorInt=',ErrorInt
D             WRITE(*,*) '       I1 ',Ins(J,1),Ins(J,2)
D             WRITE(*,*) '       I2 ',Ins(J+Sign,1),Ins(J+Sign,2)

              IF (ErrorInt.EQ.0) THEN
D               WRITE(*,*) 'Int=',XInt,YInt
                RETURN
              END IF
            END DO
```

```
      END DO

      WRITE(*,*) 'Warning: no intersection found between ',
     & 'moved boundary and insulator.'
      WRITE(*,*) '       Time step is probably too big.'
      WRITE(*,*) ' Sign=',Sign
      WRITE(*,*) 'MOVED=',Moved(ILimits(1),1),Moved(ILimits(1),2)
      WRITE(*,*) '      ',Moved(ILimits(2),1),Moved(ILimits(2),2)
      WRITE(*,*) '  INS=',Ins(JLimits(1),1),Ins(JLimits(1),2)
      WRITE(*,*) '      ',Ins(JLimits(2),1),Ins(JLimits(2),2)

      WRITE(*,*) 'In Routine: LocateIntersects'
      Error=.TRUE.

      END

C ********************************************************************** MacNat

      LOGICAL FUNCTION MacNat(BCType)

C This function returns .TRUE. if BCType represents a natural boundary
C condition.

      CHARACTER*(*) BCType
      LOGICAL KineticBC
      EXTERNAL KineticBC

      IF (BCType.EQ.'CRNT') THEN
        MacNat=.TRUE.
      ELSE IF (KineticBC(BCType)) THEN

C For Kinetic BCs, the potential is guessed at and the current is solved
C for using BEM. Then potential is re-guessed using Newton-Raphson or
C Picard scheme.

        MacNat=.FALSE.
      ELSE IF (BCType.EQ.'NAT') THEN
        MacNat=.TRUE.
      ELSE IF (BCType.EQ.'ESS') THEN
        MacNat=.FALSE.
      ELSE
        WRITE(*,*) 'Error: Unknown Boundary Condition Type=',BCType
        STOP 'In Function: MacNat'
      END IF

      RETURN
      END

C ********************************************************************** Mag

      REAL*8 FUNCTION Mag(X,Y)

      REAL*8 X,Y

      Mag=SQRT(X*X+Y*Y)

      RETURN
      END
```

```
C ******************************************************************** MakeCoord

      SUBROUTINE MakeCoord(X,Y,MicNod,MacElm,SaveKinetic)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBPar.INC'

      INTEGER MicNod,MacElm
      REAL*8 X,Y
      LOGICAL KineticBC,SaveKinetic,MacNat
      EXTERNAL KineticBC,MacNat

      Coords(MicNod,1)=X
      Coords(MicNod,2)=Y

      BCType(MicNod)=MacBCType(MacElm)
      Natural(MicNod)=MacNat(MacBCType(MacElm))

      IF (MacBCType(MacElm).EQ.'CRNT') THEN

C BC Type CRNT no longer allowed.

        WRITE(*,*) 'Error: Mac BC Type ''CRNT'' is no longer allowed.'
        WRITE(*,*) '       Use BC Type ''NAT''  instead.'
        STOP 'In Routine: MakeCoord'

      ELSE IF (KineticBC(MacBCType(MacElm))) THEN

        IF (.NOT.SaveKinetic) THEN
          BCValues(MicNod)=MacBCVal(MacElm)
          SolValues(MicNod)=0.0D0
        END IF

      ELSE IF (MacBCType(MacElm).EQ.'NAT') THEN

        BCValues(MicNod)=MacBCVal(MacElm)
        SolValues(MicNod)=0.0D0

      ELSE IF (MacBCType(MacElm).EQ.'ESS') THEN

        BCValues(MicNod)=MacBCVal(MacElm)
        SolValues(MicNod)=0.0D0

      END IF

      RETURN
      END

C ******************************************************************** MakeElemBC

      SUBROUTINE MakeElemBC

C Get BC conditions for element from mid node (or first node if BasisDeg=1)

      INCLUDE 'ECBHdr.INC'
```

```
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBPar.INC'

      INTEGER I

      DO I=1,NoOfElems
        ElemNat(I)=Natural(NodeMap(I,BasisDeg))
        ElemBCType(I)=BCType(NodeMap(I,BasisDeg))
      END DO

      RETURN
      END

C ******************************************************************** MakeMacElem

      SUBROUTINE MakeMacElem(Mac,SaveKinetic,Smooth)

      INTEGER Mac,MacPlus1,Indx,Mic,Dim,PrevElems,MacPrev,MacNext
      LOGICAL SaveKinetic,Smooth,MacNat,TransNext
      EXTERNAL MacNat
      CHARACTER*4 Shp,NextShp
      REAL*8 MaxY,Test,MinY

      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBBC.INC'
      INCLUDE 'ECBElm.INC'

C Compute Coord Locations

      Shp=MacShape(Mac)
      CALL IncrementI(Mac,MacPlus1,1,NoOfMacElems)
      MacNext=MacPlus1
      NextShp=MacShape(MacNext)

      IF (NextShp.EQ.'TRNS') THEN
        TransNext=.TRUE.
        CALL IncrementI(MacPlus1,MacNext,1,NoOfMacElems)
        NextShp=MacShape(MacNext)
      ELSE
        TransNext=.FALSE.
      END IF

      IF (NextShp.EQ.'FLAT') THEN
        CALL ShpFlatHiPt(MacNext,MaxY)
        IF (TransNext) THEN
          MacVertY(MacIndx(MacPlus1,2))=
     & MaxY+MacPar(MacNext,1)
          MacVertY(MacIndx(MacPlus1,1))=
     & MacVertY(MacIndx(MacPlus1,2))-0.05
          MacVertY(MacIndx(Mac,2))=
     & MacVertY(MacIndx(MacPlus1,1))
        ELSE
          MacVertY(MacIndx(Mac,2))=MaxY+MacPar(MacNext,1)
        END IF
```

```
      ELSE IF (NextShp.EQ.'FLTL') THEN

C Flat spaced from low point.

        CALL ShpFlatLoPt(MacNext,MinY)
        IF (TransNext) THEN
          MacVertY(MacIndx(MacPlus1,2))=
     & MinY+MacPar(MacNext,1)
          MacVertY(MacIndx(MacPlus1,1))=
     & MacVertY(MacIndx(MacPlus1,2))-0.05
          MacVertY(MacIndx(Mac,2))=
     & MacVertY(MacIndx(MacPlus1,1))
        ELSE
          MacVertY(MacIndx(Mac,2))=MinY+MacPar(MacNext,1)
        END IF

      ELSE IF (NextShp.EQ.'PREV') THEN

C Must compute Prev boundary & Store Loc of MacroVertex

D       WRITE(*,*) ' MakeMacElem:First Call to ShpPrev'
        CALL ShpPrev(MacPlus1,SaveKinetic)

        IF (TransNext) THEN

          MacVertX(MacIndx(MacPlus1,2))=
     & MacVertX(MacIndx(MacNext,1))
          MacVertY(MacIndx(MacPlus1,2))=
     & MacVertY(MacIndx(MacNext,1))

          MacVertX(MacIndx(MacPlus1,1))=
     & MacVertX(MacIndx(MacNext,1))
          MacVertY(MacIndx(MacPlus1,1))=
     & MacVertY(MacIndx(MacNext,1))-0.05

          MacVertX(MacIndx(Mac,2))=
     & MacVertX(MacIndx(MacPlus1,1))
          MacVertY(MacIndx(Mac,2))=
     & MacVertY(MacIndx(MacPlus1,1))

        ELSE

          MacVertX(MacIndx(Mac,2))=
     & MacVertX(MacIndx(MacNext,1))
          MacVertY(MacIndx(Mac,2))=
     & MacVertY(MacIndx(MacNext,1))

        END IF

      END IF

      IF (Shp.EQ.'LINE') THEN
        CALL ShpLine(Mac,SaveKinetic)
      ELSE IF (Shp.EQ.'TRNS') THEN

        IF (MacBCVal(MacPlus1).EQ.0.0) THEN
          CALL IncrementI(Mac,MacPrev,-1,NoOfMacElems)
```

```
            MacVertX(MacIndx(Mac,1))=MacVertX(MacIndx(MacPrev,2))
            MacVertY(MacIndx(Mac,1))=MacVertY(MacIndx(MacPrev,2))

            MacVertX(MacIndx(Mac,2))=MacVertX(MacIndx(MacPrev,2))
            MacVertY(MacIndx(Mac,2))=MacVertY(MacIndx(MacPrev,2))-0.05

          END IF

          MacVertX(MacIndx(MacPlus1,1))=
     & MacVertX(MacIndx(Mac,2))
          MacVertY(MacIndx(MacPlus1,1))=
     & MacVertY(MacIndx(Mac,2))

          CALL ShpTrns(Mac,SaveKinetic)

        ELSE IF (Shp.EQ.'SINE') THEN
          CALL ShpSine(Mac,SaveKinetic)
        ELSE IF (Shp.EQ.'CIR') THEN
          CALL ShpCir(Mac,SaveKinetic)
        ELSE IF ((Shp.EQ.'USER').OR.
     & (Shp.EQ.'USR')) THEN
          CALL ShpUser(Mac,SaveKinetic,Smooth)
        ELSE IF (Shp.EQ.'PREV') THEN
D         WRITE(*,*) '  MakeMacElem:Second Call to ShpPrev'
          CALL ShpPrev(Mac,SaveKinetic)
        ELSE IF (Shp.EQ.'FLAT') THEN
          CALL ShpFlat(Mac,SaveKinetic)
        ELSE IF (Shp.EQ.'FLTL') THEN
          CALL ShpFltL(Mac,SaveKinetic)
        ELSE
          WRITE(*,*) 'Error: Unknown Mac Shp!=',Shp
          STOP 'In Routine: MakeMacElem'
        END IF

C Make the node at the end of the macroelement.  If this is not a double
C node, it will be re-made in the next loop.  However, we must have the last
C point on the element stored, so that we can next check the micro elements of
C the macro element to see that the iso-parametric transformation is 1-1.

        Indx=MacIndx(Mac,2)
        CALL MakeCoord(MacVertX(MacIndx(Mac,2)),MacVertY(MacIndx(Mac,2)),
     & MacEnd(Mac),Mac,SaveKinetic)

C Check the x,y locations on micro elements to make sure the isoparametric
C mapping is (1-1).

        IF (BasisDeg.EQ.1) RETURN

        IF (BasisDeg.NE.2) THEN

          WRITE(*,*) 'Warning: Unable to test nodes for 1-1',
     & ' isoparametric mapping'

        ELSE

D         WRITE(*,*) 'Shp ',Shp,'  Mac',Mac
          CALL ChkIsoPara(Mac)
```

```
      END IF

      RETURN
      END

C ******************************************************************** MakeMeshPts

      SUBROUTINE MakeMeshPts(SaveKinetic,Smooth)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBBC.INC'

      INTEGER Mac,Sum
      LOGICAL SaveKinetic,Smooth

      CALL MatSumI(NoOfMicElems,1,NoOfMacElems,NoOfElems)
      Sum=NoOfElems*BasisDeg
      IF (Sum.GT.MaxNoOfCoords) THEN
        WRITE(*,*) 'Error: ',Sum,' coordinates have been specified.',
     & '  I only have room for ',MaxNoOfCoords,'.'
        WRITE(*,*) '       Increase the parameter MaxNoOfCoords in ',
     & 'ECBMsh.INC and recompile, or, alter the'
        WRITE(*,*) '       number of microelements specified in your',
     & ' *.MAC file.'
        STOP 'In routine: MakeMeshPts'
      END IF

c     IF (Smooth) WRITE(*,*) 'Smoothing Spline will be used!'

      DO Mac=1,NoOfMacElems
        CALL MakeMacElem(Mac,SaveKinetic,Smooth)
      END DO

      CALL MakeElemBC

      RETURN
      END

C ******************************************************************** MakeMoved

      SUBROUTINE MakeMoved(ElemStart,ElemEnd,NodeMap,Coords)

C Passed Variables

C ElemStart     Moving boundary starts with first point of
C               microelement ElemStart
C ElemEnd       Moving Boundary ends with last point of microelement ElemEnd
C Coords(,)     (original) Coordinates of mesh

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER ElemStart,ElemEnd,
     & NodeMap(MaxNoOfCoords,MaxBasisDeg+1)
      REAL*8 Coords(MaxNoOfCoords,MaxNoOfDim)
```

```
      INTEGER Elem,Loc

      NoOfMoved=0

C Forces the 1st point on the boundary to be an endpoint for an element

D     WRITE(*,*) '    MakeMoved: ElemStart=',ElemStart,
D   & '     ElemEnd=',ElemEnd

      DO Elem=ElemStart,ElemEnd
        DO Loc=1,BasisDeg+1
          CALL CopyPoint(NodeMap(Elem,Loc),Coords,NoOfMoved,Moved)
        END DO
      END DO

      Orig(1,1)=Moved(2,1)
      Orig(1,2)=Moved(2,2)
      Orig(2,1)=Moved(NoOfMoved-1,1)
      Orig(2,2)=Moved(NoOfMoved-1,2)

      RETURN
      END

C ******************************************************************* MakeNodeMap

      SUBROUTINE MakeNodeMap

      INTEGER Mac,Loc,I,PrevElem,MacPlus1
      LOGICAL MacNat
      EXTERNAL MacNat

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBMac.INC'

      NoOfCoords=1
      DO Mac=1,NoOfMacElems

        MacStart(Mac)=NoOfCoords

C Compute NodeMap

        CALL ComputePrevElem(Mac,PrevElem)
        DO I=PrevElem+1,PrevElem+NoOfMicElems(Mac)

          DO Loc=1,BasisDeg+1
            NodeMap(I,Loc)=NoOfCoords
            NoOfCoords=NoOfCoords+1
          END DO
          NoOfCoords=NoOfCoords-1

        END DO

C Make Corner Points if necessary (Double node where BC changes)

        CALL IncrementI(Mac,MacPlus1,1,NoOfMacElems)

        MacEnd(Mac)=NoOfCoords
```

```
          IF (MacNat(MacBCType(MacPlus1)).NE.MacNat(MacBCType(Mac)))
     & NoOfCoords=NoOfCoords+1

        END DO

        NoOfCoords=NoOfCoords-1
        IF (MacNat(MacBCType(1)).EQ.MacNat(MacBCType(NoOfMacElems)))
     & THEN
          CALL ComputePrevElem(NoOfMacElems,PrevElem)
          NodeMap(PrevElem+NoOfMicElems(NoOfMacElems),BasisDeg+1)=1
        END IF

        RETURN
        END

C ******************************************************************** MakeSrcPts

        SUBROUTINE MakeSrcPts

        INCLUDE 'ECBHDR.INC'
        INCLUDE 'ECBSRC.INC'
        INCLUDE 'ECBPAR.INC'
        INCLUDE 'ECBMsh.INC'
        INCLUDE 'ECBBC.INC'
        INCLUDE '[kj.for.util]pi.inc'
        INCLUDE 'EcbElm.INC'

        REAL*8 R,Zeta,Phi(MaxBasisDeg+1),
     & PhiZeta(MaxBasisDeg+1),dRdZeta(MaxNoOfDim),VecNormDP
        REAL*8 Norm,Nx(MaxNoOfDim)
        INTEGER I,Ip1,NSrc,Loc
        EXTERNAL VecNormDP

        NoOfSrc=NoOfCoords

        NSrc=0
        DO I=1,NoOfElems

          CALL GetElemR(I)
          R = SQRT( (ElemR(1,1)-ElemR(1,2))*(ElemR(1,1)-ElemR(1,2)) +
     & (ElemR(2,1)-ElemR(2,2))*(ElemR(2,1)-ElemR(2,2)) ) +
     & SQRT( (ElemR(1,2)-ElemR(1,3))*(ElemR(1,2)-ElemR(1,3)) +
     & (ElemR(2,2)-ElemR(2,3))*(ElemR(2,2)-ElemR(2,3)) )

          DO Loc=1,BasisDeg+1
            Zeta=FLOAT(Loc-1)/FLOAT(BasisDeg)
            CALL Compute1DPhis(BasisDeg,Zeta,Phi,PhiZeta)

            CALL ComputeRZ(PhiZeta,dRdZeta)
            Norm=VecNormDP(NoOfDim,dRdZeta)

C Outward Pointing Normal.

            IF (Norm.NE.0) THEN
                Nx(1) = dRdZeta(2)/Norm
                Nx(2) =-dRdZeta(1)/Norm
            ELSE
                WRITE(*,*) 'Error: no normal to surface!'
```

```
            WRITE(*,*) '       ElemR=',ElemR(1,1),ElemR(2,1)
            WRITE(*,*) '             ',ElemR(1,2),ElemR(2,2)
            WRITE(*,*) '             ',ElemR(1,3),ElemR(2,3)
            WRITE(*,*) '        Loc=',Loc,'   Elem=',I
            Stop 'In routine: MakeSrcPts'
          END IF

          NSrc=NSrc+1
          Src(NSrc,1)=Coords(NodeMap(I,Loc),1)+SrcFactor*Nx(1)*R
          Src(NSrc,2)=Coords(NodeMap(I,Loc),2)+SrcFactor*Nx(2)*R
        END DO

        CALL IncrementI(I,IP1,1,NoOfCoords)
        IF (ElemBCType(I).EQ.ElemBCType(IP1)) NSrc=NSrc-1

      END DO

      RETURN
      END

C ************************************************************************ Move

      SUBROUTINE Move(dt)

C This subroutine moves the coords.

C Variables-----------------------------------------------------------------

C dt            The size of the current time step.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER I,J
      REAL*8 dt

      DO I=1,NoOfMoved
        DO J=1,NoOfDim
          Moved(I,J)=Moved(I,J)-dXdN(I)*Nx(I,J)*dt
        END DO
      END DO

      RETURN
      END

C ************************************************************** NewtonRaphson

      SUBROUTINE NewtonRaphson(Iteration,NoOfdBCValues,
     & dBCValues,MagDiff,MagBC)

C Newton Raphson Iteration Scheme.  This is guaranteed to converge if our
C first guess is close enough to the solution.  For Kinetics BCs, the
C current is solved for using BEM.  The potential guess is adjusted
C until the current solved for does not differ from the current using
C kinetic relation.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
```

```
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBPAR.INC'
      INCLUDE 'ECBBC.INC'

      REAL*8 dBCValues(*),MagDiff,MagBC
      REAL*8 Jacobian(MaxNoOfCoords,MaxNoOfCoords),
     & VecB1(MaxNoOfCoords),SlopePot,Slope,VecB(MaxNoOfCoords)
      REAL*8 SlopePots(MaxNoOfCoords)
      INTEGER NoOfdBCValues,JCol,JRow,I,J,Iteration
      LOGICAL KineticBC
      EXTERNAL KineticBC

      MagDiff=0.0
      MagBC=0.0

      JCol=1

      DO I=1,NoOfCoords
        IF (KineticBC(BCType(I))) THEN

          IF (Iteration.LT.1) THEN

C Compute the Bem Part of the Jacobian

          DO J=1,NoOfCoords
            VecB1(J)=BemH(J,I)
          END DO
          CALL SolveRelEq(VecB1,dBCValues)

          JRow=1
          DO J=1,NoOfCoords
            IF (KineticBC(BCType(J))) THEN
                Jacobian(JRow,JCol)=dBCValues(J)
              JRow=JRow+1
            END IF
          END DO

          ELSE
            Jacobian(JCol,JCol)=Jacobian(JCol,JCol)+SlopePots(JCol)
          END IF

C Compute the Kinetic Part of the Jacobian

          CALL ECBKinetics(BCValues(I),BCType(I),
     & Slope,SlopePots(JCol),I)
          Jacobian(JCol,JCol)=Jacobian(JCol,JCol)-SlopePots(JCol)

C The following lines compute the B vector for the matrix equation
C Jacobian.dBCValues=VecB

          VecB(JCol)=-(SolValues(I)-Slope)
          MagDiff=MagDiff+VecB(JCol)*VecB(JCol)
          MagBC=MagBC+BCValues(I)*BCValues(I)

C Increment number of columns

          JCol=JCol+1
```

```
          END IF
        END DO

C The following line computes the number of Kinetic bc values

        NoOfdBCValues=JCol-1

        IF (NoOfdBCValues.LE.0) RETURN

        MagDiff=SQRT(MagDiff)
        MagBC=SQRT(MagBC)

C Call IMSL Routine DLSARG to get new BCValues by solving equation
C dBCValues=INV(Jacobian)VecB

        CALL DLSARG(NoOfdBCValues,Jacobian,MaxNoOfCoords,VecB,1,
     & dBCValues)

        RETURN
        END

C ************************************************************** ObtAngAtBdryBeg

        SUBROUTINE ObtAngAtBdryBeg

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMOVE.INC'

        REAL*8 Xc,Yc,R,XInt1,YInt1,XInt2,YInt2
        REAL*8 PtsX(MaxNoOfCoords),PtsY(MaxNoOfCoords)
        INTEGER I,J,K,Error,NPts

        REAL*8 Mag
        EXTERNAL Mag

D       WRITE(*,*) '---------: ObtAngAtBdryBeg'

        Xc=Ins(First,1)
        Yc=Ins(First,2)
        R=Mag(Moved(1,1)-Xc,Moved(1,2)-Yc)

        IF (.NOT.StaRev) THEN

          DO J=First,NoOfIns-1
            CALL CirLinNtrsxn(Ins(J,1),Ins(J,2),Ins(J+1,1),Ins(J+1,2),
     &    Xc,Yc,R,XInt1,YInt1,XInt2,YInt2,Error)
            IF (Error.EQ.0) GOTO 1
            IF (Error.EQ.1) THEN
                WRITE(*,*) 'Error: Moved boundary intersects insulator in'
                WRITE(*,*) '       two possible locations.'
                STOP 'In Routine: ObtAngAtBdryBeg'
            END IF
          END DO
          CALL ObtAngErr

        ELSE IF (First.NE.1) THEN

          J=1
```

```
          CALL CirLinNtrsxn(Ins(J,1),Ins(J,2),Ins(J+1,1),Ins(J+1,2),
     & Xc,Yc,R,XInt1,YInt1,XInt2,YInt2,Error)
          IF (Error.EQ.1) THEN
            WRITE(*,*) 'Error: Moved boundary intersects insulator in'
            WRITE(*,*) '       two possible locations.'
            STOP 'In Routine: ObtAngAtBdryBeg'
          ELSE IF (Error.NE.0) THEN
            CALL ObtAngErr
          END IF

        ELSE

          WRITE(*,*) 'Error: Start of boundary is moving backwards, and'
          WRITE(*,*) '       insulator is not extended.'
          WRITE(*,*) '       To extend the insulator, make a file: INS.EXT'
          WRITE(*,*) '       containing: x at Beg, y at Beg'
          WRITE(*,*) '                   x at End, y at End'
          STOP 'In Routine: ObtAngAtBdryBeg'

        END IF

1       CALL RemoveBdryBefore(NoOfIns,Ins,J,XInt1,YInt1)
        Last=Last-J+1
        NextToLast=NextToLast-J+1

C Use Counterclockwise circle for backward boundary movement,
C Clockwise circle for forward boundary movement.

        CALL CirPts(Xc,Yc,XInt1,YInt1,Moved(1,1),Moved(1,2),
     & NPts,PtsX,PtsY,.NOT.StaRev)

10      FORMAT(X,A40,2X,G12.4,2X,G12.4)

D       WRITE(*,*) '      Bdry Start'
D       IF (.NOT.StaRev) THEN
D         WRITE(*,10) 'Counter-Clockwise Circle, Center:',Xc,Yc
D       ELSE
D         WRITE(*,10) 'Clockwise Circle, Center:',Xc,Yc
D       END IF
D       WRITE(*,10) 'From:',XInt1,YInt1
D       WRITE(*,10) 'To:',Moved(1,1),Moved(1,2)

        IF (NPts.EQ.2) THEN
          IF ((Xint1.EQ.Moved(1,1)).AND.(YInt1.EQ.Moved(1,2))) THEN
            RETURN
          END IF
        END IF

C Bump Points up to make room for new points

          DO I=NoOfMoved,1,-1
            DO K=1,NoOfDim
              Moved(I+NPts-1,K)=Moved(I,K)
            END DO
          END DO

C Add New Points
```

```
D       WRITE(*,*) 'Adding Points at Beginning:'
        DO I=1,NPts-1
          Moved(I,1)=PtsX(I)
          Moved(I,2)=PtsY(I)
D         WRITE(*,*) PtsX(I),PtsY(I)
        END DO
        NoOfMoved=NoOfMoved+NPts-1

        RETURN
        END

C ************************************************************** ObtAngAtBdryEnd

        SUBROUTINE ObtAngAtBdryEnd

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMOVE.INC'

        REAL*8 XInt1,YInt1,XInt2,YInt2,R,Xc,Yc
        REAL*8 PtsX(MaxNoOfCoords),PtsY(MaxNoOfCoords)
        INTEGER I,J,K,Error,NPts

        REAL*8 Mag
        EXTERNAL Mag

        R=Mag(Moved(NoOfMoved,1)-Ins(Last,1),
    & Moved(NoOfMoved,2)-Ins(Last,2))
        Xc=Ins(Last,1)
        Yc=Ins(Last,2)

        IF (.NOT.EndRev) THEN

          DO J=Last,2,-1
            CALL CirLinNtrsxn(Ins(J,1),Ins(J,2),Ins(J-1,1),Ins(J-1,2),
    &     Xc,Yc,R,XInt1,YInt1,XInt2,YInt2,Error)
            IF (Error.EQ.0) GOTO 1
            IF (Error.EQ.1) THEN
                WRITE(*,*) 'Error: Moved boundary intersects insulator in'
                WRITE(*,*) '       two possible locations.'
                STOP 'In Routine: ObtAngAtBdryEnd'
            END IF
          END DO
          CALL ObtAngErr

        ELSE IF (Last.NE.NoOfIns) THEN

          J=NoOfIns
          CALL CirLinNtrsxn(Ins(J,1),Ins(J,2),Ins(J-1,1),Ins(J-1,2),
    & Xc,Yc,R,XInt1,YInt1,XInt2,YInt2,Error)
          IF (Error.EQ.1) THEN
            WRITE(*,*) 'Error: Moved boundary intersects insulator in'
            WRITE(*,*) '       two possible locations.'
            STOP 'In Routine: ObtAngAtBdryEnd'
          ELSE IF (Error.NE.0) THEN
            CALL ObtAngErr
          END IF

        ELSE
```

```
          WRITE(*,*) 'Error: End of boundary is moving backwards, and'
          WRITE(*,*) '       insulator is not extended.'
          WRITE(*,*) '       To extend the insulator, make a file: INS.EXT'
          WRITE(*,*) '       containing: x at Beg, y at Beg'
          WRITE(*,*) '                   x at End, y at End'
          WRITE(*,*) 'Last=',Last,'NoOfIns=',NoOfIns
          STOP 'In Routine: ObtAngAtBdryEnd'

        END IF

1       CALL RemoveBdryAfter(NoOfIns,Ins,J,XInt1,YInt1)

C Use clockwise circle for forward boundary movement, counterclockwise
C for backward movement.

D       WRITE(*,*) '      Bdry End'
D       IF (.NOT.EndRev) THEN
D         WRITE(*,10) 'Clockwise Circle, Center:',Xc,Yc
D       ELSE
D         WRITE(*,10) 'Counter-Clockwise Circle, Center:',Xc,Yc
D       END IF
D       WRITE(*,10) 'From:',Moved(NoOfMoved,1),
D     & Moved(NoOfMoved,2)
D       WRITE(*,10) 'To:',XInt1,YInt1

        CALL CirPts(Xc,Yc,Moved(NoOfMoved,1),Moved(NoOfMoved,2),
     & XInt1,YInt1,NPts,PtsX,PtsY,.NOT.EndRev)

10      FORMAT(X,A40,2X,G12.4,2X,G12.4)

C Store new points.

        IF (NPts.EQ.2) THEN
          IF ((Xint1.EQ.Moved(NoOfMoved,1))
     & .AND.(YInt1.EQ.Moved(NoOfMoved,2))) THEN
            RETURN
          END IF
        END IF

D       WRITE(*,*) 'Adding Points at Ending:'
        DO I=2,NPts
          Moved(NoOfMoved+I-1,1)=PtsX(I)
          Moved(NoOfMoved+I-1,2)=PtsY(I)
D         WRITE(*,*) PtsX(I),PtsY(I)
        END DO

        NoOfMoved=NoOfMoved+NPts-1

        RETURN
        END

C ******************************************************************** ObtAngErr

        SUBROUTINE ObtAngErr

        WRITE(*,*) 'Error: No intersection found between insulator and',
     & ' moved boundary.'
        STOP 'In routine: ObtAngErr'
```

```
      END

C ********************************************************************** Picard

      SUBROUTINE Picard(NoOfdBCValues,dBCValues,MagDiff)

C This subroutine uses a Picard iteration scheme instead of Newton-Raphson
C type of iteration. The Picard scheme, if it converges, can be much faster
C than the Newton-Raphson scheme. For Kinetic BCs, the current is solved for
C using BEM. The potential guess is adjusted until the current solved for
C does not differ from the current using kinetic relation.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      INCLUDE 'ECBBC.INC'

      REAL*8 dBCValues(*),MagDiff,VecNormDP
      REAL*8 dBCValue,Jacobian(MaxNoOfCoords,MaxNoOfCoords),
     & VecB(MaxNoOfCoords),SlopePot
      REAL*8 Slope(MaxNoOfCoords),PotPert,PotUnpert
      INTEGER NoOfdBCValues,JCol,JRow,I,J
      LOGICAL KineticBC,FindSlope
      EXTERNAL KineticBC,VecNormDP

      WRITE (*,*) 'Error: Picard routine called. This routine doesn''t'
      WRITE(*,*) 'currently work! Sorry...'

c     MagDiff=0.0
c     FindSlope=.FALSE.
c
c     dBCValue=0.0001
c     JCol=1
c     DO I=1,NoOfCoords
c       IF (KineticBC(BCType(I))) THEN
c
c         CALL ECBKinetics(BCType(I),SolValues(I),PotUnpert,I,
c    & FindSlope)
c         dBCValues(JCol)=PotUnPert-BCValues(I)
c         MagDiff=MagDiff+dBCValues(JCol)*dBCValues(JCol)
cc        JCol=JCol+1
c
c       END IF
c     END DO

c     NoOfdBCValues=JCol-1
c     MagDiff=SQRT(MagDiff)


      STOP 'In Routine: Picard'
      END

C ********************************************************************** Pla2B3

      REAL*8 FUNCTION Pla2B3(GaussPt)

C For an Planar Problem, with 2nd Order basis functions,
C this function returns the bounded portion of the integrand for
```

```
C loc=3

      INCLUDE 'ECBHDR.INC'
      REAL*8 GaussPt

      REAL*8 UstarB,RBig,Eta,IsoJacob,Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=1-GaussPt
      CALL PlaSub(Eta,3,IsoJacob,Phi,RBig)

      UStarB=-0.5*LOG(RBig)
      Pla2B3 = UStarB*IsoJacob*Phi(3)

      RETURN
      END

C ********************************************************************** Pla2B2

      REAL*8 FUNCTION Pla2B2(GaussPt)

C For an Planar Problem, with 2nd Order basis functions,
C this function returns the bounded portion of the integrand
C for loc=2

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 UstarB,RBig,Eta,IsoJacob,Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=(1.0-GaussPt)/2.0
      CALL PlaSub(Eta,2,IsoJacob,Phi,RBig)

      UStarB=-0.5*LOG(RBig)
      Pla2B2 = UStarB*IsoJacob*Phi(2)/2.0

      Eta=(1.0+GaussPt)/2.0
      CALL PlaSub(Eta,2,IsoJacob,Phi,RBig)

      UStarB=-0.5*LOG(RBig)
      Pla2B2 = Pla2B2 + UStarB*IsoJacob*Phi(2)/2.0

      RETURN
      END

C ********************************************************************** Pla2B1

      REAL*8 FUNCTION Pla2B1(GaussPt)

C For an Planar Problem, with 2nd Order basis functions,
C this function returns the bounded portion of the integrand for
C loc=1

      REAL*8 GaussPt
```

```
      INCLUDE 'ECBHDR.INC'
      REAL*8 UstarB,RBig,IsoJacob,Phi(MaxBasisDeg+1)

C From Common ElemR

      CALL PlaSub(GaussPt,1,IsoJacob,Phi,RBig)

      UStarB=-0.5*LOG(RBig)
      Pla2B1 = UStarB*IsoJacob*Phi(1)

      RETURN
      END

C ************************************************************************ Pla2U3

      REAL*8 FUNCTION Pla2U3(GaussPt)

C For an Planar Problem, with 2nd Order basis functions,
C this function returns the unbounded portion of the integrand for
C loc=3 (Note: the unbounded LN(1/GaussPt) term is NOT included.  This
C is accounted for in the integration routine points and weights)

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 RBig,Eta,IsoJacob,Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=1-GaussPt
      CALL PlaSub(Eta,1,IsoJacob,Phi,RBig)

      Pla2U3 = IsoJacob*Phi(3)

      RETURN
      END

C ************************************************************************ Pla2U2

      REAL*8 FUNCTION Pla2U2(GaussPt)

C For an Planar Problem, with 2nd Order basis functions,
C this function returns the unbounded portion of the integrand for
C loc=2 (Note: the unbounded LN(1/GaussPt) term is NOT included.  This
C is accounted for in the integration routine points and weights)

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 RBig,Eta,IsoJacob,Phi(MaxBasisDeg+1)

C From Common ElemR

      Eta=(1.0-GaussPt)/2.0
      CALL PlaSub(Eta,2,IsoJacob,Phi,RBig)

      Pla2U2 = IsoJacob*Phi(2)/2.0
```

```
      Eta=(1.0+GaussPt)/2.0
      CALL PlaSub(Eta,2,IsoJacob,Phi,RBig)

      Pla2U2 = Pla2U2 + IsoJacob*Phi(2)/2.0

      RETURN
      END

C ************************************************************************ Pla2U1

      REAL*8 FUNCTION Pla2U1(GaussPt)

C For an Planar Problem, with 2nd Order basis functions,
C this function returns the unbounded portion of the integrand for
C loc=1 (Note: the unbounded LN(1/GaussPt) term is NOT included.  This
C is accounted for in the integration routine points and weights)

      REAL*8 GaussPt

      INCLUDE 'ECBHDR.INC'
      REAL*8 RBig,IsoJacob,Phi(MaxBasisDeg+1)

C From Common ElemR

      CALL PlaSub(GaussPt,1,IsoJacob,Phi,RBig)

      Pla2U1 = IsoJacob*Phi(1)

      RETURN
      END

C ************************************************************************ PlaSub

      SUBROUTINE PlaSub(GaussPt,Loc,IsoJacob,Phi,RBig)

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBPAR.INC'

C From Common ElemR, SourceR
      INCLUDE 'ECBELM.INC'

      INTEGER Loc
      REAL*8 GaussPt,IsoJacob,Phi(*),RBig

      REAL*8 R(MaxNoOfDim)
      REAL*8 dRdZeta(MaxNoOfDim),PhiZeta(MaxBasisDeg+1)

C Functions

      REAL*8 IsoJacobian
      EXTERNAL IsoJacobian

C From Common BasisDeg, ElemR, SourceR

      CALL Compute1DPhis(BasisDeg,GaussPt,Phi,PhiZeta)
      CALL ComputeRZ(Phi,R)

      IsoJacob=IsoJacobian(PhiZeta,dRdZeta)
```

```
      CALL ComputeRBig(GaussPt,Loc,RBig)

      RETURN
      END

C ********************************************************************** PrepInt

      SUBROUTINE PrepInt(Elem)

C This subroutine prepares variables for use in the integration subroutine.
C Two things are accomplished here: 1) The node coordinates on the element
C are put into ElemR and 2) values that need only be calculated once for
C each element are computed, such as Isojacobian, Normal, ... at each
C gaussian integration point.

C Common Variables

      INCLUDE 'ECBPar.INc'
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECBG.INC'
      INCLUDE 'EcbElm.INC'

C Passed Variables

      INTEGER Elem

C Internal Variables

      INTEGER Gs,Loc,I
      REAL*8 R(MaxNoOfDim),dRdZeta(MaxNoOfDim),
     & IsoJacob,Normal(MaxNoOfDim)
      REAL*8 Phi(MaxBasisDeg+1),PhiZeta(MaxBasisDeg+1)

C Functions

      REAL*8 IsoJacobian
      EXTERNAL IsoJacobian

C Code Follows

      CALL GetElemR(Elem)

C The following DO-Loop computes variables of interest that only need
C to be calculated one time and stored, so that we save computing time
C when in the loop over source points.

      DO Gs=1,NoOfGaussPts
        CALL Compute1DPhis(BasisDeg,GaussPts(Gs),Phi,PhiZeta)
        CALL ComputeRZ(Phi,R)
        IsoJacob=IsoJacobian(PhiZeta,dRdZeta)

C The following two statements compute the outward pointing normal multiplied
C by IsoJacob.

        Normal(1) =+dRdZeta(2)
        Normal(2) =-dRdZeta(1)
```

```
          GIsoJacob(Gs)=IsoJacob
          DO I=1,NoOfDim
            GNormal(Gs,I)=Normal(I)
            GR(Gs,I)=R(I)
          END DO
          DO I=1,BasisDeg+1
            GPhi(Gs,I)=Phi(I)
          END DO
        END DO

        RETURN
        END

C ******************************************************************** PrintMesh

        SUBROUTINE PrintMesh(Lun,Msg)

        INTEGER Lun,I,K
        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMsh.INC'
        INCLUDE 'ECBBC.INC'
        LOGICAL LastNat
        CHARACTER*(*) Msg

1       FORMAT(X,80('*'))
2       FORMAT(X,A40,29X,'Mesh Report')
10      FORMAT(X,'  Boundary Conditions',12X,'Coordinates',
    & /,X,' Nat  BC Value  Sol Value',5X,'i',7X,'X',9X,'Y',
    & /)
11      FORMAT(X,2X,A1,2X,G10.4,X,G10.4,3X,I3,2X,G10.4,X,G10.4)

C123                 123456789a12
C   Boundary Conditions          Coordinates
C1  12        12         12345 1234567 123456789
C Nat  BC Value  Sol Value    i       X        Y
C12 12                    123   12
C  a  ddddddddddd dddddddddd   iii  ddddddddddd dddddddddd
C     123456789a 123456789a   123  123456789a 123456789a

        WRITE(Lun,1)
        WRITE(Lun,2) Msg
        WRITE(Lun,1)
        WRITE(Lun,*) ' '
        WRITE(Lun,10)
        LastNat=Natural(1)
        DO I=1,NoOfCoords
            IF (LastNat.NE.Natural(I)) WRITE(Lun,*) ' '
            LastNat=Natural(I)

            IF (Natural(I)) THEN
                WRITE(Lun,11) 'T',BCValues(I),
    & SolValues(I),I,
    & Coords(I,1),Coords(I,2)
            ELSE
                WRITE(Lun,11) 'F',BCValues(I),
    & SolValues(I),I,
    & Coords(I,1),Coords(I,2)
            END IF
```

```
      END DO

      RETURN
      END

C ******************************************************************* PrintNodeMap

      SUBROUTINE PrintNodeMap(Lun)

      INTEGER Lun,I,K
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBBC.INC'

C       123
C Element   Nodes
C  iii      iii,iii,iii,iii,...
C     123456
20    FORMAT(/,X,'Element   Nodes'/)
30    FORMAT(X,X,I3,6X,I3,10(:,',',I3))

      WRITE(Lun,20)
      DO I=1,NoOfElems
        WRITE(Lun,30) I,(NodeMap(I,K),K=1,BasisDeg+1)
      END DO

      RETURN
      END

C ******************************************************************** PrintSrc

      SUBROUTINE PrintSrc(Lun)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBSrc.INC'

      INTEGER Lun,I

C S O U R C E   P O I N T S
C 1 1234567 123456789a
C  i      X          Y
C iii dddddddddd dddddddddd
C 123 123456789a 123456789a

10    FORMAT(X,'S O U R C E   P O I N T S',
     & /,X,' i',7X,'X',10X,'Y',
     & /)
20    FORMAT(X,I3,X,G10.4,X,G10.4)

      WRITE(Lun,10)
      DO I=1,NoOfSrc
        WRITE(Lun,20) I,Src(I,1),Src(I,2)
      END DO

      RETURN
      END

C ********************************************************************** QStar
```

```
      REAL*8 FUNCTION QStar(Dist2,Gs,Den,M)

      INCLUDE 'ECBPar.INc'

C The following two include files are necessary for dimensioning items in
C the following third include file.
C QStar is multiplied by 2Pi

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECBG.INC'
      INCLUDE 'ECBElm.INC'

      INTEGER Gs
      REAL*8 Dist2,Den,M
      REAL*8 T1,T2,T3,First,Second

      REAL*8 KElliptic,EElliptic
      EXTERNAL KElliptic,EElliptic

      IF (Axisymmetric) THEN

C The problem is an Axisymmetric.
C QStar is multiplied by 2Pi

        T1 = KElliptic(M)

        T2 = EElliptic(M)*
     & (SourceR(1)*SourceR(1)-GR(Gs,1)*GR(Gs,1)
     & + (Gr(Gs,2)-SourceR(2))*(Gr(Gs,2)-SourceR(2)))
        T2 = T2/( (GR(Gs,1)-SourceR(1))*(GR(Gs,1)-SourceR(1))
     & + (Gr(Gs,2)-SourceR(2))*(Gr(Gs,2)-SourceR(2)) )

        T3 = 2.0*EElliptic(M) * (SourceR(2)-Gr(Gs,2))
     & / ( (GR(Gs,1)-SourceR(1))*(GR(Gs,1)-SourceR(1))
     &   + (Gr(Gs,2)-SourceR(2))*(Gr(Gs,2)-SourceR(2)) )

        QStar = ( (-T1+T2) * GNormal(Gs,1)
     &  + T3 * GNormal(Gs,2) * Gr(Gs,1) )/SQRT(Den)

      ELSE

C The problem is planar
C QStar is multiplied by 2Pi

        QStar=( (SourceR(1)-GR(Gs,1)) * GNormal(Gs,1) +
     & (SourceR(2)-GR(Gs,2)) * GNormal(Gs,2) ) / (Dist2)

      END IF

      RETURN
      END

C ************************************************************************ ReadCal

      SUBROUTINE ReadCal(FileName)

      INCLUDE 'ECBHdr.INC'
```

```
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      CHARACTER*(*) FileName
      INTEGER I,J,Version
      LOGICAL FileExist

      INQUIRE(FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) CALL ErrorFileName(FileName,'ReadCalcs')

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      READ(1,*) Version

      IF (Version.EQ.1) THEN
        DO I=1,NoOfCoords
          DO J=1,NoOfCoords
            READ(1,*) BemG(I,J),BemH(I,J)
          END DO
        END DO
      ELSE
        CALL ErrorVersion(Version,FileName,'ReadCal')
      END IF

      CLOSE(1)

      RETURN
      END

C ************************************************************************ ReadInt

      SUBROUTINE ReadInt(FileName)

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBINT.INC'

      CHARACTER*(*) FileName
      LOGICAL FileExist
      INTEGER Version,I,K

      INQUIRE(FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) CALL ErrorFileName(FileName,'ReadInt')

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')
      READ(1,*) Version

      IF (Version.EQ.1) THEN
        READ(1,*) NoOfInternal,NoOfSpines,IntPerSpine
        DO I=1,NoOfInternal
          READ(1,*) (InternalPts(I,K),K=1,NoOfDim),InternalPot(I)
        END DO
      ELSE IF (Version.EQ.0) THEN
        READ(1,*) NoOfInternal,NoOfSpines,IntPerSpine
        DO I=1,NoOfInternal
          READ(1,*) (InternalPts(I,K),K=1,NoOfDim)
        END DO
      ELSE
        CALL ErrorVersion(Version,FileName,'ReadInt')
      END IF
```

```
      CLOSE(1)

      RETURN
      END

C ********************************************************************** ReadMac

      SUBROUTINE ReadMac(FileName)

C This subroutine reads the contents of the BEM.MAC file.  This file contains
C the information required to make the BEM.INP file, the more detailed file
C used by the BEMLIN.FOR program for input.

      INCLUDE 'ECBMAC.INC'
      CHARACTER*(*) FileName
      INTEGER I,Version,FLen
      LOGICAL FileExist

      INQUIRE (FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) CALL ErrorFileName(FileName,'ReadMac')

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      READ(1,*) Version
      IF ((Version.NE.1).AND.(Version.NE.2).AND.(Version.NE.3))
     & CALL ErrorVersion(Version,FileName,'ReadMac')

      READ(1,*) NoOfMacVert
      IF (NoOfMacVert.GT.MaxNoOfMacVert) THEN
        WRITE(*,*) 'Error: Number of macro-vertices,',NoOfMacVert
        WRITE(*,*) '       is larger than the maximum number of '
        WRITE(*,*) '       macro-vertices allowed:',MaxNoOfMacVert
        STOP 'In Routine: ReadMac'
      END IF

      CALL StringLength(FileName,FLen)
D     WRITE(*,*) 'Reading File ',FileName(1:FLen),' Version:',Version
D     WRITE(*,*) '             ',NoOfMacVert,' macro vertices'
      DO I=1,NoOfMacVert
        READ(1,*) MacVertX(I),MacVertY(I)
      END DO

      READ(1,*) NoOfMacElems
D     WRITE(*,*) '             ',NoOfMacElems,' macro elems'
      DO I=1,NoOfMacElems
        READ(1,*) MacShape(I),NoOfMicElems(I),MacBCType(I),MacBCVal(I),
     & MacSpac(I),MacRatio(I)
        IF (MacShape(I).EQ.'LINE') THEN
          READ(1,*) MacIndx(I,1),MacIndx(I,2)
        ELSE IF (MacShape(I).EQ.'SINE') THEN
          READ(1,*) MacIndx(I,1),MacIndx(I,2)
          READ(1,*) MacPar(I,1),MacPar(I,2)

C (Y-Y0)=A*Sin((Par1*X+Par2)*Pi) A and Y0 are solved for

        ELSE IF (MacShape(I).EQ.'CIR') THEN
          READ(1,*) MacIndx(I,1),MacIndx(I,2)
          READ(1,*) MacPar(I,1),MacPar(I,2)
```

```
C CIR, Radius is determined from center and starting mac vertex #

          ELSE IF ((MacShape(I).EQ.'USER').OR.
     & (MacShape(I).EQ.'USR')) THEN
            IF (Version.LE.2) THEN
              READ(1,*) MacIndx(I,1),MacIndx(I,2)
              MacBdryFile='USER'
            ELSE
              READ(1,*) MacIndx(I,1),MacIndx(I,2),MacBdryFile
            END IF
          ELSE IF (MacShape(I).EQ.'TRNS') THEN
            READ(1,*) MacIndx(I,1),MacIndx(I,2)
          ELSE IF (MacShape(I).EQ.'PREV') THEN
            READ(1,*) MacIndx(I,1),MacIndx(I,2)
            READ(1,*) MacPar(I,1),MacPar(I,2),MacPar(I,3),MacPar(I,4)
          ELSE IF (MacShape(I).EQ.'FLAT') THEN
            READ(1,*) MacIndx(I,1),MacIndx(I,2)
            READ(1,*) MacPar(I,1),MacPar(I,2)
C                 BL thickness, Element to space away from
          ELSE IF (MacShape(I).EQ.'FLTL') THEN
            READ(1,*) MacIndx(I,1),MacIndx(I,2)
            READ(1,*) MacPar(I,1),MacPar(I,2)
C                 BL thickness, Element to space away from
          ELSE
            WRITE(*,*) 'Error: Unknown Macro-Element Type:',MacShape(I)
            STOP 'In Routine: ReadMac'
          END IF
          IF (Version.GE.2) READ(1,*) MacStart(I),MacEnd(I)
          IF (MacSpac(I).EQ.'USER') READ(1,*) MacSpaFile(I)
        END DO
        CLOSE(1)
D       WRITE(*,*) ' '

        RETURN
        END

C ********************************************************************** ReadMesh

        SUBROUTINE ReadMesh(FileName)

        CHARACTER*(*) FileName
        INTEGER Length,Version,I,J
        LOGICAL FileExist

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMsh.INC'
        INCLUDE 'ECBBC.Inc'

        INQUIRE(FILE=FileName,EXIST=FileExist)

        IF (.NOT.FileExist) CALL ErrorFileName(FileName,'ReadMesh')

        OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

        READ(1,*) Version

        IF ((Version.EQ.1).OR.(Version.EQ.2)) THEN
```

```
      READ(1,*) NoOfCoords
      DO I=1,NoOfCoords
        READ(1,*) (Coords(I,J),J=1,NoOfDim)
        READ(1,*) BCValues(I),SolValues(I),Natural(I),BCType(I)
      END DO

      READ(1,*) NoOfElems
      DO I=1,NoOfElems
        READ(1,*) (NodeMap(I,J),J=1,BasisDeg+1),ElemNat(I),
     & ElemBCType(I)
      END DO

      IF (Version.EQ.1) THEN
        DO I=1,NoOfElems
          DO J=1,BasisDeg+1
            Natural(NodeMap(I,J))=ElemNat(I)
            BCType(NodeMap(I,J))=ElemBCType(I)
          END DO
        END DO
      END IF

    ELSE

      CALL ErrorVersion(Version,FileName,'ReadMesh')

    END IF
    CLOSE(1)

    RETURN
    END

C ************************************************************************ ReadPar

    SUBROUTINE ReadPar(FileName)

    INCLUDE 'ECBHdr.INC'
    INCLUDE 'ECBPar.INC'
    INCLUDE 'ECGaus.INC'

    CHARACTER*(*) FileName
    LOGICAL FileExist
    INTEGER Length,I,Version
    REAL*8 JohnNewman,IRatio

    INQUIRE(FILE=FileName,EXIST=FileExist)
    IF (.NOT.FileExist) CALL ErrorFileName(FileName,'ReadPar')

    OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')
    READ(1,*) Version

    CorLev=.FALSE.
    Spline3D=.FALSE.
    MaxIter=25

    IF (Version.EQ.1) THEN
      READ(1,*) NoOfDim,BasisDeg,NoOfGaussPts
      READ(1,*) SrcFactor,Tolerance
      READ(1,*) Axisymmetric
```

```
      READ(1,*) JohnNewman,IRatio
      READ(1,*) aA,aC
      MaxIter=25
      WagLin= 1/ (JohnNewman*IRatio)
    ELSE IF (Version.EQ.2) THEN
      READ(1,*) NoOfDim,BasisDeg,NoOfGaussPts
      READ(1,*) SrcFactor,Tolerance,MaxIter
      READ(1,*) Axisymmetric
      READ(1,*) JohnNewman,IRatio
      READ(1,*) aA,aC
      WagLin= 1/ (JohnNewman*IRatio)
    ELSE IF (Version.EQ.3) THEN
      READ(1,*) NoOfDim,BasisDeg,NoOfGaussPts
      READ(1,*) Spline3D
      READ(1,*) SrcFactor,Tolerance,MaxIter
      READ(1,*) Axisymmetric
      READ(1,*) JohnNewman,IRatio
      READ(1,*) aA,aC
      WagLin= 1/ (JohnNewman*IRatio)
    ELSE IF (Version.EQ.4) THEN
      READ(1,*) NoOfDim,BasisDeg,NoOfGaussPts
      READ(1,*) Spline3D
      READ(1,*) SrcFactor,Tolerance,MaxIter
      READ(1,*) Axisymmetric
      READ(1,*) JohnNewman,IRatio
      READ(1,*) aA,aC
      READ(1,*) KineticsFile
      WagLin= 1/ (JohnNewman*IRatio)
    ELSE IF (Version.EQ.5) THEN
      READ(1,*) NoOfDim,BasisDeg,NoOfGaussPts
      READ(1,*) Spline3D, Axisymmetric
      READ(1,*) CorLev, PerRev
      READ(1,*) Corros, CRRatio,WaLinPCR
      READ(1,*) SrcFactor,Tolerance,MaxIter
      READ(1,*) WagLin
      READ(1,*) aA,aC
      READ(1,*) KineticsFile
      READ(1,*) tstar,istar
    ELSE IF (Version.EQ.6) THEN
      READ(1,*) NoOfDim,BasisDeg,NoOfGaussPts
      READ(1,*) Spline3D, Axisymmetric
      READ(1,*) CorLev, PerRev
      READ(1,*) Corros
      READ(1,*) NoOfPCR,CRRatio,WaLinPCR,PCRMax
      READ(1,*) SrcFactor,Tolerance,MaxIter
      READ(1,*) WagLin
      READ(1,*) aA,aC
      READ(1,*) KineticsFile
      READ(1,*) tstar,istar
    ELSE
      CALL ErrorVersion(Version,FileName,'ReadPar')
    END IF

    CLOSE(1)

    RETURN
    END
```

```
C ********************************************************************** ReadSrc

      SUBROUTINE ReadSrc(FileName)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBSrc.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER Version,I,J
      CHARACTER*(*) FileName
      LOGICAL FileExist

      INQUIRE(FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) CALL ErrorFileName(FileName,'ReadSrc')

      OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')

      READ(1,*) Version

      IF (Version.EQ.1) THEN
        READ(1,*) NoOfSrc

        IF (NoOfSrc.NE.NoOfCoords) THEN
          WRITE(*,*) 'Error: Number of source points is not equal to'
          WRITE(*,*) '       the number of coordinates.'
          WRITE(*,*) '      Filename=',FileName
          WRITE(*,*) '       NoOfSrc=',NoOfSrc
          WRITE(*,*) '    NoOfCoords=',NoOfCoords
          STOP 'In Routine: ReadSrc'
        END IF

        DO I=1,NoOfSrc
          READ(1,*) (Src(I,J),J=1,NoOfDim)
        END DO
      ELSE
        CALL ErrorVersion(Version,FileName,'ReadSrc')
      END IF
      CLOSE(1)

      RETURN
      END

C ********************************************************************** ReMesh

      SUBROUTINE ReMesh(NoOfSpa,Spa,NoOfPos,Pos)

C This subroutine remeshes the nodal locations.

C Input Variables

C NoOfSpa      The desired number of Spacings
C Spa()        The original Spacing
C NoOfPos      The number of original spacings and positions Pos(i,1..2)
C Pos(,2)      The original values of the function whose curvature is
C              equidistributed.

C Output Variables
```

```
C Spa() The relocated Spacings

C Passed Variables

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBRemesh.INC'
      REAL*8 Spa(*),Pos(MaxNoOfCoords,MaxNoOfDim)
      INTEGER NoOfSpa,NoOfPos

C Local Variables

      INTEGER I,J,K
      REAL*8 Curvature(MaxNoOfCoords),TSpa,LowerBound,UpperBound,
     & PosX(MaxNoOfCoords),PosY(MaxNoOfCoords)
      REAL*8 Error(MaxNoOfCoords),AvgError,Sum,YMax,YMin
      REAL*8 XCSCoef(4,MaxNoOfCoords),XBreak(MaxNoOfCoords)
      REAL*8 YCSCoef(4,MaxNoOfCoords),YBreak(MaxNoOfCoords)
      REAL*8 Orig(MaxNoOfCoords),Slope
      LOGICAL Changes,FileRead/.FALSE./

C Functions

      REAL*8 DCSDER,DCSITG,DCSVAL
      EXTERNAL DCSDER,DCSITG,DCSVAL

      IF (.NOT.FileRead) THEN
        OPEN(UNIT=2,FILE='REMESH.ECB',STATUS='OLD',FORM='FORMATTED')
        READ(2,*) SpaFac
        READ(2,*) UpFac,LowFac
        CLOSE(2)
        FileRead=.TRUE.
      END IF

      DO I=1,NoOfPos
        Orig(I)=Spa(I)
        PosX(I)=Pos(I,1)
        PosY(I)=Pos(I,2)
      END DO

D     WRITE(*,*) 'Calling DCSSCV from Remesh: x'
      CALL DCSSCV(NoOfPos,Spa,PosX,2,XBreak,XCSCoef)
D     WRITE(*,*) 'Calling DCSSCV from Remesh: y'
      CALL DCSSCV(NoOfPos,Spa,PosY,2,YBreak,YCSCoef)

C Compute the Curvature

      DO I=1,NoOfPos
        Curvature(I)=SQRT(
     & DCSDER(2,Spa(I),NoOfPos-1,XBreak,XCSCoef)**2 +
     & DCSDER(2,Spa(I),NoOfPos-1,YBreak,YCSCoef)**2)
      END DO

C Bound the Curvature

      Sum=0.0
      DO I=1,NoOfPos-1
        Sum=Sum+(Spa(I+1)-Spa(I))*
     & (Curvature(I+1)+Curvature(I))/2.0D0
```

```
      END DO

      AvgError=Sum/(NoOfPos-1)

      UpperBound=AvgError*UpFac
      LowerBound=AvgError*LowFac

      DO I=1,NoOfPos
        Curvature(I)=MIN(UpperBound,Curvature(I))
        Curvature(I)=MAX(LowerBound,Curvature(I))
      END DO

C Compute New Integral Error

D     WRITE(*,*) 'Error'
D     WRITE(*,*) NoOfPos
D     WRITE(*,*) 1,0.0D0,Spa(1)

      Error(1)=0.0
      DO I=2,NoOfPos
        Error(I)=Error(I-1)+(Spa(I)-Spa(I-1))*
     & (Curvature(I)+Curvature(I-1))/2.0D0
D       WRITE(*,*) I,Error(I),Spa(I)
      END DO
D     WRITE(*,*) ' '

      AvgError=Error(NoOfPos)/FLOAT(NoOfSpa-1)

C Now equi-distribute the residual error for each element.

C     Spa(1) keeps its original value.
C     Spa(NoOfSpa) keeps the endpoint value

      Spa(NoOfSpa)=Spa(NoOfPos)

      TSpa=0.0
      K=1
      DO I=2,NoOfSpa-1
        TSpa=TSpa+AvgError
        DO WHILE (TSpa.GT.Error(K+1))
          K=K+1
D         WRITE(*,*) '  Error:',K,Error(K),Orig(K)
        END DO
        Spa(I) = Orig(K) + (TSpa-Error(K)) *
     & (Orig(K+1)-Orig(K))/
     &  (Error(K+1)-Error(K))
D       WRITE(*,*) 'Location',I,TSpa,Spa(I)
      END DO

      RETURN
      END

C ************************************************************** RemoveBdryAfter

      SUBROUTINE RemoveBdryAfter(NoOfMoved,Moved,I,XInt,YInt)

C Remove points between intersection (I,) and end from moved surface.
C The segment from Moved(I,) to Moved(I+1,) intersects with insulator.
```

```
C Moved(I)=intersection

      INCLUDE 'ECBHdr.INC'
      REAL*8 Moved(MaxNoOfCoords,MaxNoOfDim),XInt,YInt
      INTEGER I,NoOfMoved

D     WRITE(*,*) '---------: RemoveBdryAfter',I

      Moved(I,1)=XInt
      Moved(I,2)=YInt

      NoOfMoved=I

      RETURN
      END

C ************************************************************* RemoveBdryBefore

      SUBROUTINE RemoveBdryBefore(NoOfMoved,Moved,I,XInt,YInt)

C Remove points between start and intersection (1,I) from moved surface.
C The segment from Moved(I,) to Moved(I+1,) intersects with insulator.
C Moved(1)=intersection
C Moved(2)=Moved(I+1),...

      INCLUDE 'ECBHdr.INC'
      REAL*8 Moved(MaxNoOfCoords,MaxNoOfDim),XInt,YInt
      INTEGER I,NoOfMoved
      INTEGER K

D     WRITE(*,*) '---------: RemoveBdryBefore',I
      Moved(1,1)=XInt
      Moved(1,2)=YInt

      IF (I.EQ.1) RETURN

      DO K=I+1,NoOfMoved
        Moved(K-I+1,1)=Moved(K,1)
        Moved(K-I+1,2)=Moved(K,2)
      END DO

      NoOfMoved=NoOfMoved-I+1

      RETURN
      END

C ************************************************************* RemoveBdryIntersects

      SUBROUTINE RemoveBdryIntersects

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER I,J,K,Error
      LOGICAL Thinking
      REAL*8 XInt,YInt,Dist
```

```
D       WRITE(*,*) '---------: RemoveBdryIntersects'
        I=2
        Thinking=(I.LT.NoOfMoved)
        DO WHILE(Thinking)

C Check following line segments for intersections with current line segment

          J=I+2
          IF (J.GT.NoOfMoved) THEN
            Error=0
          ELSE
            Error=1
            Dist=SQRT((Moved(I,1)-Moved(J-1,1))**2+
     & (Moved(I,2)-Moved(J-1,2))**2)
            IF (Dist.LT.Tolerance*10.0D0) THEN
              Error=0
              XInt=(Moved(I,1)+Moved(J-1,1))/2.0D0
              YInt=(Moved(I,2)+Moved(J-1,2))/2.0D0
            END IF
          END IF

          DO WHILE(Error.NE.0)
            CALL ComputeIntersection(Moved(I-1,1),Moved(I-1,2),
     & Moved(I,1),Moved(I,2),
     & Moved(J-1,1),Moved(J-1,2),
     & Moved(J,1),Moved(J,2),XInt,YInt,Error)
            IF (Error.NE.0) THEN
              J=J+1
              IF (J.GT.NoOfMoved) Error=0
            END IF
          END DO

          IF (J.LE.NoOfMoved) THEN

C Insert intersection point between node I-1 and node J.

            Moved(I,1)=XInt
            Moved(I,2)=YInt

C Remove nodes between I and J-1.

            DO K=J,NoOfMoved
              Moved(I+K+1-J,1)=Moved(K,1)
              Moved(I+K+1-J,2)=Moved(K,2)
            END DO
            NoOfMoved=NoOfMoved-(J-I)+1

C Recheck for intersections beginning with node I

          ELSE

C No intersection found with segment I, increment I to check for next segment.

            I=I+1

          END IF

          Thinking=(I.LT.NoOfMoved)
```

```
      END DO

      RETURN
      END

C ************************************************************ RemoveInsIntersects

      SUBROUTINE RemoveInsIntersects

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMOVE.INC'

      REAL*8 XInt,YInt,R,R2
      INTEGER I,J,K
      LOGICAL Thinking,Error

D     WRITE(*,*) '---------: RemoveInsIntersects'

C First check one end of boundary ---------------------------------------------

      IF (AcuteAngleBefore) THEN

C If Acute angle with insulator, then loop from start until intersection with
C conductor segment and insulator is found.

C Check for intersection between boundary and insulator.

        CALL LocateIntersects(1,I,J,XInt,YInt,Error)

        IF (.NOT.Error) THEN
          IF (XInt.EQ.Moved(I+1,1).AND.(XInt.EQ.Moved(I+1,2))) I=I+1

          CALL RemoveBdryBefore(NoOfMoved,Moved,I,XInt,YInt)
          CALL RemoveBdryBefore(NoOfIns,Ins,J,XInt,YInt)

          Last=Last-J+1
          NextToLast=NextToLast-J+1

        ELSE
          WRITE(*,*) 'Warning: Treating nonintersection like',
     & ' obtuse angle.'

C Use clockwise circle for forward boundary movement,
C Counterclockwise circle for backward boundary movement.

          CALL ObtAngAtBdryBeg
        END IF

      ELSE IF (.NOT.NoAngleBefore) THEN

C Compute new point on boundary, add to beginning.

        CALL ObtAngAtBdryBeg

      END IF

C Second, check other end of boundary (Similar to Above Code) -----------------
```

```
      IF (AcuteAngleAfter) THEN

C Check for intersection between boundary and insulator.

        CALL LocateIntersects(-1,I,J,XInt,YInt,Error)

        IF (.NOT.Error) THEN
D         WRITE(*,*) 'Calling RemoveBdryAfter from RemoveInsIntersects'
D         WRITE(*,*) '        I=',I,'  J=',J
D         WRITE(*,*) '        X=',XInt,'   Y=',YInt
          CALL RemoveBdryAfter(NoOfMoved,Moved,I,XInt,YInt)
          CALL RemoveBdryAfter(NoOfIns,Ins,J,XInt,YInt)
        ELSE
          WRITE(*,*) 'Warning: Treating nonintersection like',
     & ' obtuse angle.'
          CALL ObtAngAtBdryEnd
        END IF

      ELSE IF (.NOT.NoAngleAfter) THEN

        CALL ObtAngAtBdryEnd

      END IF

D     WRITE(*,*) '          :--------'

      RETURN
      END

C ************************************************************ RemoveLocIntersects

      SUBROUTINE RemoveLocIntersects

C This subroutine averages the beginning endpoint with previous segments
C ending endpoint after a boundary has been moved.

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER I,J,OffSet
      REAL*8 Avg

D     WRITE(*,*) '---------: RemoveLocIntersects'
      DO I=BasisDeg+1,NoOfMoved-1,BasisDeg+1
        DO J=1,NoOfDim
          Avg=(Moved(I,J)+Moved(I+1,J))/2.0D0
          Moved(I,J)=Avg
          Moved(I+1,J)=Avg
        END DO
      END DO

C Remove Duplicate Points

      Offset=0
      DO I=2,NoOfMoved
        IF ((Moved(I,1).EQ.Moved(I-Offset-1,1)).AND.
```

```
     & (Moved(I,2).EQ.Moved(I-Offset-1,2))) THEN
          Offset=Offset+1
        ELSE
          Moved(I-Offset,1)=Moved(I,1)
          Moved(I-Offset,2)=Moved(I,2)
        END IF
      END DO
      NoOfMoved=NoOfMoved-Offset
D     WRITE(*,*) '          ',Offset,'points removed :---------'

      RETURN
      END

C ************************************************************************ ShpCir

      SUBROUTINE ShpCir(MacElm,SaveKinetic)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE '[KJ.FOR.UTIL]Pi.INC'

      INTEGER MacElm,Error,I,NoOfSpa
      REAL*8 X,Y,Xc,Yc,Radius,AngleStart,AngleEnd,Spa(MaxNoOfCoords)
      LOGICAL SaveKinetic

C Get center of circle.

      Xc=MacVertX(MacPar(MacElm,1))
      Yc=MacVertY(MacPar(MacElm,1))

C Compute the radius

      Radius=SQRT(
     & (Xc-MacVertX(MacIndx(MacElm,1)))
     &  * (Xc-MacVertX(MacIndx(MacElm,1))) +
     & (Yc-MacVertY(MacIndx(MacElm,1)))
     &  * (Yc-MacVertY(MacIndx(MacElm,1))) )

C Locate Start and End Angles

      CALL XY2T(MacVertX(MacIndx(MacElm,1))-Xc,
     & MacVertY(MacIndx(MacElm,1))-Yc,AngleStart)
      CALL XY2T(MacVertX(MacIndx(MacElm,2))-Xc,
     & MacVertY(MacIndx(MacElm,2))-Yc,AngleEnd)

      IF (MacPar(Macelm,2).GE.0) THEN

C Clockwise Circle

        DO WHILE(AngleStart.LE.AngleEnd)
          AngleStart=AngleStart+2.0*Pi
        END DO
      ELSE

C Counter-Clockwise Circle

        DO WHILE(AngleStart.GE.AngleEnd)
```

```
          AngleStart=AngleStart-2.0*Pi
        END DO
      END IF

      CALL ShpSpacing(MacElm,NoOfSpa,Spa)
      DO I=1,NoOfSpa
        X=Xc+Radius*COS(AngleStart+(AngleEnd-AngleStart)*Spa(I))
        Y=Yc+Radius*SIN(AngleStart+(AngleEnd-AngleStart)*Spa(I))
        CALL MakeCoord(X,Y,MacStart(MacElm)+(I-1),MacElm,SaveKinetic)
      END DO

      RETURN
      END

C ********************************************************************** ShpFlat

      SUBROUTINE ShpFlat(MacElm,SaveKinetic)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER MacElm,NoOfSpa,I
      LOGICAL SaveKinetic
      REAL*8 X,Y,Spa(MaxNoOfCoords),MaxY

C Note: Must Alter Endpoints for Start and End (Note: No TRNS elements!)

      CALL ShpFlatHiPt(MacElm,MaxY)
      Y=MaxY+MacPar(MacElm,1)
      MacVertY(MacIndx(MacElm,2))=Y

      CALL ShpSpacing(MacElm,NoOfSpa,Spa)
      DO I=1,NoOfSpa
        X=MacVertX(MacIndx(MacElm,1)) +
   & (MacVertX(MacIndx(MacElm,2)) - MacVertX(MacIndx(MacElm,1)))
   & * Spa(I)
        CALL MakeCoord(X,Y,MacStart(MacElm)+(I-1),MacElm,SaveKinetic)
      END DO

      RETURN
      END

C ******************************************************************** ShpFlatHiPt

      SUBROUTINE ShpFlatHiPt(MacElm,MaxY)

C This subroutine locates the high (maximum Y) point on macro element
C MacPar(MacElm,2)

C This subroutine is used for FLAT mass transfer boundary layers located
C above a profile below.

      INTEGER MacElm
      REAL*8 MaxY

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
```

```
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER PrevMac,PrevElems,MicNod,K

      PrevMac=MacPar(MacElm,2)
      CALL ComputePrevElem(PrevMac,PrevElems)

      MaxY=Coords(NodeMap(PrevElems+1,1),2)
      DO MicNod=PrevElems+1,PrevElems+NoOfMicElems(PrevMac)
        DO K=1,BasisDeg+1
          IF (MaxY.LT.Coords(NodeMap(MicNod,K),2))
     & MaxY=Coords(NodeMap(MicNod,K),2)
        END DO
      END DO

      RETURN
      END

C ***************************************************************** ShpFlatLoPt

      SUBROUTINE ShpFlatLoPt(MacElm,MinY)

C This subroutine locates the low (minimum Y) point on macro element
C MacPar(MacElm,2)

C This subroutine is used for FLTL mass transfer boundary layers located
C above a profile below.

      INTEGER MacElm
      REAL*8 MinY

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER PrevMac,PrevElems,MicNod,K

      PrevMac=MacPar(MacElm,2)
      CALL ComputePrevElem(PrevMac,PrevElems)

      MinY=Coords(NodeMap(PrevElems+1,1),2)
      DO MicNod=PrevElems+1,PrevElems+NoOfMicElems(PrevMac)
        DO K=1,BasisDeg+1
          IF (MinY.GT.Coords(NodeMap(MicNod,K),2))
     & MinY=Coords(NodeMap(MicNod,K),2)
        END DO
      END DO

      RETURN
      END

C ******************************************************************** ShpFltL

      SUBROUTINE ShpFltL(MacElm,SaveKinetic)

      INCLUDE 'ECBHdr.INC'
```

```
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER MacElm,NoOfSpa,I
      LOGICAL SaveKinetic
      REAL*8 X,Y,Spa(MaxNoOfCoords),MinY

C Note: Must Alter Endpoints for Start and End (Note: No TRNS elements!)

      CALL ShpFlatLoPt(MacElm,MinY)
      Y=MinY+MacPar(MacElm,1)
      MacVertY(MacIndx(MacElm,2))=Y

      CALL ShpSpacing(MacElm,NoOfSpa,Spa)
      DO I=1,NoOfSpa
        X=MacVertX(MacIndx(MacElm,1)) +
     & (MacVertX(MacIndx(MacElm,2)) - MacVertX(MacIndx(MacElm,1)))
     & * Spa(I)
        CALL MakeCoord(X,Y,MacStart(MacElm)+(I-1),MacElm,SaveKinetic)
      END DO

      RETURN
      END

C ********************************************************************** ShpLine

      SUBROUTINE ShpLine(MacElm,SaveKinetic)

C This subroutine generates the mesh points for line Shps. (Straight
C line segments between macro mesh pts)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER MacElm,NoOfSpa,I
      REAL*8 X,Y,Spa(MaxNoOfCoords)
      LOGICAL SaveKinetic

      CALL ShpSpacing(MacElm,NoOfSpa,Spa)
      DO I=1,NoOfSpa
        X=MacVertX(MacIndx(MacElm,1)) +
     & ( MacVertX(MacIndx(MacElm,2))
     &   - MacVertX(MacIndx(MacElm,1)) )
     & * Spa(I)
        Y=MacVertY(MacIndx(MacElm,1)) +
     & (MacVertY(MacIndx(MacElm,2))-MacVertY(MacIndx(MacElm,1)))
     & * Spa(I)
        CALL MakeCoord(X,Y,MacStart(MacElm)+(I-1),MacElm,SaveKinetic)
      END DO

      RETURN
      END

C ********************************************************************** ShpPrev

      SUBROUTINE ShpPrev(MacElm,SaveKinetic)
```

```
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INc'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBMOVE.INC'

      INTEGER MacElm,MicNod,NoOfSpa,PrevElems,
     & ElemStart,ElemEnd,MacFollow,J,I
      LOGICAL SaveKinetic,Smooth
      REAL*8 X,Y,Spa(MaxNoOfCoords),D,Temp
      REAL*8 dtime/1.0/

C Get nodes of previous element

      MacFollow=MacPar(MacElm,2)
      CALL ComputePrevElem(MacFollow,PrevElems)

      ElemStart=PrevElems+1
      ElemEnd=PrevElems+NoOfMicElems(MacFollow)
D     WRITE(*,*) '     ShpPrev: ElemStart=',ElemStart,
D    & '     ElemEnd=',ElemEnd

D     WRITE(*,*) '     ShpPrev: Calling MakeMoved'
      CALL MakeMoved(ElemStart,ElemEnd,NodeMap,Coords)
D     WRITE(*,*) '     ShpPrev: NoOfMoved=',NoOfMoved

D     WRITE(*,*) '     ShpPrev: First Moved Point=',
D    & Moved(1,1),Moved(1,2)
D     WRITE(*,*) '     ShpPrev: Last Moved Point=',
D    & Moved(NoOfMoved,1),Moved(NoOfMoved,2)

      DO MicNod=1,NoOfMoved
        dXdN(MicNod)=-MacPar(MacElm,1)
      END DO
D     WRITE(*,*) '     ShpPrev: Calling ComputeNx'
      CALL ComputeNx(ElemEnd-ElemStart+1)

C Move Previous Elements Nodes by boundary layer thickness

      NoOfIns=4

      Ins(1,1)=Moved(1,1)
      Ins(1,2)=Moved(1,2)

      Ins(2,1)=MacVertX(MacPar(MacElm,3))
      Ins(2,2)=MacVertY(MacPar(MacElm,3))

      Ins(3,1)=MacVertX(MacPar(MacElm,4))
      Ins(3,2)=MacVertY(MacPar(MacElm,4))

      Ins(4,1)=Moved(NoOfMoved,1)
      Ins(4,2)=Moved(NoOfMoved,2)

D     WRITE(*,*) '     ShpPrev: Calling Moved'
      CALL Move(dTime)
D     WRITE(*,*) '     ShpPrev: Last Moved Point=',
D    & Moved(NoOfMoved,1),Moved(NoOfMoved,2)
D     WRITE(*,*) '     ShpPrev: Calling RemoveLocIntersects'
```

```
      CALL RemoveLocIntersects
D     WRITE(*,*) '     ShpPrev: Last Moved Point=',
D   & Moved(NoOfMoved,1),Moved(NoOfMoved,2)
D     WRITE(*,*) '     ShpPrev: Calling RemoveBdryIntersects'
      CALL RemoveBdryIntersects
D     WRITE(*,*) '     ShpPrev: Last Moved Point=',
D   & Moved(NoOfMoved,1),Moved(NoOfMoved,2)

      StaRev=.FALSE.
      EndRev=.FALSE.

      First=1
      Second=2

      NextToLast=3
      Last=4

D     WRITE(*,*) '     ShpPrev: Calling AcuteAngles'
      CALL AcuteAngles
D     WRITE(*,*) '     ShpPrev: Calling RemoveInsIntersects'
      CALL RemoveInsIntersects

C Save the intersection of the profile following boundary with the insulator
C on the far side. (The whole point of doing the previous steps.)

      MacVertX(MacIndx(MacElm,1))=Moved(NoOfMoved,1)
      MacVertY(MacIndx(MacElm,1))=Moved(NoOfMoved,2)

      MacVertX(MacIndx(MacElm,2))=Moved(1,1)
      MacVertY(MacIndx(MacElm,2))=Moved(1,2)

C Compute the location of the boundary at specified x locations
C Note: The orientation of Moved is the inverse of what it should be
C       for SHPPREV.
C       (e.g. it starts at the end of the element, and ends at the
C       beginning of the element) The following lines reverse the Moved Pts
C       to account for this.

      DO I=1,(NoOfMoved+1)/2
        CALL DpXCHNG(Moved(I,1),Moved(NoOfMoved-I+1,1))
        CALL DpXCHNG(Moved(I,2),Moved(NoOfMoved-I+1,2))
      END DO

      CALL GetSpa(MacElm,Spa,NoOfSpa)

      IF (MacSpac(MacElm).NE.'=CRV') THEN
        DO I=1,NoOfSpa
          X=MacVertX(MacIndx(MacElm,1)) +
   & (MacVertX(MacIndx(MacElm,2)) - MacVertX(MacIndx(MacElm,1)))
   & * Spa(I)
          Spa(I)=X
        END DO
      END IF

D     WRITE(*,*) '     ShpPrev:  NoOfSpa=',NoOfSpa
D     WRITE(*,*) '             : Calling ComputeBdry'

      CALL ComputeBdry(Spa,NoOfSpa,Smooth,Spline3D)
```

```
C Transfer the node locations from moved to coords

      DO MicNod=1,NoOfSpa
D       WRITE(*,*) '   ShpPrev: Spa ',MicNod,Spa(MicNod)
D       WRITE(*,*) '          :',Moved(MicNod,1),Moved(MicNod,2)
        CALL MakeCoord(Moved(MicNod,1),Moved(MicNod,2),
     & MacStart(MacElm)+MicNod-1,MacElm,SaveKinetic)
      END DO

      RETURN
      END

C ********************************************************************** ShpSine

      SUBROUTINE ShpSine(MacElm,SaveKinetic)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE '[KJ.FOR.UTIL]Pi.INC'

      INTEGER MacElm,I,NoOfSpa
      REAL*8 X,Y,A,Y0,Spa(MaxNoOfCoords)
      LOGICAL SaveKinetic

C Compute Sine parameters
C Y=Y0+A*SIN( Pi* (Par1*X+Par2) )

      A=(MacVertY(MacIndx(MacElm,1))-MacVertY(MacIndx(MacElm,2)))
     & / (SIN( Pi*(MacPar(MacElm,1)*MacVertX(MacIndx(MacElm,1))
     &   + MacPar(MacElm,2) ))
     &  - SIN( Pi*(MacPar(MacElm,1)*MacVertX(MacIndx(MacElm,2))
     &   + MacPar(MacElm,2) )) )

      Y0=MacVertY(MacIndx(MacElm,1))-A*
     & SIN(Pi*( MacPar(MacElm,1)*MacVertX(MacIndx(MacElm,1))
     &  + MacPar(MacElm,2) ))

      CALL ShpSpacing(MacElm,NoOfSpa,Spa)
      DO I=1,NoOfSpa
        X=MacVertX(MacIndx(MacElm,1)) +
     & (MacVertX(MacIndx(MacElm,2))-MacVertX(MacIndx(MacElm,1)))
     & *Spa(I)
        Y=Y0+A*SIN(Pi*( MacPar(MacElm,1)*X+MacPar(MacElm,2) ))
        CALL MakeCoord(X,Y,MacStart(MacElm)+(I-1),MacElm,SaveKinetic)
      END DO

      RETURN
      END

C ********************************************************************** ShpSpacing

      SUBROUTINE ShpSpacing(MacElm,NoOfSpa,Spa)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE '[KJ.FOR.UTIL]Pi.INC'
```

```
      INTEGER MacElm,NoOfSpa,I,Length,NoOfPts,TNoOfSpa
      REAL*8 Spa(*),StepX

      REAL*8 GeomSeries
      EXTERNAL GeomSeries

      NoOfSpa=NoOfMicElems(MacElm)*BasisDeg

      Spa(1)=0.0
      Spa(NoOfSpa+1)=1.0


      TNoOfSpa=(NoOfSpa+1)/2
      IF (MacSpac(MacElm).EQ.'EQUL') THEN

      IF (BasisDeg.EQ.2) THEN
        DO I=2,TNoOfSpa
          Spa(2*I-1)=FLOAT(I-1)/FLOAT(TNoOfSpa)
        END DO
        DO I=2,NoOfSpa,2
          Spa(I)=(Spa(I+1)+Spa(I-1))/2.0
        END DO
      ELSE
        DO I=2,NoOfSpa
          Spa(I)=FLOAT(I-1)/FLOAT(NoOfSpa)
        END DO
      END IF

      ELSE IF (MacSpac(MacElm).EQ.'LOG') THEN

      IF (BasisDeg.EQ.2) THEN
        StepX=1.0/GeomSeries(MacRatio(MacElm),TNoOfSpa-1)

        DO I=2,TNoOfSpa
          Spa(2*I-1) = Spa(2*(I-1)-1) + StepX
          StepX=StepX*MacRatio(MacElm)
        END DO
        DO I=2,NoOfSpa,2
          Spa(I)=(Spa(I+1)+Spa(I-1))/2.0
        END DO

      ELSE

        StepX=1.0/GeomSeries(MacRatio(MacElm),NoOfSpa-1)

        DO I=2,NoOfSpa
          Spa(I) = Spa(I-1) + StepX
          StepX=StepX*MacRatio(MacElm)
        END DO

      END IF

      ELSE IF (MacSpac(MacElm).EQ.'CHEB') THEN

      IF (BasisDeg.EQ.2) THEN

        DO I=2,TNoOfSpa
          Spa(2*I-1)=(1.0-COS((I-1)*Pi/TNoOfSpa))/2.0
```

```
          END DO

          DO I=2,NoOfSpa,2
            Spa(I)=(Spa(I+1)+Spa(I-1))/2.0
          END DO

        ELSE

          DO I=2,NoOfSpa
            Spa(I)=(1.0-COS((I-1)*Pi/NoOfSpa))/2.0
          END DO
        END IF

        ELSE IF (MacSpac(MacElm).EQ.'USER') THEN

          OPEN(UNIT=1,FILE=MAcSpaFile(MacElm),STATUS='OLD',
     & FORM='FORMATTED')
          READ(1,*) NoOfPts

          IF (NoOfPts.NE.NoOfSpa) THEN
            CLOSE(1)
            CALL StringLength(MacSpaFile(MacElm),Length)
            WRITE(*,*) 'Error: number of points in ',
     & MacSpaFile(MacElm)(1:Length),' is not ',NoOfSpa
            WRITE(*,*) '       ',MacSpaFile(MacElm)(1:Length),' has ',
     & NoOfPts,' points'
            WRITE(*,*) '       Program cannot continue until this is ',
     & 'corrected.'
            STOP 'In Routine: ShpUser'
          END IF

          DO I=1,NoOfSpa
            READ(1,*) Spa(I)
          END DO
          CLOSE(1)

        ELSE

          WRITE(*,*) 'Error: Unknown Spacing=',MacSpac(MacElm)
          STOP 'In Routine: ShpUser'

        END IF

        RETURN
        END

C ********************************************************************** ShpTRNS

        SUBROUTINE ShpTRNS(MacElm,SaveKinetic)

C This subroutine generates the mesh points for line Shps.  (Straight
C line segments between macro mesh pts)

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBPar.INC'
        INCLUDE 'ECBMac.INC'
        INCLUDE 'ECBMsh.INC'
        INCLUDE 'ECBBC.INC'
```

```
      INTEGER MacElm,MicNod
      REAL*8 StepX,StepY,X,Y,StepBC
      LOGICAL SaveKinetic

C Compute the steps in the x and y directions for micro elements
C between macro-end points MacElm and MacElm plus 1.

      StepX=(MacVertX(MacIndx(MacElm,2))-MacVertX(MacIndx(MacElm,1)))/
     & (NoOfMicElems(MacElm)*BasisDeg)
      StepY=(MacVertY(MacIndx(MacElm,2))-MacVertY(MacIndx(MacElm,1)))/
     & (NoOfMicElems(MacElm)*BasisDeg)
      StepBC=(MacBCVal(MacElm+1)-MacBCVal(MacElm-1))/
     & (NoOfMicElems(MacElm)*BasisDeg)

C Make the mesh points and boundary conditions

      DO MicNod=0,BasisDeg*NoOfMicElems(MacElm)-1
        X=MacVertX(MacIndx(MacElm,1))+StepX*MicNod
        Y=MacVertY(MacIndx(MacElm,1))+StepY*MicNod
        MacBCVal(MacElm)=MacBCVal(MacElm-1)+StepBC*MicNod
        CALL MakeCoord(X,Y,MacStart(MacElm)+MicNod,MacElm,SaveKinetic)
      END DO

      RETURN
      END

C ********************************************************************** ShpUser

      SUBROUTINE ShpUser(MacElm,SaveKinetic,Smooth)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBMac.INC'
      INCLUDE 'ECBMove.INC'

      INTEGER MacElm,I,NoOfSpa,J
      LOGICAL SaveKinetic,Smooth
      REAL*8 Spa(MaxNoOfCoords),D,X

      OPEN(UNIT=1,FILE=MacBdryFile,STATUS='OLD',FORM='FORMATTED')

      READ(1,*) NoOfMoved
      DO I=1,NoOfMoved
        READ(1,*) Moved(I,1),Moved(I,2)
      END DO
      CLOSE(1)

      CALL GetSpa(MacElm,Spa,NoOfSpa)

C Compute the location of the boundary at specified x locations

D     WRITE(*,*) '     ShpUser: NoOfMoved=',NoOfMoved
D     WRITE(*,*) '            :  NoOfSpa=',NoOfSpa
D     WRITE(*,*) '            : Calling ComputeBdry'

      IF (.NOT.Spline3D) THEN
        DO I=1,NoOfSpa
          X=MacVertX(MacIndx(MacElm,1)) +
```

```
     & (MacVertX(MacIndx(MacElm,2)) - MacVertX(MacIndx(MacElm,1)))
     & * Spa(I)
            Spa(I)=X
          END DO
        END IF

        CALL ComputeBdry(Spa,NoOfSpa,Smooth,Spline3D)

        DO I=1,NoOfSpa
          CALL MakeCoord(Moved(I,1),Moved(I,2),
     & MacStart(MacElm)+(I-1),MacElm,SaveKinetic)
        END DO

        RETURN
        END

C ************************************************************ TestForSingularity

        SUBROUTINE TestForSingularity(Singularity,SingLoc)

        INCLUDE 'ECBHdr.INC'

        INTEGER SingLoc
        INCLUDE 'EcbElm.INC'
        LOGICAL Singularity

        INTEGER J
        REAL*8 D,Small/1.0E-10/

C See if we need to correct for singularities. Singularity occurs when
C source point is the same as the current node. This can be avoided by
C choosing source points interior to the region.

        Singularity=.FALSE.

        SingLoc=0
        DO J=1,BasisDeg+1
          D=((SourceR(1)-ElemR(1,J))*(SourceR(1)-ElemR(1,J)) +
     & (SourceR(2)-ElemR(2,J))*(SourceR(2)-ElemR(2,J)))
          IF (D.LT.Small*Small) THEN
            Singularity=.TRUE.
            SingLoc=J
            RETURN
          END IF
        END DO

        RETURN
        END

C ************************************************************** UpdateBCValues

        SUBROUTINE UpdateBCValues(NoOfdBCValues,dBCValues)

        INCLUDE 'ECBHdr.INC'
        INCLUDE 'ECBMsh.INC'
        INCLUDE 'ECBBC.INC'

        INTEGER J,I,NoOfdBCValues
```

```
      REAL*8 dBCValues(*),Factor,VecNormDP,MagdBC
      LOGICAL KineticBC
      EXTERNAL KineticBC,VecNormDP

C Based on the magnitude of the desired change in boundary values,
C use relaxation technique (scale down the desired step).

      MagdBC=VecNormDP(NoOfdBCValues,dBCValues)
      IF (MagdBC.GT.1.0) THEN
        Factor=MAX(MIN(0.9D0,2.0D0/MagdBC),0.2D0)
        WRITE(*,*) 'Factor=',Factor
      ELSE
        Factor=1.0
      END IF

      J=1
      DO I=1,NoOfCoords
        IF (KineticBC(BCType(I))) THEN
          BCValues(I)=BCValues(I)+dBCValues(J)*Factor
          J=J+1
        END IF
      END DO

      RETURN
      END

C ************************************************************************** UStar

      REAL*8 FUNCTION UStar(Dist2,Gs,Den,M)

      INCLUDE 'ECBPar.INC'
      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECGaus.INC'
      INCLUDE 'ECBG.INC'
      INCLUDE 'ECBElm.INC'

      INTEGER Gs
      REAL*8 Dist2,Den,M

      REAL*8 First,Second

      REAL*8 KEIP
      EXTERNAL KEIP

C Variables

C SourceR(*)    Coordinates of the Source Point

      IF (Axisymmetric) THEN

C The problem is Axisymmetric.
C UStar Multiplied by 2Pi

C Note: the Gr(Gs,1) term is here because it is not included in the computation
C of the isojacobian for axisymmetric problems.

        UStar = 2.0D0*KEIP(M)*Gr(Gs,1)/SQRT(Den)
```

```
      ELSE

C The problem is planar
C Ustar multiplied by 2Pi

        UStar = -LOG(Dist2)/2.0D0

      END IF

      RETURN
      END

C ********************************************************************** WriteCal

      SUBROUTINE WriteCal(FileName)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBCal.INC'
      CHARACTER*(*) FileName
      LOGICAL FileExist
      INTEGER I,J,Version/1/

      INQUIRE(FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) THEN
        OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')
      ELSE
        OPEN(UNIT=1,FILE=FileName,STATUS='OLD',FORM='FORMATTED')
      END IF

      WRITE(1,*) Version

      DO I=1,NoOfCoords
        DO J=1,NoOfCoords
          WRITE(1,*) BemG(I,J),BemH(I,J),I,J
        END DO
      END DO

      CLOSE(1)

      RETURN
      END

C ********************************************************************** WriteInt

      SUBROUTINE WriteInt(FileName)

      INCLUDE 'ECBHDR.INC'
      INCLUDE 'ECBINT.INC'

      CHARACTER*(*) FileName
      LOGICAL FileExist
      INTEGER Version,I,K

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED')
      WRITE(1,*) 1

      WRITE(1,*) NoOfInternal,NoOfSpines,IntPerSpine
```

```
      DO I=1,NoOfInternal
        WRITE(1,*) (InternalPts(I,K),K=1,NoOfDim),InternalPot(I)
      END DO

      CLOSE(1)

      RETURN
      END

C ********************************************************************** WriteMac

      SUBROUTINE WriteMac(FileName)

C This subroutine writes the contents of the BEM.MAC file.  This file contains
C the information required to make the BEM.INP file, the more detailed file
C used by the BEMLIN.FOR program for input.

      INCLUDE 'ECBMAC.INC'
      CHARACTER*(*) FileName
      INTEGER I,Version
      LOGICAL FileExist

      INQUIRE (FILE=FileName,EXIST=FileExist)
      IF (.NOT.FileExist) CALL ErrorFileName(FileName,'ReadMac')

      OPEN(UNIT=1,FILE=FileName,STATUS='NEW',FORM='FORMATTED',RECL=132)

      WRITE(1,*) 3

      IF (NoOfMacVert.GT.MaxNoOfMacVert) THEN
        WRITE(*,*) 'Error: Number of macro-vertices,',NoOfMacVert
        WRITE(*,*) '       is larger than the maximum number of '
        WRITE(*,*) '       macro-vertices allowed:',MaxNoOfMacVert
        STOP 'In Routine: WriteMac'
      END IF
      WRITE(1,*) NoOfMacVert

      DO I=1,NoOfMacVert
        WRITE(1,*) MacVertX(I),', ',MacVertY(I)
      END DO

      WRITE(1,*) NoOfMacElems
      DO I=1,NoOfMacElems
        WRITE(1,*) '''',MacShape(I),''',',NoOfMicElems(I),
     & ',''',MacBCType(I),''',',MacBCVal(I),',''',
     & MacSpac(I),''',',MacRatio(I)
        IF (MacShape(I).EQ.'LINE') THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2)
        ELSE IF (MacShape(I).EQ.'SINE') THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2)
          WRITE(1,*) MacPar(I,1),MacPar(I,2)

C (Y-Y0)=A*Sin(Par1*X+Par2)) A and Y0 are solved for

        ELSE IF (MacShape(I).EQ.'CIR') THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2)
          WRITE(1,*) MacPar(I,1),MacPar(I,2)
        ELSE IF ((MacShape(I).EQ.'USER').OR.
```

```
     & (MacShape(I).EQ.'USR')) THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2),',''',MacBdryFile,''''
        ELSE IF (MacShape(I).EQ.'TRNS') THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2)
        ELSE IF (MacShape(I).EQ.'PREV') THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2)
          WRITE(1,*) MacPar(I,1),MacPar(I,2),MacPar(I,3),MacPar(I,4)
        ELSE IF (MacShape(I).EQ.'FLAT') THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2)
          WRITE(1,*) MacPar(I,1),MacPar(I,2)
        ELSE IF (MacShape(I).EQ.'FLTL') THEN
          WRITE(1,*) MacIndx(I,1),MacIndx(I,2)
          WRITE(1,*) MacPar(I,1),MacPar(I,2)
        ELSE
          WRITE(*,*) 'Error: Unknown Macro-Element Type:',MacShape(I)
          STOP 'In Routine: WriteMac'
        END IF
        WRITE(1,*) MacStart(I),MacEnd(I)
        IF (MacSpac(I).EQ.'USER') WRITE(1,*) '''',MacSpaFile(I),''''
      END DO
      CLOSE(1)

      RETURN
      END

C ********************************************************************** WriteMesh

      SUBROUTINE WriteMesh(FileName)

      CHARACTER*(*) FileName
      INTEGER Length,Version/1/,I,J
      LOGICAL FileExist

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBMsh.INC'
      INCLUDE 'ECBBC.Inc'

      OPEN(UNIT=1,FILE=FileName,STATUS='UNKNOWN',FORM='FORMATTED')

      WRITE(1,*) Version

      WRITE(1,*) NoOfCoords
      DO I=1,NoOfCoords
        WRITE(1,*) (Coords(I,J),J=1,NoOfDim)
        WRITE(1,*) BCValues(I),SolValues(I),Natural(I),
     & ',''',BCType(I),''''
      END DO

      WRITE(1,*) NoOfElems
      DO I=1,NoOfElems
        WRITE(1,*) (NodeMap(I,J),J=1,BasisDeg+1),ElemNat(I),
     & ',''',ElemBCType(I),''''
      END DO

      CLOSE(1)

      RETURN
      END
```

```
C ************************************************************************ WriteSrc

      SUBROUTINE WriteSrc(FileName)

      INCLUDE 'ECBHdr.INC'
      INCLUDE 'ECBSrc.INC'
      INCLUDE 'ECBMsh.INC'

      INTEGER Version/1/,I,J
      CHARACTER*(*) FileName
      LOGICAL FileExist

      OPEN(UNIT=1,FILE=FileName,STATUS='UNKNOWN',FORM='FORMATTED')

      WRITE(1,*) Version

      IF (Version.EQ.1) THEN
        WRITE(1,*) NoOfSrc

        IF (NoOfSrc.NE.NoOfCoords) THEN
          WRITE(*,*) 'Error: Number of source points is not equal to'
          WRITE(*,*) '       the number of coordinates.'
          WRITE(*,*) '      Filename=',FileName
          WRITE(*,*) '       NoOfSrc=',NoOfSrc
          WRITE(*,*) '    NoOfCoords=',NoOfCoords
          STOP 'In Routine: WriteSrc'
        END IF

        DO I=1,NoOfSrc
          WRITE(1,*) (Src(I,J),J=1,NoOfDim)
        END DO
      ELSE
        CALL ErrorVersion(Version,FileName,'WriteSrc')
      END IF

      CLOSE(1)

      RETURN
      END

C ************************************************************************ XY2T

      SUBROUTINE XY2T(X,Y,T)

C This subroutine taken from Ken Jordan's Masters Thesis CalTech 1985

C This subroutine returns the angle, T, from the positive x-axis, of
C the point X,Y

      REAL*8 X,Y,T,PI/3.1415926536/

      IF (X.GT.0.0) THEN
        IF (Y.GE.0.0) THEN
          T=ATAN(Y/X)
        ELSE
          T=ATAN(Y/X)+2.0*PI
        END IF
      END IF
```

```
      IF (X.LT.0.0) T=ATAN(Y/X)+PI

      IF (X.EQ.0.0) THEN
        IF (Y.GT.0.0) T=PI/2.0
        IF (Y.EQ.0.0) T=0.0
        IF (Y.LT.0.0) T=3.0*PI/2.0
      END IF

      RETURN
      END
```

```
C Include File: ECBBC.INC
C Written by: Ken Jordan                    Date: 3/18/88

C This file contains the declarations for the bc values and solution results.
C Used for reading/writing .BC files.

C ------------------------------------------------------------------ Variables

C BCType()      The boundary condition for each node.
C               'ESS '  Essential Boundary Condition
C               'NAT '  Natural Boundary Condition
C               'CRNT'  Natural condition where dPot/dn=-Wagner
C               'LIN '  Linear
C               'TAFA'          Tafel Anodic
C               'TAFC' Tafel Cathodic
C               'BV  '  Butler-Volmer
C               'USR '  Usr supplied (Function...)

C BCValues(*)   The boundary condition value for each node
C ElemBCType(*)         BC Kinetics Type for each element.
C ElemNat(*)    Logical TRUE if element I has natural BC's.
C Natural(*)    Logical  True if  BCVAlues(I) is a natural BC
C SolValues(*)  if Natural(i) is true, then this is the essential value.
C               if Natural(i) is false, then this is the natural value.

C ------------------------------------------------------------------ Declarations

      REAL*8 BCValues(MaxNoOfCoords),SolValues(MaxNoOfCoords)
      LOGICAL Natural(MaxNoOfCoords),ElemNat(MaxNoOfCoords)
      CHARACTER*4 BCType(MaxNoOfCoords),ElemBCType(MaxNoOfCoords)

      COMMON/ECBBC/BCValues,SolValues,Natural,ElemNat,
     & BCType,ElemBCType
```

```
C Include File: ECBbc2
C Written By: Ken Jordan                    8/24/88

C This file declares variables for saving old BEM problem boundary conditions
C and solution values

C ------------------------------------------------------------------ Variables

C Natural2(*)    Logical  True if  BCVAlues(I) is a natural BC
C BCValues2(*) The boundary condition value for each node
C SolValues2(*) if Natural2(i) is true, then this is the essential value.
C                if Natural2(i) is false, then this is the natural value.

C ------------------------------------------------------------------ Declarations

      REAL*8 BCValues2(MaxNoOfCoords),SolValues2(MaxNoOfCoords)
      LOGICAL Natural2(MaxNoOfCoords)

      COMMON/ECBBC2/BCValues2,SolValues2,Natural2
```

```
C Include File: ECBCal
C Written by: Ken Jordan                    Date: 3/18/88

C This file contains the declarations necessary to access the BEM Matrices
C BEMH(,) and BEMG(,).

C ------------------------------------------------------------ Declarations

      REAL*8 BEMH(MaxNoOfCoords,MaxNoOfCoords),
     & BEMG(MaxNoOfCoords,MaxNoOfCoords)

      COMMON /ECBCal/BemH,BemG
```

```
C Include File: ECBELM.INC                    8/10/88
C Written by: Ken Jordan

C This include file declares the common block used to store the
C current local element coordinates, and the current source point for
C integration over the local element.

      COMMON /ECBELM/ElemR,SourceR
      REAL*8 ElemR(MaxNoOfDim,MaxBasisDeg+1),SourceR(MaxNoOfDim)
```

```
C INCLUDE FILE: ECBG.INC                    7/29/88
C Written by: Ken Jordan

C This file contains declarations for variables passed to UStar and QStar,
C such as GPhi, GR,GIsoJacob,GNormal, that only need to be calculated once
C for each gauss point on the element, and not for each source point.

      REAL*8 GPhi(MaxNoOfGauss,MaxBasisDeg+1),
     & GR(MaxNoOfGauss,MaxNoOfDim),GIsoJacob(MaxNoOfGauss),
     & GNormal(MaxNoOfGauss,MaxNoOfDim)

      COMMON /ECBG/GPhi,Gr,GIsoJacob,GNormal
```

```
C Include File: ECBHdr.INc
C Written by: Ken Jordan                    Date: 3/18/88

C This include file contains the declarations for variables and parameters
C used to solve the BEM problem

C ------------------------------------------------------------------ Parameters

C MaxBasisDeg  The maximum basis function allowed
C MaxNoOfCoords        The maximum number of mesh points and source points
C                 allowed
C MaxNoOfDim The maximum dimension allowed.

C ------------------------------------------------------------------ Variables

C BasisDeg       The degree of the basis functions.
C NoOfDim        The actual number of dimensions specified by the user.

C ------------------------------------------------------------------ Declarations

      PARAMETER MaxNoOfCoords=500
      PARAMETER MaxNoOfDim=2
      PARAMETER MaxBasisDeg=2

      INTEGER NoOfDim,BasisDeg
      COMMON /ECBHdr/NoOfDim,BasisDeg
```

```
C Include File: ECBINT.INC
C Written By: Ken Jordan                    Date: 10/20/89

C This include file declares the internal points

C NoOfInternal  The number of internal points
C NoOfSpines    The number of spines
C IntPerSpine   The number of internal points per spine
C InternalPts(,2) The locations of the internal Points.
C InternalPot()  The potential at the internal points.

      INTEGER NoOfInternal,NoOfSpines,IntPerSpine
      REAL*8 InternalPts(MaxNoOfCoords,MaxNoOfDim),
     & InternalPot(MaxNoOfCoords)

      COMMON /ECBINT/ NoOfInternal,NoOfSpines,IntPerSpine,
     & InternalPts,InternalPot
```

```
C Include File: ECBMac.INC
C Written By: Ken Jordan                    Date: 6/5/87

C This include file defines and dimensions the variables used in the ECBMAC
C common block for setting up a mesh.  The file is used by program ECBPrePro.


C ------------------------------------------------------------------ Parameters

C MaxNoOfMacVert is used to dimension the matrices used to read the BEM.MAC
C                 file.

C ------------------------------------------------------------------ Variables

C NoOfMacElems        The number of macro elements.
C NoOfMacVert Number of macro vertices in the BEM.MAC file
C NoOfMicElems() The number of elements for each macro element.
C MacBCType() The b.c. to be applied to the element
C               'ESS','NAT','CRNT','BV','LIN','TAFA','TAFC'
C MacBCVal()  The b.c. value for the element
C               Value of the potential in the node I if BCType(i)=0
C               Value of the potential derivative if BCType(i)=1
C MacIndx(,)  The first index is the macro element number,
C               If the macro element is a line then the contents for
C                 second index=1 is the index number of the macro vertex
C                               at the start of the line segment.
C                 second index=2 is the index number of the macro vertex
C                               at the end of the line segment.
C MacPar(,)   Parameters used for various macro element types.
C               'LINE'
C               'SINE' Y=Y0+A*SIN(MacPar(I,1)*X+MacPar(I,2))
C            'FLAT' Horizontal surface MacPar(I,1) above highest point
C                       of MacroElement MacPar(I,2)
C               'PREV' Surface of with shape of MacroElement MacPAr(I,2)
C                       spaced MacPar(I,1) away.
C MacShape()  The type of macro-element.  Currently, the only type allowed
C               is 'LINE','SINE',FLAT,PREV,USER,CIR_
C MacSpac()   The spacing for the macro-element. (LOG, CHEB, EQUL,=CRV,USR)
C MacRatio()  The spacing parameter for LOG spacing.
C MacVertX()  X coordinate of macro vertex
C MacVertY()  Y coordinate of macro vertex
C MacSPAFile() The filename to use for user-defined spacings
C MacBdryFile The filename to use for temporary storage of the moving
C               boundary

C ------------------------------------------------------------------ Declarations

      COMMON /ECBMac/NoOfMacVert,MacVertX,MacVertY,NoOfMacElems,
     & MacShape,NoOfMicElems,MacBCType,MacBCVal,MacIndx,MacPar,
     & MacSpac,MacRatio,MacStart,MacEnd,MacSpaFile,MacBdryFile

      PARAMETER MaxNoOfMacVert=10

      INTEGER NoOfMacVert,NoOfMacElems,NoOfMicElems(MaxNoOfMacVert),
     & MacIndx(MaxNoOfMacVert,2),MacStart(MaxNoOfMacVert),
     & MacEnd(MaxNoOfMacVert)
      REAL*8 MacVertX(MaxNoOfMacVert),MacVertY(MaxNoOfMacVert),
     & MacBCVal(MaxNoOfMacVert),MacPar(MaxNoOfMacVert,4),
```

```
     & MacRatio(MaxNoOfMacVert)

         CHARACTER*4 MacShape(MaxNoOfMacVert),MacBCType(MaxNoOfMacVert),
     & MacSpac(MaxNoOfMacVert)
         CHARACTER*50 MAcSpaFile(MaxNoOfMacVert),MacBdryFile

C *.MAC file structure

C NoOfMacVert
C MaxVertX(i),MacVertY(i)

C NoOfMacElems

C MacShape(i),NoOfMicElems(i),MacBCType(i),MacBCVal(i)

C MacIndx(i,1),MacIndx(i,2) for mac type 'LINE'

C MacIndx(i,1),MacIndx(i,2) for mac type 'SINE'
C MacPar(I,1),MacPar(I,2)

C Corners can be identified by changes in the BC Type
C Transition elements are identified by changes in the BC Value.
```

```
C Include File: ECBMove.INC
C Written By: Ken Jordan                    10/18/89 (After the Big One)

C This file declares variables for the common block ECBMOVE.  These variables
C are used in boundary movement

C NoOfIns        Number of points in unmoving boundary
C Ins(,2)        Points in unmoving boundary.  (,1)=x, (,2)=y
C Orig(2,2)      First Index: =1, second point on original surface
C                             =2, next to last point on original surface
C                Second index: dimension, (=1,x), (=2,y)
C NoOfMoved      Number of points in moving boundary
C Moved(,2)      Points in moving boundary. (,1)=x, (,2)=y
C AcuteAngleBefore      .TRUE. if an acute angle exists between moving and
C                unmoving boundaries at start of moving boundary
C NoAngleBefore                 .TRUE. if moving and unmoving boundaries start
C                        at the same point.
C AcuteAngleAfter       .TRUE. if an acute angle exists between moving and
C                unmoving boundaries at end of moving boundary
C NoAngleAfter          .TRUE. if the moving and unmoving boundaries end at
C                        the same point.
C Nx(,2)         normal to point.  (,1) x direction, (,2) y direction
C dXdN()         current density to point.
C StaRev         .TRUE. if the start of the moving boundary is moving backwards
C EndRev         .TRUE. if the end of the moving boundary is moving backwards

      INTEGER NoOfIns,NoOfMoved,NoOfdXdN
      INTEGER First,Second,NextToLast,Last
      LOGICAL AcuteAngleBefore,AcuteAngleAfter,StaRev,EndRev
      LOGICAL NoAngleBefore,NoAngleAfter
      REAL*8 Ins(MaxNoOfCoords,MaxNoOfDim)
      REAL*8 Moved(MaxNoOfCoords,MaxNoOfDim)
      REAL*8 Orig(2,MaxNoOfDim)
      REAL*8 Nx(MaxNoOfCoords,MaxNoOfDim),dXdN(MaxNoOfCoords)

      COMMON/ECBMOVE/NoOfIns,Ins,NoOfMoved,Moved,Orig,
     & AcuteAngleBefore,AcuteAngleAfter,Nx,dXdN,StaRev,EndRev,
     & First,Second,NextToLast,Last,NoAngleBefore,NoAngleAfter
```

```
C Include File: ECBMsh.INC
C Written by: Ken Jordan                    Date: 3/18/88

C This include file declares the variables that contain the mesh.

C ------------------------------------------------------------------ Variables

C Coords(*,*)   The coordinates of the node. The first index is the node
C               number; the second index is the dimension number. The
C               first dimension is alway the R dimension
C NodeMap(,)    The first index is the element number, the second index
C               is the local node number. The contents are the index
C               of the global node.
C NoOfCoords    The number of coordinates in the mesh.
C NoOfElems     The number of elements in the mesh.

C ------------------------------------------------------------------ Declarations

      INTEGER NoOfCoords,NoOfElems,NodeMap(MaxNoOfCoords,MaxBasisDeg+1)
      REAL*8 Coords(MaxNoOfCoords,MaxNoOfDim)

      COMMON /ECBMsh/NoOfCoords,Coords,NoOfElems,NodeMap
```

```
C Include File: ECBPar.INc
C Written by: Ken Jordan                    Date: 3/18/88

C This include file declares the variables used for secondary current
C distributions (aA, aC, IRatio, JohnNewman) and for geometry
C (Axisymmetric, RInner)

C ------------------------------------------------------------------ Variables

C Kinetic Parameters

C aA          Anodic transfer coefficient
C aC          Cathodic transfer coefficient
C WagLin      Wagner number for Linear Kinetics
C             = istar / i0 = kappa R T / F xstar i0

C Nondimensionalization Constants

C tstar       The quantity used to make time dimensionless
C             = rho * nelec * F * xstar / (MW * istar)
C             = rho nelec F^2 xstar^2 / MW kappa R T
C istar       The quantity used to make current density dimensionless
C             = kappa PhiStar / xstar
C             = kappa R T / F xstar

C CorLev      .TRUE. if solving problem with corrosion and deposition
C             at same time
C Corros      Quantity used to scale corrosion current to deposition
C             current.
C             = Da F CaBulk / istar xstar
C             Corros only has meaning if CorLev is .TRUE.
C             (not ideal: better is = Na(local) F / i(local) )
C                                   = Da CaBulk F / i(local) delta
C                          where delta=MT BL thickness
C          (it's the best we can do at present.)

C Periodic Current Reversal Parameters

C PerRev      .TRUE. if solving periodic current reversal problem
C CRRatio     Periodic Current Reversal Ratio (>0 always)
C             (I init/I final). Only has meaning if PerRev=.TRUE.
C WaLinPCR    Ratio of Linear Wagner Numbers for Periodic Current
C             Reversal.

C NoOfPCR     Number of times to reverse the current
C PCRMax      .TRUE. if primary c.d. during dissolution, and uniform c.d.
C             during deposition

C Method Parameters

C Axisymmetric .TRUE. if the problem is axisymmetric
C Spline3D    .TRUE. if spline should be based on x(d),y(d),d
C             .FALSE. if spline should be based on Y(x),x

C MaxIter     Maximum number of iterations allowed before program
C             quits when doing Newton Raphson technique
C SrcFactor   Factor (>1) used to determine radius of circle that
C             has source points on it.
```

```
C Tolerance     Used to determine convergence tolerance for NR technique
C               and to tell when two points are the same.

C BC Parameters

C KineticsFile  filename containing data for user defined kinetics routine.

C ------------------------------------------------------------ Declarations

      INTEGER MaxIter,NoOfPCR
      LOGICAL Axisymmetric,Spline3D,CorLev,PerRev,PCRMax
      REAL*8 WagLin,aA,aC,Tolerance,SrcFactor,Corros,CRRatio
      REAL*8 tstar, istar,WaLinPCR
      CHARACTER*80 Kineticsfile

      COMMON/ECBPar/ aA,aC,WagLin,tstar,istar,Corros,CRRatio,WALinPCR,
     & Axisymmetric,CorLev,Spline3D,PerRev,
     & Tolerance,SrcFactor,MaxIter,KineticsFile,
     & PCRMax,NoOfPCR
```

```
C Include File: ECBPar2.INC

C This include file contains declarations and COMMON statements for the
C dimensional parameters describing the ECB problem.

C ------------------------------------------------------------Input Parameters

C F
C CaBulk [mol/l], Da      (Bulk Concentration, Diffusivity) Parameters For
C                   Corrosive Chemical
C Kappa                   Electrolyte Conductivity [1/ohm-cm]
C aA, aC, i0 [mA/cm2]     Metal Electrodeposition Kinetics
C xstar [um],PhiStar [V]  Geometry Scale (amplitude), Potential Scale

C ----------------------------------Calculated Nondimensionalization Constants

C (phistar        = RT/F)
C istar           = kappa PhiStar / xstar

C ------------------------------------------------Important Dimensionless Groups

C Waglin          = istar / i0
C Corros          = Da F CaBulk / i0 delta

      REAL*8 F
      REAL*8 CaBulk,Da,kappa
      INTEGER nelec
      REAL*8 i0
      REAL*8 xstar, PhiStar
      REAL*8 rho, MW
      REAL*8 idep

      COMMON /EcbPar2/F,CaBulk,Da,kappa,nelec,i0,xstar,
     & PhiStar,rho,MW,idep
```

```
c Include File: ECBRemesh.INC

C Contains declarations for common block variables used in remeshing.

C Variables

C SpaFac        Used to weight importance of spacing (0 = unimportant
c               1=equal in importance to curvature)
C UpFac                 Bound the curvature by avg error * Up Fac as maximum
C LowFac        Bound the curvature by avg error * Low Fac as minimum
C SlopFac       Bound the changes in curvature by (Max-Min)/TotDis/SlopFac

      COMMON /ECBRemesh/SpaFac,UpFac,LowFac,SlopFac
      REAL*8 SpaFac,UpFac,LowFac,SlopFac
```

```
C Include File: ECBSrc.INC
C Written by: Ken Jordan                    Date: 3/18/88

C This include file declares the variables used to describe the source
C points.

C ------------------------------------------------------------------ Variables

C NoOfSrc       The number of source points. (This should equal the number
C               of nodes)
C Src(,) The coordinates of the source points. The first index is
C               the source point number,the second index is the dimension

C ------------------------------------------------------------------ Declarations

      INTEGER NoOfSrc
      REAL*8 Src(MaxNoOfCoords,MaxNoOfDim)

      COMMON /ECBSrc/NoOfSrc,Src
```

```
C Include File ECBTime.INC
C This file contains variables used by program ECBM to do time stepping.

C TimeCounter
C MaxTimeCounter
C Time
C dTime
C EndTime
C DesdX
C DesdTime
C MotdTime
C LastTime

      INTEGER TimeCounter,MaxTimeCounter
      REAL*8 Time,dTime,Endtime,DeltaTime
      REAL*8 DesdX,DesdTime,MotdTime
      LOGICAL LastTime

      COMMON /ECBTime/Timecounter,MaxtimeCounter,Time,Dtime,endtime,
     & DesdX,DesdTime,MotdTime,LastTime,DeltaTime
```

```
C Include File: ECBTMac.INC
C Written By: Ken Jordan                    Date: 6/5/87

C This include file defines and dimensions the variables used in the ECBMAC
C common block for setting up a mesh.  The file is used by program ECBPrePro.


C ------------------------------------------------------------------ Variables

C TNoOfMicElems() The number of elements for each macro element
C TMacBCType()        The b.c. to be applied to the element
C                 'ESS','NAT','CRNT','BV','LIN','TAFA','TAFC'
C TMacBCVal() The b.c. value for the element
C                 Value of the potential in the node I if BCType(i)=0
C                 Value of the potential derivative if BCType(i)=1
C TMacIndx(,)  The first index is the macro element number,
C                 If the macro element is a line then the contents for
C                   second index=1 is the index number of the macro vertex
C                                 at the start of the line segment.
C                   second index=2 is the index number of the macro vertex
C                                 at the end of the line segment.
C TMacPar(,)   Parameters used for various macro element types.
C                 'SINE' Y=Y0+A*SIN(MacPar(I,1)*X+MacPar(I,2))
C TMacShape()  The type of macro-element.  Currently, the only type allowed
C                 is 'LINE','SINE'
C TMacSpac()   The spacing for the macro-element. (LOG, CHEB, EQUL)
C TMacRatio()  The spacing parameter for LOG spacing.

C ------------------------------------------------------------------ Declarations

      COMMON /ECBTMac/ TMacShape,TNoOfMicElems,TMacBCType,TMacBCVal,
     & TMacIndx,TMacPar,TMacSpac,TMacRatio

      INTEGER TNoOfMicElems(MaxNoOfMacVert),
     & TMacIndx(MaxNoOfMacVert,2)
      REAL*8 TMacBCVal(MaxNoOfMacVert),TMacPar(MaxNoOfMacVert,3),
     & TMacRatio(MaxNoOfMacVert)

      CHARACTER*4 TMacShape(MaxNoOfMacVert),TMacBCType(MaxNoOfMacVert),
     & TMacSpac(MaxNoOfMacVert)
```

```
C Include File: ECGaus.INC
C Written by: Ken Jordan                    Date: 3/18/88

C Contains declarations for the common block containing gaussian quadrature
C variables.  The Gauss Points and Weights may be obtained by calling
C subroutine GauLeg()

C ------------------------------------------------------------------ Parameters

C MaxNoOfGauss          The maximum number of gauss points and weigths allowed by
C                  the program.

C ------------------------------------------------------------------ Variables

C NoOfGaussPts The number of gauss points specified by the user.
C GaussWts(*)   The gauss weights to use (interval 0->1)
C GaussPts(*)   The Gauss points to use (interval 0->1)

C ------------------------------------------------------------------ Declarations

      PARAMETER MaxNoOfGauss=20

      INTEGER NoOfGaussPts
      REAL*8 GaussWts(MaxNoOfGauss),GaussPts(MaxNoOfGauss)

      COMMON/ECGaus/ NoOfGaussPts,GaussPts,GaussWts
```

```
C Include File: ECGLOG.INC
C Written by: Ken Jordan                    Date: 3/18/88

C Contains declarations for the common block containing gaussian quadrature
C variables. The Gauss Points and Weights may be obtained by calling
C subroutine GauLOG(). Points and weights are for integrating from 0 to
C 1, f(x)*LN(1/x).

C ------------------------------------------------------------------ Parameters

C MaxNoOfGauLog       The maximum number of gauss points and weigths allowed by
C                     the program.

C ------------------------------------------------------------------ Variables

C NoOfGauLogPts       The number of gauss points specified by the user.
C GauLogWts(*) The gauss weights to use (interval 0->1)
C GauLogPts(*)  The GauLog points to use (interval 0->1)

C ------------------------------------------------------------------ Declarations

      PARAMETER MaxNoOfGauLog=20

      INTEGER NoOfGauLogPts
      REAL*8 GauLogWts(MaxNoOfGauLog),GauLogPts(MaxNoOfGauLog)

      COMMON/ECGaus/ NoOfGauLogPts,GauLogPts,GauLogWts
```

```
C I/O Subroutines

c       SUBROUTINE Center(String,TWidth,Centered)
c       SUBROUTINE WriteCenter(Lun,Width,Line)
c       SUBROUTINE ClrScr

c       SUBROUTINE ChInp(PROMPT,VAR)                (Obtains Ch,DP,Dt,I, or R
c       SUBROUTINE DPInp(Prompt,Var)                from the user. If a <CR>
c       SUBROUTINE DtInp(Prompt,Date$,Date)  is pressed, 0 or ' ' is
c       SUBROUTINE IInp(Prompt,Var)          stored in the variable)
c       SUBROUTINE RInp(Prompt,Var)
C       SUBROUTINE GetFileName(Prompt,FileName)

c       SUBROUTINE EditCh(Prompt,Var)               (Obtains Ch,DP,Dt,I, or R
c       SUBROUTINE EditI(Prompt,Var)         from the user. If a <CR>
c       SUBROUTINE EditR(Prompt,Var)         is pressed, the previous
c       SUBROUTINE EditDP(Prompt,Var)               value of the variable remains
c       SUBROUTINE EditDtCh(Prompt,Date$,Date)      in the variable)
c       SUBROUTINE EditDtI(Prompt,Date$,Date)

c       SUBROUTINE PauseCrt
c       SUBROUTINE Menu(MenuTitle,MenuTxt,NTxt,MenuPrompt,MenuResps,NResps,
c     & DefResp,UsrResp)
c       SUBROUTINE Yorn(Prompt$,Yes)

C Special I/O Conversion routines. (e.g. 1 line of data to a matrix/
C                                   matrix to 1 line of data)

c       SUBROUTINE CodeI(List,NoOfElems,String)
c       SUBROUTINE CodeR(List,NoOfElems,String)

c       SUBROUTINE DecodeCh(String,Delimiter,List,NoOfElems,MaxNoOfElems)
c       SUBROUTINE DecodeI(String,List,NoOfElems,MaxNoOfElems)
c       SUBROUTINE DecodeR(String,List,NoOfElems,MaxNoOfElems)
C       SUBROUTINE DecodeDP(String,List,NoOfElems,MaxNoOfElems)

C Matrix Manipulation subroutines

c       SUBROUTINE CompareListI(List1,NoOfElems1,
c     & List2,NoOfElems2,Matches,NoOfMatches)
c       SUBROUTINE CompareListR(List1,NoOfElems1,
c     & List2,NoOfElems2,Matches,NoOfMatches)

c       SUBROUTINE FindMaxCh(Matrix,NoOfElems,MaxCh,MaxLoc)
C       SUBROUTINE FindMaxDt(Dates,NoOfDates,MaxDt,MaxLoc)
c       SUBROUTINE FindMaxI(Matrix,NoOfElems,MaxI,MaxLoc)

c       SUBROUTINE FindMinCh(Matrix,NoOfElems,MinCh,MinLoc)
c       SUBROUTINE FindMinDt(Matrix,NoOfElems,MinDt,MinLoc)
c       SUBROUTINE FindMinI(Matrix,NoOfElems,MinI,MinLoc)

c       SUBROUTINE FindMatchesDP(Matrix,NElements,MatchElement,
c     & MatchLocs,Found,NoLocs)
c       SUBROUTINE FindMatchesI(Matrix,NElements,MatchElement,
c     & MatchLocs,Found,NoLocs)
c       SUBROUTINE FindMatchesR(Matrix,NElements,MatchElement,
c     & MatchLocs,Found,NoLocs)
```

```
c       SUBROUTINE MatDateBend(DateCh,NoOfElems,DateN)
c       SUBROUTINE MatExtremaR(Matrix,NoOfPts,Maxi,Mini)
c       SUBROUTINE MatExtremaDP(Matrix,NoOfPts,Maxi,Mini)

c       SUBROUTINE SearchCh(Matrix,NElements,MatchElement,Location,Found,
c   & NoLocs)
c       SUBROUTINE SearchChSub(Matrix,NElements,StartLoc,EndLoc,
c   & MatchElement,Location,Found,NoLocs)
c       SUBROUTINE SearchDP(Matrix,NElements,MatchElement,Location,Found,
c   & NoLocs)
c       SUBROUTINE SearchI(Matrix,NElements,MatchElement,Location,Found,
c   & NoLocs)
c       SUBROUTINE SearchL(Matrix,NElements,MatchElement,Location,Found,
c   & NoLocs)
c       SUBROUTINE SearchR(Matrix,NElements,MatchElement,Location,Found,
c   & NoLocs)

c       SUBROUTINE SortCh(Matrix,NoElems,Pointer)
c       SUBROUTINE SortDP(Matrix,NoElems,Pointer)
c       SUBROUTINE SortI(Matrix,NoElems,Pointer)
c       SUBROUTINE SortR(Matrix,NoElems,Pointer)

C Single Variable Character/Date Manipulation subroutines

c       SUBROUTINE UpperCaseCh(String)
c       SUBROUTINE LowerCaseCh(String)
c       SUBROUTINE ConvertChToI(String2,Int)
c       SUBROUTINE ConvertChToR(String2,Re)
c       SUBROUTINE ConvertChToDP(String2,Re)
c       SUBROUTINE ConvertDtToI(Date$,Mo,Da,Yr)
c       SUBROUTINE ConvertDPToWords(Number,Words)

c       SUBROUTINE DateBend(Date$,Date)          (converts MM/DD/YY to YYMMDD)
c       SUBROUTINE DateUnBend(TDate,Date$) (converts YYMMDD to MM/DD/YY)

c       SUBROUTINE TrimCh(String)          (Places first non-blank char
c                                          in String(1:). Sets all
C                                          trailing blanks to nuls CHAR(0))

c       SUBROUTINE RemoveBlanksFromCh(String)     (Places all non-blank chars
C                                          next to each other starting
C                                          in String(1:). E.g.
C                                          'This is a test' becomes
C                                          'Thisisatest    '
c       SUBROUTINE RemoveCharFromCh(String,Blank)
c       SUBROUTINE StringLength(String,Length)
c       SUBROUTINE RptString(Repeat,NoOfTimes,String)

C File Manipulation Subroutines

c       SUBROUTINE FileRename(OldName,NewName)
c       SUBROUTINE FileDelete(FileName)
c       SUBROUTINE FilePurge(FileName)

C Numeric Manipulation Subroutines

c       Subroutine PRoundDP(Number,PowerOfTen,Rounded)
c       SUBROUTINE IXCHNG(I1,I2)
```

C Plotting Routines

```
c       SUBROUTINE SetPlotter(TermType)
c       SUBROUTINE GetPlotter(TermType)
c       SUBROUTINE SetUpImagen
c       SUBROUTINE SetUpTalaris
c       SUBROUTINE SetUpVT240
```

```
C ********************************************************************** Center

      SUBROUTINE Center(String,TWidth,Centered)

      CHARACTER*(*) STRING,CENTERED
      INTEGER WIDTH,TWidth
      INTEGER BlankColumns,l,Length

      Width=TWidth
      IF (Width.GT.LEN(Centered)) Width=LEN(Centered)

      CALL TrimCh(String)

      Length=LEN(String)
      DO L=Length,1,-1
        IF (String(L:L).NE.' ') GOTO 100
      END DO

      Centered(1:)=' '
      RETURN

100   IF (L.GT.Width) L=Width
      BlankColumns=(Width-L)/2

      Centered(1:BlankColumns)=' '

      IF (BlankColumns.NE.0) THEN

        Centered(BlankColumns:)=String(1:
     & MIN(LEN(CENTERED)-BlankColumns,L))

      ELSE
        Centered(1:)=String(1:)
      END IF

      RETURN
      END

C ********************************************************************** ChInp

      SUBROUTINE ChInp(PROMPT,VAR)

C This subroutine prints the PROMPT and obtains a character response in VAR

      COMMON /EDITSUBS/DataEntered

      CHARACTER*(*) PROMPT,VAR
      INTEGER Apostrophe,Length
      LOGICAL DataEntered

      CALL StringLength(Prompt,Length)
      WRITE(*,*) PROMPT(1:Length)
      READ(*,100) VAR
100   FORMAT(A)

      IF (Var.EQ.' ') THEN
        DataEntered=.FALSE.
      ELSE
```

```
          DataEntered=.TRUE.
        END IF

200     Apostrophe=INDEX(Var,'''')
        IF (Apostrophe.NE.0) THEN
          Var(Apostrophe:Apostrophe)=''
          GOTO 200
        END IF

        RETURN
        END

c ************************************************************************ ClrScr

        SUBROUTINE ClrScr

C This subroutine clears the digital terminal's screen

        BYTE ESCAPE/27/
        WRITE(*,100) ESCAPE
100     FORMAT (' ',A1,'[2J')

        RETURN
        END

C ************************************************************************ CodeI

        SUBROUTINE CodeI(List,NoOfElems,String)

        CHARACTER*(*) String
        CHARACTER*80 Dummy
        INTEGER List(*)
        INTEGER NoOfElems,Length,I,Comma
        LOGICAL Zeros

        Length=0
        DO I=1,NoOfElems
          Dummy(1:)=' '
          write(unit=Dummy(1:),FMT=10) List(I)
10        FORMAT (I,',')
          CALL RemoveBlanksFromCh(Dummy)

          Comma=INDEX(Dummy,',')

          String(Length+1:Length+Comma)=Dummy(1:Comma)
          Length=Length+Comma

        END DO

        IF (Length.GT.0) THEN
          String(Length:)=' '
        ELSE
          String=' '
        END IF

        RETURN
        END
```

```
C ************************************************************************ CodeR

      SUBROUTINE CodeR(List,NoOfElems,String)

      CHARACTER*(*) String
      REAL List(*)
      CHARACTER*80 Dummy
      INTEGER NoOfElems,Length,I,Comma,Dot,ELoc,J,NewELoc
      LOGICAL Zeros

      Length=0
      DO I=1,NoOfElems
        Dummy(1:)=' '
        write(unit=Dummy(1:),FMT=10) List(I)
10      FORMAT (G,',')
        CALL RemoveBlanksFromCh(Dummy)

        Comma=INDEX(Dummy,',')
        Dot=INDEX(Dummy,'.')
        ELoc=INDEX(Dummy,'E')
        IF (ELoc.EQ.0) THEN
          DO J=Comma-1,Dot+1,-1
              IF (Dummy(J:J).NE.'0') THEN
                Comma=J+1
                GOTO 20
            END IF
          END DO
          Comma=Dot
20        Dummy(Comma:)=','
        ELSE
          NewELoc=Eloc
          DO J=Eloc-1,Dot+1,-1
              IF (Dummy(J:J).NE.'0') THEN
                NewELoc=J+1
              GOTO 30
              END IF
          END DO
          NewELoc=Dot
30        Dummy(NewELoc:)=Dummy(ELoc:)
          Comma=INDEX(Dummy,',')
        END IF

        String(Length+1:Length+Comma)=Dummy(1:Comma)
        Length=Length+Comma

      END DO

      IF (Length.GT.0) THEN
        String(Length:)=' '
      ELSE
        String=' '
      END IF

      RETURN
      END

C ********************************************************************** CompareListI
```

```
      SUBROUTINE CompareListI(List1,NoOfElems1,
     & List2,NoOfElems2,Matches,NoOfMatches)

      INTEGER NoOfElems1,NoOfElems2,NoOfMatches
      INTEGER List1(*),List2(*),Matches(*),I,J

      NoOfMatches=0
      DO I=1,NoOfElems1
        DO J=1,NoOfElems2
          IF (List1(I).EQ.List2(J)) THEN
              NoOfMatches=NoOfMatches+1
              Matches(NoOfMatches)=List1(I)
          END IF
        END DO
      END DO

      RETURN
      END

C ******************************************************************* CompareListR

      SUBROUTINE CompareListR(List1,NoOfElems1,
     & List2,NoOfElems2,Matches,NoOfMatches)

      INTEGER NoOfElems1,NoOfElems2,NoOfMatches,I,J
      REAL List1(*),List2(*),Matches(*)

      NoOfMatches=0
      DO I=1,NoOfElems1
        DO J=1,NoOfElems2
          IF (List1(I).EQ.List2(J)) THEN
              NoOfMatches=NoOfMatches+1
              Matches(NoOfMatches)=List1(I)
          END IF
        END DO
      END DO

      RETURN
      END

C ******************************************************************* ConvertChToDP

      SUBROUTINE ConvertChToDP(String2,Re)

      CHARACTER*(*) String2
      CHARACTER*80 String
      INTEGER StringLength,NoCharacters,NoDots,NoSigns,NoE,ErrNo,I,J
      LOGICAL Quit
      DOUBLE PRECISION Re

      String=String2
      CALL UpperCaseCh(String)

      NoE=0
      NoSigns=0
      NoDots=0
      NoCharacters=0
      StringLength=LEN(String)
```

```
      I=0
      QUIT=.FALSE.
      DO WHILE (.NOT.QUIT)
        I=I+1
        IF (I.GT.StringLength) GOTO 3

        IF (((String(I:I).GE.'0').AND.(String(I:I).LE.'9'))
     & .OR.(String(I:I).EQ.'.').OR.(String(I:I).EQ.'+')
     & .OR.(String(I:I).EQ.'-').OR.(String(I:I).EQ.'E')) THEN

          IF ((String(I:I).EQ.'-').OR.(String(I:I).EQ.'+')) THEN
            NoSigns=NoSigns+1
            IF (NoSigns.GE.2) GOTO 5
          END IF

          IF (String(I:I).EQ.'.') THEN
            NoDots=NoDots+1
            IF (NoDots.GE.2) goto 5
          END IF

          IF (String(I:I).EQ.'E') THEN
            NoE=NoE+1
            IF (NoE.GE.2) GOTO 5
            NoSigns=0
              IF ((String(I+1:I+1).EQ.'-')
     & .OR.(String(I+1:I+1).EQ.'+')) THEN
                StringLength=MIN(I+3,StringLength)
              ELSE
                StringLength=MIN(I+2,StringLength)
              END IF
            IF (NoDots.EQ.0) THEN

                DO J=StringLength,I,-1
                  String(J+1:J+1)=String(J:J)
                END DO
                StringLength=StringLength+1
                NoCharacters=NoCharacters+1
                String(NoCharacters:NoCharacters)='.'
                NoDots=NoDots+1
                I=I+1

              END IF
          END IF

          NoCharacters=NoCharacters+1
          String(NoCharacters:NoCharacters)=String(I:I)

        END IF
3       IF (I.GE.StringLength) QUIT=.TRUE.
      END DO

5     DO I=StringLength,NoCharacters+1,-1
        String(I:I)=CHAR(0)
      END DO

      IF (NoCharacters.EQ.0) THEN
        Rc=0.
        RETURN
```

```
      END IF

      IF ((String(NoCharacters:NoCharacters).EQ.'+').OR.
     & (String(NoCharacters:NoCharacters).EQ.'-')) THEN
        NoCharacters=NoCharacters-1
      END IF

      IF (NoCharacters.EQ.0) THEN
        Re=0.
        RETURN
      END IF

      IF (NoDots.EQ.0) THEN
        NoDots=NoDots+1
        NoCharacters=NoCharacters+1
        String(NoCharacters:NoCharacters)='.'
      END IF

      IF (String(1:1).EQ.'.') THEN
        DO J=NoCharacters,1,-1
          String(J+1:J+1)=String(J:J)
        END DO
        NoCharacters=NoCharacters+1
        String(1:1)='0'
      END IF

      IF (((String(1:1).EQ.'-').OR.(String(1:1).EQ.'+'))
     & .AND.(String(2:2).EQ.'.')) THEN
        DO J=NoCharacters,2,-1
          String(J+1:J+1)=String(J:J)
        END DO
        NoCharacters=NoCharacters+1
        String(2:2)='0'
      END IF

      IF (NoE.EQ.1) THEN

        IF (String(1:1).EQ.'E') THEN
          DO I=NoCharacters,1,-1
            String(I+2:I+2)=String(I:I)
          END DO
          String(1:1)='1'
          String(2:2)='.'
          NoCharacters=NoCharacters+2
        END IF

        IF (String(NoCharacters:NoCharacters).EQ.'E') THEN
          NoCharacters=NoCharacters-1
          NoE=NoE-1
        END IF

      END IF

      DECODE(NoCharacters,10,String,IOSTAT=ErrNo,ERR=20) Re
10    FORMAT(G)
20    IF (ErrNo.GT.0) Re=1.E+37

      RETURN
```

```
      END

C ************************************************************** ConvertChToI

      SUBROUTINE ConvertChToI(String2,Int)

      CHARACTER*(*) String2
      CHARACTER*80 String
      INTEGER Int,StringLength,NoCharacters,Sighn,I

      String=String2
      NoCharacters=0
      StringLength=LEN(String)
      Sighn=1

      DO I=1,StringLength
        IF (String(I:I).EQ.'-') Sighn=-Sighn
        IF ((String(I:I).GE.'0').AND.(String(I:I).LE.'9')) THEN
          NoCharacters=NoCharacters+1
          String(NoCharacters:NoCharacters)=String(I:I)
        END IF
        IF (String(I:I).EQ.'.') GOTO 5
      END DO

5     DO I=StringLength,NoCharacters+1,-1
        String(I:I)=CHAR(0)
      END DO

      IF (NoCharacters.GT.9) NoCharacters=9

      DECODE(NoCharacters,10,String) Int
10    FORMAT(I<NoCharacters>)

      Int=Sighn*Int

      RETURN
      END

C ************************************************************** ConvertChToR

      SUBROUTINE ConvertChToR(String,Re)

      CHARACTER*(*) String
      REAL Re
      REAL*8 Dp

      CALL ConvertChToDP(String,Dp)
      Re=Dp

      RETURN
      END

C ************************************************************** ConvertDPToWords

      SUBROUTINE ConvertDPToWords(Number,Words)

C Include file:   Numwords.inc
```

```
C Written by:    Ken Jordan              Date:    7/12/85
C Approved by:                           Date:

C This file contains declarations and data statements for variables
C useful for converting numbers to words.

C ---------------------------------------------------------------------
C VARIABLES

C One_to_20$()          The words for one to twenty.
C Tea$()                The words for ten to ninety.
C Hundred$              'Hundred'
C Thousand$             'Thousand'

C ---------------------------------------------------------------------
C Declarations

      CHARACTER*10 One_to_20$(19)/' One',' Two',' Three',' Four',
     & ' Five',' Six',' Seven',' Eight',' Nine',' Ten',' Eleven',
     & ' Twelve',' Thirteen',' Fourteen',' Fifteen',' Sixteen',
     & ' Seventeen',' Eighteen',' Nineteen'/

      CHARACTER*8 Tea$(9)/' Ten',' Twenty',' Thirty',' Forty',
     & ' Fifty',' Sixty',' Seventy',' Eighty',' Ninety'/

      CHARACTER*7 Hundred$/' Hundred'/
      CHARACTER*9 Thousand$/' Thousand'/

      INTEGER WordLoc,I,Space
      DOUBLE PRECISION Number,Amt
      CHARACTER*80 WordList(10)
      CHARACTER*(*) Words
      LOGICAL Thou_flag

      Thou_flag=.FALSE.
      WordLoc=0

      DO I=1,10
        WordList(I)=' '
      END DO

      Amt=Number
      IF  (Amt.GE.1E5) THEN
        WordLoc=INT(Amt/1E5)
        WordList(10)=One_to_20$(WordLoc)
        WordList(9)=Hundred$
        Amt=Amt-WordLoc*1E5
        Thou_flag=.TRUE.
      END IF

      IF (Amt.GE.20000) THEN
        WordLoc=INT(Amt/1E4)
        Thou_flag=.TRUE.
        WordList(8)=Tea$(WordLoc)
        Amt=Amt-WordLoc*1E4
      END IF

      IF  (Amt.GE.1000) THEN
```

```
        Thou_flag=.TRUE.
        WordLoc=INT(Amt/1000)
        WordList(7)=One_to_20$(WordLoc)
        Amt=Amt-WordLoc*1000
      END IF

      IF (Thou_flag) WordList(6)=Thousand$

      IF (Amt.GE.100) THEN
        WordLoc=INT(Amt/100)
        WordList(5)=One_to_20$(WordLoc)
        WordList(4)=Hundred$
        Amt=Amt-WordLoc*100
      END IF

      IF (Amt.GE.20) THEN
        WordLoc=INT(Amt/10)
        WordList(3)=Tea$(WordLoc)
        Amt=Amt-WordLoc*10
      END IF

      IF  (Amt.GE.1) THEN
        WordLoc=INT(Amt)
        WordList(2)=One_to_20$(WordLoc)
        Amt=Amt-WordLoc
      END IF

C Print out the pay in words

      CALL PRoundDP(Amt*100,0,Amt)

      WRITE(UNIT=Words,FMT=10) INT(Amt)
10    FORMAT(' and ',I2,'/100 Dollars')

      DO I=2,10
        IF (WordList(I).NE.' ') THEN
          CALL RemoveBlanksFromCh(WordList(I))
          Space=INDEX(WordList(I),' ')
          Words(1:)=WordList(I)(1:Space)//Words(1:)
        END IF
      END DO

      RETURN
      END

C ******************************************************************* ConvertDtToI

      SUBROUTINE ConvertDtToI(Date$,Mo,Da,Yr)

      CHARACTER*(*) Date$
      INTEGER Mo,Da,Yr

      CALL ConvertChToI(Date$(1:2),Mo)
      CALL ConvertChToI(Date$(4:5),Da)
      CALL ConvertChToI(Date$(7:8),Yr)

      RETURN
      END
```

```
C ************************************************************************ DPInp

      SUBROUTINE DPInp(Prompt,Var)

C This subroutine prints the Prompt and obtains a double precision Var
C from the user

      COMMON /EDITSUBS/DataEntered

      CHARACTER*(*) PROMPT
      DOUBLE PRECISION Var
      CHARACTER*80 Response
      LOGICAL DataEntered

      CALL ChInp(Prompt,Response)
      IF (Response.EQ.' ') THEN
        DataEntered=.FALSE.
      ELSE
        DataEntered=.TRUE.
      END IF
      CALL ConvertChToDP(Response,Var)

      RETURN
      END

C ********************************************************************** DateBend

      SUBROUTINE DateBend(Date$,Date)

      INTEGER Date,Year,Month,Day
      CHARACTER*8 Date$,Temp

      Temp=Date$(7:8)
      CALL ConvertChToI(Temp,Year)
      Temp=Date$(1:2)
      CALL ConvertChToI(Temp,Month)
      Temp=Date$(4:5)
      CALL ConvertChToI(Temp,Day)
      Date=Year*10000+Month*100+Day

      RETURN
      END

C ****************************************************************** DateUnBend

      SUBROUTINE DateUnBend(TDate,Date$)

      INTEGER Year,Date,Month,Day,Tdate
      CHARACTER*8 Date$

      IF (TDate.GT.999999) THEN
        DATE$='99/99/99'
        RETURN
      END IF

      Date=Tdate
      Year=Date/10000
      Date=Date-Year*10000
```

```
      Month=Date/100
      Day=Date-Month*100

      WRITE(Unit=Date$(1:),FMT=10) Month
10    FORMAT(I2)
      WRITE(Unit=Date$(4:),FMT=10) Day
      WRITE(UNIT=Date$(7:),FMT=10) Year

      Date$(3:3)='/'
      Date$(6:6)='/'

      RETURN
      END

C ********************************************************************** DecodeCh

      SUBROUTINE DecodeCh(String,Delimiter,List,NoOfElems,MaxNoOfElems)

      CHARACTER*(*) String,Delimiter,List(*)
      INTEGER NoOfElems,MaxNoOfElems,NoOfParans,Length,MaxLength
      INTEGER I,NoOfChar,MaxLen2
      LOGICAL Spaces,Word,Quotes,Skip

      Spaces=(Delimiter.EQ.' ')
      NoOfElems=0
      NoOfChar=0
      NoOfParans=0

      MaxLength=LEN(List(1))
      CALL StringLength(String,Length)
      MaxLen2=LEN(String)

      Word=.FALSE.
      Quotes=.FALSE.
      Skip=.FALSE.

      DO I=1,MaxNoOfElems
        List(I)(1:)=' '
      END DO

      DO I=1,Length

        IF ((String(I:MIN(I+1,MaxLen2)).EQ.'''''')
     & .OR.(String(I:MIN(I+1,MaxLen2)).EQ.'''')) THEN
          Skip=.TRUE.
        ELSE IF ((String(I:I).EQ.'''').OR.(String(I:I).EQ.''')) THEN
          IF (.NOT.Skip) Quotes=.NOT.Quotes
          Skip=.FALSE.
        ELSE IF (.NOT.Quotes) THEN
          IF (String(I:I).EQ.'(') NoOfParans=NoOfParans+1
          IF (String(I:I).EQ.')') NoOfParans=NoOfParans-1
        END IF

        IF (Word) THEN
          IF ((String(I:I).EQ.Delimiter(1:1)).AND.
     & (.NOT.Quotes).AND.(NoOfParans.LE.0)) THEN
              Word=.FALSE.
          ELSE
```

```
            NoOfChar=NoOfChar+1
          IF (NoOfChar.LE.MaxLength) THEN
              List(NoOfElems)(NoOfChar:NoOfChar)=String(I:I)
          ELSE

            WRITE(*,*) 'Warning: too many characters.  The'
            WRITE(*,*) 'following was ignored:'
            WRITE(*,*) String(I:)
            WRITE(*,*) 'The following was kept:'
            WRITE(*,*) List(NoOfElems)
            WRITE(*,*) ' '
            RETURN

          END IF

        END IF
      ELSE

        IF (String(I:I).NE.Delimiter(1:1)) THEN

            Word=.TRUE.
            NoOfChar=1
            IF (NoOfElems.GE.MaxNoOfElems) THEN
              WRITE(*,*) 'WARNING, TOO MANY ELEMENTS IN LIST'
              WRITE(*,*) 'THE FOLLOWING WAS IGNORED:',String(I:)
              GOTO 100
            END IF
            NoOfElems=NoOfElems+1
            List(NoOfElems)(NoOfChar:NoOfChar)=String(I:I)

        ELSE

            IF (.NOT.Spaces) THEN
              IF (NoOfElems.GE.MaxNoOfElems) THEN
                WRITE(*,*) 'WARNING, TOO MANY ELEMENTS IN LIST'
                WRITE(*,*) 'THE FOLLOWING WAS IGNORED:',String(I:)
                GOTO 100
              END IF
              NoOfElems=NoOfElems+1
              List(NoOfElems)=' '
            END IF

        END IF
      END IF
    END DO

100 DO I=1,NoOfElems
      CALL RemoveBlanksFromCh(List(I))
    END DO

    RETURN
    END

C ******************************************************************** DecodeDP

    SUBROUTINE DecodeDP(String,List,NoOfElems,MaxNoOfElems)

    CHARACTER*(*) String
```

```
      CHARACTER*80 Dummy
      DOUBLE PRECISION List(*)
      INTEGER NoOfElems,MaxNoOfElems
      INTEGER CommaLoc,Length

      NoOfElems=0
      CommaLoc=1
      CALL StringLength(String,Length)
      String(Length+1:Length+1)=','

      DO WHILE (CommaLoc.GT.0)
        CommaLoc=INDEX(String,',')
        IF (CommaLoc.GT.0) THEN
          IF (NoOfElems.LT.MaxNoOfElems) THEN

              IF (CommaLoc.EQ.1) THEN
                Dummy='0'
              ELSE
                Dummy=String(1:CommaLoc-1)
              END IF

              String=String(CommaLoc+1:)
            NoOfElems=NoOfElems+1
              CALL ConvertChToDP(Dummy,List(NoOfElems))

          ELSE

              WRITE(*,*) 'Warning: the following information was'
              WRITE(*,*) 'lost:',String
              CALL PauseCrt
              RETURN

          END IF
        END IF
      END DO

      RETURN
      END

C ************************************************************************ DecodeI

      SUBROUTINE DecodeI(String,List,NoOfElems,MaxNoOfElems)

      PARAMETER MaxListLen=200
      CHARACTER*(*) String
      INTEGER List(*)
      INTEGER NoOfElems,MaxNoOfElems,I
      DOUBLE PRECISION ListDP(MaxListLen)

      IF (MaxNoOfElems.GT.MaxListLen) THEN

        WRITE(*,*) 'Warning: Programming error.  DecodeI has '
        WRITE(*,*) 'been called with MaxNoOfElems greater than'
        WRITE(*,*) 'MaxListLen.  MaxListLen in subroutine DecodeI'
        WRITE(*,*) 'will have to be increased to ',MaxNoOfElems
        WRITE(*,*) 'and the accounting programs re-compiled'
        WRITE(*,*) '(Note: DecodeI is in file SUBS.FOR)'
        STOP
```

```
      ELSE

        CALL DecodeDP(String,ListDP,NoOfElems,MaxListLen)
        DO I=1,NoOfElems
          List(I)=INT(ListDP(I))
        END DO

      END IF

      RETURN
      END

C ********************************************************************** DecodeR

      SUBROUTINE DecodeR(String,List,NoOfElems,MaxNoOfElems)

      PARAMETER MaxListLen=200
      CHARACTER*(*) String
      REAL List(*)
      INTEGER NoOfElems,MaxNoOfElems,I
      DOUBLE PRECISION ListDP(MaxListLen)

      IF (MaxNoOfElems.GT.MaxListLen) THEN

        WRITE(*,*) 'Warning: Programming error.  DecodeR has '
        WRITE(*,*) 'been called with MaxNoOfElems greater than'
        WRITE(*,*) 'MaxListLen.  MaxListLen in subroutine DecodeR'
        WRITE(*,*) 'will have to be increased to ',MaxNoOfElems
        WRITE(*,*) 'and the accounting programs re-compiled'
        WRITE(*,*) '(Note: DecodeR is in file SUBS.FOR)'
        STOP

      ELSE

        CALL DecodeDP(String,ListDP,NoOfElems,MaxListLen)
        DO I=1,NoOfElems
          List(I)=ListDP(I)
        END DO

      END IF

      RETURN
      END

C ********************************************************************** DpXChng

      SUBROUTINE DpXCHNG(Dp1,Dp2)

      REAL*8 Dp1,Dp2,Dp3

      Dp3 = Dp1
      Dp1 = Dp2
      Dp2 = Dp3

      RETURN
      END

C ********************************************************************** DtInp
```

```
      SUBROUTINE DtInp(Prompt,Date$,Date)

      CHARACTER*(*) Prompt,Date$
      INTEGER Date,Day,Month,Year,I
      CHARACTER*2 Dummy
      LOGICAL Err

6870  WRITE(*,*) 'Enter date as MM/DD/YY'
      CALL ChInp(Prompt,Date$)
      IF (Date$.EQ.' ') THEN
        Date=0
        RETURN
      END IF

      IF  (LEN(Date$).EQ.8) GO TO 6820

      WRITE (*,6760) Date$
6760  FORMAT (' Error: date is not 8 characters long: ',A)
      GO TO 6870

6820  IF  ((Date$(3:3).EQ.'/').AND.(Date$(6:6).EQ.'/')) GO TO 6890

      WRITE (*,6830) Date$
6830  FORMAT (' Error: Invalid date: ',A,/,
     1 ' 3rd and 6th characters must be '/'')
      GO TO 6870

6890  Err=.FALSE.
      DO I=1,8
        IF  ((I.NE.3).AND.(I.NE.6)) THEN
        IF  ((Date$(I:I).LT.'0').AND.(Date$(I:I).GT.'9')) THEN
          IF (.NOT.Err) THEN
            WRITE(*,*) 'Error: nonnumeric characters at positions:'
          END IF
          Err=.TRUE.
          WRITE(*,*) I
        END IF
        END IF
      END DO

      IF  (Err) GO TO 6870

      Dummy=Date$(1:2)
      CALL ConvertChToI(Dummy,Month)
      IF  ((Month.GE.1).AND.(Month.LE.12)) GO TO 7080

      WRITE (*,6990) Month
6990  FORMAT (' Error: impossible month: ',I5)

      GO TO 6870

7080  Dummy=Date$(4:5)
      CALL ConvertChToI(Dummy,Day)
      IF  ((Day.GE.1).AND.(Day.LE.31)) GO TO 7140
      WRITE (*,7090) Day
7090  FORMAT (' Error: impossible day: ',I5)

      GO TO 6870
```

```
7140    Dummy=Date$(7:8)
        CALL ConvertChToI(Dummy,Year)

        Date=10000*Year+100*Month+Day

        RETURN
        END

C ************************************************************************ EditCh

        SUBROUTINE EditCh(Prompt,Var)

C This subroutine prints the Prompt and the old value of Var, and obtains
C a new value of Var from the user

        COMMON /EDITSUBS/DataEntered

        CHARACTER*(*) Prompt,Var
        CHARACTER*80 OVar
        INTEGER VarLen,Length
        LOGICAL DataEntered

        VarLen=LEN(Var)
        OVar=Var
        CALL StringLength(OVar,Length)
        IF (Length.GT.0) THEN
          WRITE(*,*) 'Old Value was:',OVar(1:Length)
          WRITE(*,*) '<cr> keeps old value'
        END IF

        CALL ChInp(Prompt,Var)
        IF (.NOT.DataEntered) Var=OVar(1:VarLen)

        RETURN
        END

C ************************************************************************ EditDP

        SUBROUTINE EditDP(Prompt,Var)

C This subroutine prints the Prompt and the old value of Var, and obtains
C a new value of Var from the user

        COMMON /EDITSUBS/DataEntered

        CHARACTER*(*) Prompt
        DOUBLE PRECISION Var,OVar
        LOGICAL DataEntered

        OVar=Var
        IF (OVar.NE.0) THEN
          WRITE(*,*) 'Old Value was:',OVar
          WRITE(*,*) '<cr> keeps old value'
        END IF

        CALL DPInp(Prompt,Var)
        IF(.NOT.DataEntered) Var=OVar
```

```
      RETURN
      END

C ********************************************************************** EditDtCh

      SUBROUTINE EditDtCh(Prompt,Date$,Date)

      CHARACTER*(*) Prompt,Date$
      INTEGER Date,Day,Month,Year,I
      CHARACTER Dummy*2,ODate$*8
      LOGICAL Err

      ODate$=Date$

6870  WRITE(*,*) 'Enter date as MM/DD/YY'
      Date$=ODate$
      CALL EditCh(Prompt,Date$)

      IF  (LEN(Date$).EQ.8) GO TO 6820

      WRITE (*,6760) Date$
6760  FORMAT (' Error: date is not 8 characters long: ',A)
      GO TO 6870

6820  IF  ((Date$(3:3).EQ.'/').AND.(Date$(6:6).EQ.'/')) GO TO 6890

      WRITE (*,6830) Date$
6830  FORMAT (' Error: Invalid date: ',A,/,
     1 ' 3rd and 6th characters must be '/'')
      GO TO 6870

6890  Err=.FALSE.
      DO I=1,8
        IF  ((I.NE.3).AND.(I.NE.6)) THEN
        IF  ((Date$(I:I).LT.'0').AND.(Date$(I:I).GT.'9')) THEN
          IF (.NOT.Err) THEN
            WRITE(*,*) 'Error: nonnumeric characters at positions:'
          END IF
          Err=.TRUE.
          WRITE(*,*) I
        END IF
        END IF
      END DO

      IF  (Err) GO TO 6870

      Dummy=Date$(1:2)
      CALL ConvertChToI(Dummy,Month)
      IF  ((Month.GE.1).AND.(Month.LE.12)) GO TO 7080

      WRITE (*,6990) Month
6990  FORMAT (' Error: impossible month: ',I5)

      GO TO 6870

7080  Dummy=Date$(4:5)
      CALL ConvertChToI(Dummy,Day)
      IF  ((Day.GE.1).AND.(Day.LE.31)) GO TO 7140
```

```
      WRITE (*,7090) Day
7090  FORMAT (' Error: impossible day: ',I5)

      GO TO 6870

7140  Dummy=Date$(7:8)
      CALL ConvertChToI(Dummy,Year)

      Date=10000*Year+100*Month+Day

      RETURN
      END

C ************************************************************************ EditDtI

      SUBROUTINE EditDtI(Prompt,Date$,Date)

      CHARACTER*(*) Prompt,Date$
      INTEGER Date
      CHARACTER*2 Dummy

      CALL DateUnBend(Date,Date$)
      CALL EditDtCh(Prompt,Date$,Date)

      RETURN
      END

C ************************************************************************ EditI

      SUBROUTINE EditI(Prompt,Var)

C This subroutine prints the Prompt and the old value of Var, and obtains
C a new value of Var from the user

      COMMON /EDITSUBS/DataEntered

      CHARACTER*(*) Prompt
      INTEGER Var,OVar
      LOGICAL DataEntered

      OVar=Var
      IF (OVar.NE.0) THEN
        WRITE(*,*) 'Old Value was:',OVar
        WRITE(*,*) '<cr> keeps old value'
      END IF
      CALL IInp(Prompt,Var)

      IF(.NOT.DataEntered) Var=OVar

      RETURN
      END

C ************************************************************************ EditR

      SUBROUTINE EditR(Prompt,Var)

C This subroutine prints the Prompt and the old value of Var, and obtains
C a new value of Var from the user
```

```
      COMMON /EDITSUBS/DataEntered

      CHARACTER*(*) Prompt
      REAL Var,OVar
      LOGICAL DataEntered

      OVar=Var
      IF (OVar.NE.0) THEN
        WRITE(*,*) 'Old Value was:',OVar
        WRITE(*,*) '<cr> keeps old value'
      END IF

      CALL RInp(Prompt,Var)
      IF(.NOT.DataEntered) Var=OVar

      RETURN
      END

C ********************************************************************** EditYorN

      SUBROUTINE EditYorn(Prompt$,Yes)

      LOGICAL Yes,No
      CHARACTER Prompt$*(*),Response*1

C This subprogram obtains a Y or N response from the user.

C Prompt$ is the prompt to be used.
C Yes is the logical variable

      IF (Yes) THEN
        Response='Y'
      ELSE
        Response='N'
      END IF

      Yes=.FALSE.
      No=.FALSE.
      DO WHILE(.NOT.(Yes.OR.No))

        WRITE (*,*) Prompt$
        CALL EditCh('(Y or N)?',Response)

        Yes=(Response.EQ.'Y').OR.(Response.EQ.'y')
        No=(Response.EQ.'N').OR.(Response.EQ.'n')

        IF (.NOT.(Yes.OR.No)) THEN

          WRITE(*,100) Response
100       FORMAT (' ','Error: response ',A,' is not Y or N.')

        END IF

      END DO

      RETURN
      END
```

```
C ******************************************************************** FileDelete

      SUBROUTINE FileDelete(FileName)

      CHARACTER*(*) FileName
      CHARACTER*80 DFile
      LOGICAL FExist

      DFile=FileName
      CALL RemoveBlanksFromCh(DFile)
      INQUIRE(FILE=DFile,EXIST=FExist)

      IF (FExist) THEN
        CALL LIB$DELETE_FILE(DFile//';*')
D     ELSE
D       WRITE(*,*) 'Warning: Attempt to delete a non-existent file.'
D       WRITE(*,*) 'File name:',FileName
      END IF

      RETURN
      END

C ******************************************************************** FilePurge

      SUBROUTINE FilePurge(FileName)

      CHARACTER*(*) FileName
      LOGICAL FExist

      INQUIRE(FILE=FileName,Exist=FExist)

      IF (FExist) THEN
        CALL FileRename(FileName,'TEMP_'//FileName)
        CALL FileDelete(FileName)
        CALL FileRename('TEMP_'//FileName,FileName)
D     ELSE
D       WRITE(*,*) 'Warning: Attempt to Purge a nonexistent file.'
D       WRITE(*,*) 'File name:',FileName
      END IF

      RETURN
      END

C ******************************************************************** FileRename

      SUBROUTINE FileRename(OldName,NewName)

      CHARACTER*(*) OldName,NewName
      CHARACTER*80 DOld,DNew
      LOGICAL FExist

      DOld=OldName
      CALL RemoveBlanksFromCh(DOld)
      DNew=NewName
      CALL RemoveBlanksFromCh(DNew)

      INQUIRE(FILE=DOld,EXIST=FExist)
```

```
      IF (FExist) THEN
        CALL LIB$RENAME_FILE(DOld,DNew)
      ELSE
        WRITE(*,*) 'Warning: attempting to rename a nonexistent file.'
        WRITE(*,*) 'Old Name:',OldName
        WRITE(*,*) 'New Name:',NewName
        CALL PauseCrt
      END IF

      RETURN
      END

C ************************************************************** FindMatchesCh

      SUBROUTINE FindMatchesCh(Matrix,NElements,MatchElement,
     & MatchLocs,Found,NoLocs)

      CHARACTER*(*) Matrix(*),MatchElement
      INTEGER NElements,MatchLocs(*),Length,NoLocs,I
      LOGICAL Found

      Found=.FALSE.
      NoLocs=0

      IF (LEN(Matrix(1)).GT.LEN(MatchElement)) THEN
        Length=LEN(MatchElement)
      ELSE
        Length=LEN(Matrix(1))
      END IF

      DO I=1,NElements
        IF (Matrix(I)(1:Length).EQ.MatchElement(1:Length)) THEN
          Found=.TRUE.
          NoLocs=NoLocs+1
          MatchLocs(NoLocs)=I
        END IF
      END DO

      RETURN
      END

C ************************************************************** FindMatchesDP

      SUBROUTINE FindMatchesDP(Matrix,NElements,MatchElement,
     & MatchLocs,Found,NoLocs)

      DOUBLE PRECISION Matrix(*),MatchElement
      INTEGER NElements,MatchLocs(*),NoLocs,I
      LOGICAL Found

      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I).EQ.MatchElement) THEN
          Found=.TRUE.
          NoLocs=NoLocs+1
          MatchLocs(NoLocs)=I
```

```
        END IF
      END DO

      RETURN
      END

C ******************************************************************** FindMatchesI

      SUBROUTINE FindMatchesI(Matrix,NElements,MatchElement,
     & MatchLocs,Found,NoLocs)

      INTEGER Matrix(*),MatchElement
      INTEGER NElements,MatchLocs(*),NoLocs,I
      LOGICAL Found

      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I).EQ.MatchElement) THEN
          Found=.TRUE.
          NoLocs=NoLocs+1
          MatchLocs(NoLocs)=I
        END IF
      END DO

      RETURN
      END

C ******************************************************************** FindMatchesR

      SUBROUTINE FindMatchesR(Matrix,NElements,MatchElement,
     & MatchLocs,Found,NoLocs)

      REAL Matrix(*),MatchElement
      INTEGER NElements,MatchLocs(*),NoLocs,I
      LOGICAL Found

      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I).EQ.MatchElement) THEN
          Found=.TRUE.
          NoLocs=NoLocs+1
          MatchLocs(NoLocs)=I
        END IF
      END DO

      RETURN
      END

C ******************************************************************** FindMaxCh

      SUBROUTINE FindMaxCh(Matrix,NoOfElems,MaxCh,MaxLoc)

      CHARACTER*(*) Matrix(*),MaxCh
      INTEGER MaxLoc,NoOfElems,I
```

```
      IF (NoOfElems.LT.1) THEN
        MaxLoc=0
        MaxCh=' '
        RETURN
      END IF

      MaxLoc=1
      MaxCh=' '

      DO I=2,NoOfElems
        IF (Matrix(I).GT.MaxCh) THEN
          MaxLoc=I
          MaxCh=Matrix(I)
        END IF
      END DO

      RETURN
      END

C ********************************************************************** FindMaxDt

      SUBROUTINE FindMaxDt(Dates,NoOfDates,Date,Loc)

      CHARACTER*8 Dates(*)
      INTEGER NoOfDates,Date,Loc,TDate,I

      IF (NoOfDates.LE.0) THEN
        Date=0
        Loc=0
        RETURN
      END IF

      CALL DateBend(Dates(1),Date)
      Loc=1

      DO I=2,NoOfDates
        CALL DateBend(Dates(I),TDate)
        IF (TDate.GT.Date) THEN
          Loc=I
          Date=TDate
        END IF
      END DO

      RETURN
      END

C ********************************************************************** FindMaxI

      SUBROUTINE FindMaxI(Matrix,NoOfElems,MaxI,MaxLoc)

      INTEGER MaxLoc,NoOfElems,Matrix(*),MaxI,I

      IF (NoOfElems.LT.1) THEN
        MaxLoc=0
        MaxI=0
        RETURN
      END IF
```

```
      MaxLoc=1
      MaxI=Matrix(1)

      DO I=2,NoOfElems
        IF (Matrix(I).GT.MaxI) THEN
          MaxLoc=I
          MaxI=Matrix(I)
        END IF
      END DO

      RETURN
      END

C ********************************************************************** FindMinCh

      SUBROUTINE FindMinCh(Matrix,NoOfElems,MinCh,MinLoc)

      CHARACTER*(*) Matrix(*),MinCh
      INTEGER MinLoc,NoOfElems,I

      IF (NoOfElems.LT.1) THEN
        MinLoc=0
        MinCh=' '
        RETURN
      END IF

      MinLoc=1
      MinCh=' '

      DO I=2,NoOfElems
        IF (Matrix(I).LT.MinCh) THEN
          MinLoc=I
          MinCh=Matrix(I)
        END IF
      END DO

      RETURN
      END

C ********************************************************************** FindMinDt

      SUBROUTINE FindMinDt(Matrix,NoOfElems,MinDt,MinLoc)

      CHARACTER*(*) Matrix(*),MinDt
      INTEGER MinLoc,NoOfElems,Elem1,Elem2,I

      IF (NoOfElems.LT.1) THEN
        MinLoc=0
        MinDt='00/00/00'
        RETURN
      END IF

      MinLoc=1
      MinDt=Matrix(1)
      CALL DateBend(MinDt,Elem2)

      DO I=2,NoOfElems
        CALL DateBend(Matrix(I),Elem1)
```

```
      IF (Elem1.LT.Elem2) THEN
        MinLoc=I
        MinDt=Matrix(I)
        Elem2=Elem1
      END IF
    END DO

    RETURN
    END

C ********************************************************************** FindMinI

    SUBROUTINE FindMinI(Matrix,NoOfElems,MinI,MinLoc)

    INTEGER MinLoc,NoOfElems,Matrix(*),MinI,I

    IF (NoOfElems.LT.1) THEN
      MinLoc=0
      MinI=0
      RETURN
    END IF

    MinLoc=1
    MinI=Matrix(1)

    DO I=2,NoOfElems
      IF (Matrix(I).LT.MinI) THEN
        MinLoc=I
        MinI=Matrix(I)
      END IF
    END DO

    RETURN
    END

C ********************************************************************** FraDP

    SUBROUTINE FraDP(Number,Fraction)

    DOUBLE PRECISION Number,Fraction

    Fraction=Number-INT(Number)

    RETURN
    END

C ********************************************************************** GetAxis

    SUBROUTINE GetAxis(AxisName,AxisLabel,AxisMin,AxisMax,AxisLog)

    CHARACTER*(*) AxisName,AxisLabel
    LOGICAL AxisLog
    REAL AxisMin,AxisMax

    WRITE(*,*) 'If you don''t enter an axis label, axis will NOT'
    WRITE(*,*) 'be plotted!'
    CALL EditCh(AxisName//' Axis Label?',AxisLabel)
```

```
      CALL EditR(AxisName//' Minumum',AxisMin)
      CALL EditR(AxisName//' Maximum',AxisMax)

      IF ((AxisMin.LE.0).OR.(AxisMax.LE.0)) AxisLog=.FALSE.

      IF (AxisLog)
     & CALL EditYorN('Take Log of '//AxisName//' Values?',AxisLog)

      IF (AxisLog) THEN
        AxisMin=LOG(AxisMin)
        AxisMax=LOG(AxisMax)
      END IF

      RETURN
      END

C ******************************************************************** GetFileName

      SUBROUTINE GetFileName(Prompt,FileName)

      INCLUDE 'IO.INC'

      INTEGER Length
      CHARACTER*(*) Prompt,FileName
      LOGICAL FileExist

      IF (LunO.EQ.0) LunO=6

      FileExist=.FALSE.
      DO WHILE (.NOT.FileExist)

        CALL EditCh(Prompt,FileName)

        CALL TrimCh(FileName)
        CALL StringLength(FileName,Length)
        IF (Length.EQ.0) RETURN

        INQUIRE(FILE=FileName,EXIST=FileExist)
        IF (.NOT.FileExist) WRITE(LunO,20) FileName
      END DO

      RETURN
20    FORMAT(X,'Warning: Filename ',A<Length>,' doesn''t exist.'/,
     & X,'        Please try again.'/)

      END

C ******************************************************************** GetPlotter

      SUBROUTINE GetPlotter(TermType)

      CHARACTER*80 TermTitle/'Plotting Device Selection'/,
     & TermTxt(8)/
     1 'Code  Plotting Device',
     2 ' ',
     3 ' IMA  Imagen Printer',
     4 ' TAL  Talaris Printer',
     5 ' TEK  Tektronics 4010 Terminal',
```

```
     6 ' VT   VT240 Terminal',
     7 ' SEL  Selanar 100 Terminal',
     8 ' LAS  Apple Laserwriter'/,
     & TermPrompt/'Plotting Device Code?'/
         CHARACTER*3 TermResps(6)/'IMA','TAL','VT','TEK','SEL','LAS'/,
     & TermDefResp/'VT'/,TermType

         INTEGER TermNTxt/8/,TermNResps/6/

         CALL Menu(TermTitle,TermTxt,TermNTxt,TermPrompt,
     & TermResps,TermNResps,TermDefResp,TermType)

         RETURN
         END

C ******************************************************************** HeapSortDP

         SUBROUTINE HeapSortDP(N,Matrix)

C This subroutine will use the Heap Sort technique to sort Matrix in ascending
C numerical order. This subroutine is taken from Numerical Recipes: The
C Art of Scientific Computing by Press, Flannery, Teukolsky, and Vetterling,
C p. 231.

         INTEGER N,L,I,J,IL,IR
         REAL*8 Matrix(*),RRA

         IF (N.EQ.1) RETURN

         L=N/2+1
         IR=N

C The index L will be decremented from its initial value down to 1 during
C the "hiring" (heap creation) phase. Once it reaches 1, the index IR will
C be decremented from its initial value down to 1 during the "retirement
C and promotion" (heap selection) phase.

10       CONTINUE
           IF (L.GT.1) THEN

C This section of the IF-THEN-ELSE block is executed if we are still
C in the hiring phase.

             L=L-1
             RRA=Matrix(L)
           ELSE

C This section of the IF-THEN-ELSE block is executed if we are in the retirment
C and promotion phase. First we clear a space at the top of the array,
C and retire the top of the heap into it. Then we decrease the size
C of the corporation.

             RRA=Matrix(IR)
             Matrix(IR)=Matrix(1)
             IR=IR-1

             IF (IR.EQ.1) THEN
```

```
C This IF-THEN block is executed if the last promotion has been made.

            Matrix(1)=RRA
            RETURN
          END IF

        END IF

C Whether we are in the hiring phase or promtion phase, we here set up to
C sift down element RRa to its proper level.

        I=L
        J=L+L

        DO WHILE(J.LE.IR)

          IF (J.LT.IR) THEN
            IF (Matrix(J).LT.Matrix(J+1)) J=J+1
          END IF

          IF (RRA.LT.Matrix(J)) THEN
            Matrix(I)=Matrix(J)
            I=J
            J=J+J
          ELSE
            J=IR+1
          END IF

        END DO

        Matrix(I)=RRA

      GOTO 10

      END

C ************************************************************** HeapSortIndexCh

      SUBROUTINE HeapSortIndexCh(N,Matrix,Index)

C This subroutine will use the Heap Sort technique to sort Matrix in ascending
C numerical order. This subroutine is taken from Numerical Recipes: The
C Art of Scientific Computing by Press, Flannery, Teukolsky, and Vetterling,
C p. 233.

      INTEGER N,L,I,J,IL,IR,Index(*),Indx
      CHARACTER*(*) Matrix(*)

      DO I=1,N
        Index(I)=I
      END DO

      IF (N.EQ.1) RETURN

      L=N/2+1
      IR=N

C The index L will be decremented from its initial value down to 1 during
```

```
C the "hiring" (heap creation) phase.  Once it reaches 1, the index IR will
C be decremented from its initial value down to 1 during the "retirement
C and promotion" (heap selection) phase.

10      CONTINUE
          IF (L.GT.1) THEN

C This section of the IF-THEN-ELSE block is executed if we are still
C in the hiring phase.

            L=L-1
            Indx=Index(L)
            Matrix(N+1)=Matrix(Indx)
          ELSE

C This section of the IF-THEN-ELSE block is executed if we are in the retirment
C and promotion phase.  First we clear a space at the top of the array,
C and retire the top of the heap into it.  Then we decrease the size
C of the corporation.

            Indx=Index(IR)
            Matrix(N+1)=Matrix(Indx)
            Index(IR)=Index(1)
            IR=IR-1

            IF (IR.EQ.1) THEN

C This IF-THEN block is executed if the last promotion has been made.

              Index(1)=Indx
              RETURN
            END IF

          END IF

C Whether we are in the hiring phase or promtion phase, we here set up to
C sift down element RRa to its proper level.

          I=L
          J=L+L

          DO WHILE(J.LE.IR)

            IF (J.LT.IR) THEN
              IF (Matrix(Index(J)).LT.Matrix(Index(J+1))) J=J+1
            END IF

            IF (Matrix(N+1).LT.Matrix(Index(J))) THEN
              Index(I)=Index(J)
              I=J
              J=J+J
            ELSE
              J=IR+1
            END IF

          END DO

          Index(I)=Indx
```

```
      GOTO 10

      END

C ************************************************************** HeapSortIndexDP

      SUBROUTINE HeapSortIndexDP(N,Matrix,Index)

C This subroutine will use the Heap Sort technique to sort Matrix in ascending
C numerical order. This subroutine is taken from Numerical Recipes: The
C Art of Scientific Computing by Press, Flannery, Teukolsky, and Vetterling,
C p. 233.

      INTEGER N,L,I,J,IL,IR,Index(*),Indx
      REAL*8 Matrix(*),RRA

      DO I=1,N
        Index(I)=I
      END DO

      IF (N.EQ.1) RETURN

      L=N/2+1
      IR=N

C The index L will be decremented from its initial value down to 1 during
C the "hiring" (heap creation) phase. Once it reaches 1, the index IR will
C be decremented from its initial value down to 1 during the "retirement
C and promotion" (heap selection) phase.

10    CONTINUE
        IF (L.GT.1) THEN

C This section of the IF-THEN-ELSE block is executed if we are still
C in the hiring phase.

          L=L-1
          Indx=Index(L)
          RRA=Matrix(Indx)
        ELSE

C This section of the IF-THEN-ELSE block is executed if we are in the retirment
C and promotion phase. First we clear a space at the top of the array,
C and retire the top of the heap into it. Then we decrease the size
C of the corporation.

          Indx=Index(IR)
          RRA=Matrix(Indx)
          Index(IR)=Index(1)
          IR=IR-1

          IF (IR.EQ.1) THEN

C This IF-THEN block is executed if the last promotion has been made.

            Index(1)=Indx
            RETURN
          END IF
```

```
          END IF

C Whether we are in the hiring phase or promtion phase, we here set up to
C sift down element RRa to its proper level.

          I=L
          J=L+L

          DO WHILE(J.LE.IR)

            IF (J.LT.IR) THEN
              IF (Matrix(Index(J)).LT.Matrix(Index(J+1))) J=J+1
            END IF

            IF (RRA.LT.Matrix(Index(J))) THEN
              Index(I)=Index(J)
              I=J
              J=J+J
            ELSE
              J=IR+1
            END IF

          END DO

          Index(I)=Indx

        GOTO 10

        END

C ************************************************************** HeapSortIndexR

        SUBROUTINE HeapSortIndexR(N,Matrix,Index)

C This subroutine will use the Heap Sort technique to sort Matrix in ascending
C numerical order. This subroutine is taken from Numerical Recipes: The
C Art of Scientific Computing by Press, Flannery, Teukolsky, and Vetterling,
C p. 233.

        INTEGER N,L,I,J,IL,IR,Index(*),Indx
        REAL Matrix(*),RRA

        DO I=1,N
          Index(I)=I
        END DO

        IF (N.EQ.1) RETURN

        L=N/2+1
        IR=N

C The index L will be decremented from its initial value down to 1 during
C the "hiring" (heap creation) phase. Once it reaches 1, the index IR will
C be decremented from its initial value down to 1 during the "retirement
C and promotion" (heap selection) phase.

10      CONTINUE
          IF (L.GT.1) THEN
```

```
C This section of the IF-THEN-ELSE block is executed if we are still
C in the hiring phase.

          L=L-1
          Indx=Index(L)
          RRA=Matrix(Indx)
        ELSE

C This section of the IF-THEN-ELSE block is executed if we are in the retirment
C and promotion phase.  First we clear a space at the top of the array,
C and retire the top of the heap into it.  Then we decrease the size
C of the corporation.

          Indx=Index(IR)
          RRA=Matrix(Indx)
          Index(IR)=Index(1)
          IR=IR-1

          IF (IR.EQ.1) THEN

C This IF-THEN block is executed if the last promotion has been made.

            Index(1)=Indx
            RETURN
          END IF

        END IF

C Whether we are in the hiring phase or promtion phase, we here set up to
C sift down element RRa to its proper level.

        I=L
        J=L+L

        DO WHILE(J.LE.IR)

          IF (J.LT.IR) THEN
            IF (Matrix(Index(J)).LT.Matrix(Index(J+1))) J=J+1
          END IF

          IF (RRA.LT.Matrix(Index(J))) THEN
            Index(I)=Index(J)
            I=J
            J=J+J
          ELSE
            J=IR+1
          END IF

        END DO

        Index(I)=Indx

      GOTO 10

      END

C ******************************************************************** HeapSortR
```

```
      SUBROUTINE HeapSortR(N,Matrix)

C This subroutine will use the Heap Sort technique to sort Matrix in ascending
C numerical order. This subroutine is taken from Numerical Recipes: The
C Art of Scientific Computing by Press, Flannery, Teukolsky, and Vetterling,
C p. 231.

      INTEGER N,L,I,J,IL,IR
      REAL Matrix(*),RRA

      IF (N.EQ.1) RETURN

      L=N/2+1
      IR=N

C The index L will be decremented from its initial value down to 1 during
C the "hiring" (heap creation) phase. Once it reaches 1, the index IR will
C be decremented from its initial value down to 1 during the "retirement
C and prómotion" (heap selection) phase.

10    CONTINUE
        IF (L.GT.1) THEN

C This section of the IF-THEN-ELSE block is executed if we are still
C in the hiring phase.

          L=L-1
          RRA=Matrix(L)
        ELSE

C This section of the IF-THEN-ELSE block is executed if we are in the retirment
C and promotion phase. First we clear a space at the top of the array,
C and retire the top of the heap into it. Then we decrease the size
C of the corporation.

          RRA=Matrix(IR)
          Matrix(IR)=Matrix(1)
          IR=IR-1

          IF (IR.EQ.1) THEN

C This IF-THEN block is executed if the last promotion has been made.

            Matrix(1)=RRA
            RETURN
          END IF

        END IF

C Whether we are in the hiring phase or promtion phase, we here set up to
C sift down element RRa to its proper level.

        I=L
        J=L+L

        DO WHILE(J.LE.IR)

          IF (J.LT.IR) THEN
```

```
            IF (Matrix(J).LT.Matrix(J+1)) J=J+1
          END IF

          IF (RRA.LT.Matrix(J)) THEN
            Matrix(I)=Matrix(J)
            I=J
            J=J+J
          ELSE
            J=IR+1
          END IF

        END DO

        Matrix(I)=RRA

      GOTO 10

      END

C ************************************************************************** IInp

      SUBROUTINE IInp(Prompt,Var)

C This subroutine prints the Prompt and obtains an Integer from the user

      COMMON /EDITSUBS/DataEntered

      CHARACTER*(*) Prompt
      INTEGER Var
      CHARACTER*80 Response
      LOGICAL DataEntered

      CALL ChInp(Prompt,Response)
      IF (Response.EQ.' ') THEN
        DataEntered=.FALSE.
      ELSE
        DataEntered=.TRUE.
      END IF
      CALL ConvertChToI(Response,Var)

      RETURN
      END

C ************************************************************************** IXChng

      SUBROUTINE IXCHNG(I1,I2)

      INTEGER I1,I2,I3

      I3=I1
      I1=I2
      I2=I3

      RETURN
      END

C ********************************************************************** IncrementI
```

```
      SUBROUTINE IncrementI(I,Ip1,Inc,Limit)

C This subroutine returns I+Inc in Ip1 (or I+Inc-Limit if I+Inc is over
C Limit)

      INTEGER I,Ip1,Inc,Limit

      Ip1=I+Inc
      IF (Ip1.GT.Limit) Ip1=Ip1-Limit
      IF (Ip1.LE.0) Ip1=Limit+Ip1

      RETURN
      END

C ******************************************************************** LowerCaseCh

      SUBROUTINE LowerCaseCh(String)

      CHARACTER*(*) String
      INTEGER StringLength,NoCharacters,Char,I

      NoCharacters=0
      StringLength=LEN(String)

      DO I=1,StringLength
        IF ((String(I:I).GE.' ').AND.(String(I:I).LE.'~')) THEN
          NoCharacters=NoCharacters+1
          IF ((String(I:I).GE.'A').AND.(String(I:I).LE.'Z')) THEN
              String(NoCharacters:NoCharacters)=CHAR(ICHAR(String(I:I))+32)
          ELSE
              String(NoCharacters:NoCharacters)=String(I:I)
          END IF
        END IF
      END DO

      DO I=StringLength,NoCharacters+1,-1
        String(I:I)=' '
      END DO

      RETURN
      END

C ******************************************************************** MatDPToR

      SUBROUTINE MatDPToR(DP,NoOfElems,R)

      REAL*8 DP(*)
      REAL R(*)
      INTEGER NoOfElems,I

      DO I=1,NoOfElems
        R(I)=DP(I)
      END DO

      RETURN
      END

C ******************************************************************** MatDateBend
```

```
      SUBROUTINE MatDateBend(DateCh,NoOfElems,DateN)

      CHARACTER*(*) DateCh(*)
      INTEGER NoOfElems,DateN(*),I

      DO I=1,NoOfElems
        CALL DateBend(DateCh(I),DateN(I))
      END DO

      RETURN
      END

C ******************************************************************* MatExtremaDP

      SUBROUTINE MatExtremaDP(Matrix,NoOfPts,Maxi,Mini)

C This subroutine locates the minimum and maximum elements in a REAL matrix.

      REAL*8 Matrix(*),Maxi,Mini
      INTEGER NoOfPTs,I

      IF (Maxi.EQ.Mini) THEN
        Maxi=Matrix(1)
        Mini=Matrix(1)
      END IF

      DO I=1,NoOfPts
        Maxi=MAX(Maxi,Matrix(I))
        Mini=MIN(Mini,Matrix(I))
      END DO

      RETURN
      END

C ******************************************************************* MatExtremaR

      SUBROUTINE MatExtremaR(Matrix,NoOfPts,MaxI,MinI)
      REAL Matrix(*),Maxi,Mini
      INTEGER NoOfPTs,I
      WRITE(*,*) 'Error: incorrect routine.  Use VecExtremaR'
      STOP
      END

C ******************************************************************** MatSumI

      SUBROUTINE MatSumI(Matrix,Start,End,Sum)

C This subroutine finds the sum of values in Matrix(*) from item Start to
C item End.  The sum is returned in variable Sum.

      INTEGER Matrix(*),Start,End,Sum,I

      Sum=0
      DO I=Start,End
        Sum=Sum+Matrix(I)
      END DO

      RETURN
```

```
      END

C ************************************************************************** Menu

      SUBROUTINE Menu(MenuTitle,MenuTxt,NTxt,MenuPrompt,MenuResps,NResps,
     & DefResp,UsrResp)

C This subroutine prints a menu title, menu text, and a menu prompt, and
C obtains the response from the user.  If the user mereley presses return,
C the DEFRESP is used.  If MENUTITLE is passed as blanks, only the prompt
C is used.  Special Response codes are:
C date - the computer checks the input for a valid date
C #    - the computer checks each character to verify it is a digit, comma
C        E, +, -, period, or space.

      INTEGER NTxt,NResps,I,J,Length

      CHARACTER*(*) MenuTxt(NTxt)
      CHARACTER*(*) MenuResps(NResps),DefResp,UsrResp
      CHARACTER*(*) MenuTitle,MenuPrompt

      LOGICAL Match,NumMatch,DateMatch

      IF (MenuTitle.NE.' ') THEN
        CALL StringLength(MenuTitle,Length)
        Length=MAX(1,Length)
        WRITE(*,*) MenuTitle(1:Length)

        DO I=1,NTxt
          CALL StringLength(MenuTxt(I),Length)
          Length=MAX(1,Length)
          WRITE(*,*) MenuTxt(I)(1:Length)
        END DO
        WRITE(*,*) ' '
      END IF

      Match=.FALSE.
      DO WHILE (Match.EQ..FALSE.)

        WRITE(*,*) 'Default Response:',DefResp
        CALL ChInp(MenuPrompt,UsrResp)
        CALL UpperCaseCh(UsrResp)
        IF (UsrResp.EQ.' ') THEN
          UsrResp=DefResp
          WRITE(*,*) UsrResp
        END IF

        DO I=1,NResps
          IF (MenuResps(I).EQ.'#') THEN
              NumMatch=.TRUE.
              DO J=1,LEN(UsrResp)
              IF (INDEX(' ,.+-1234567890E',UsrResp(J:J)).LE.0)
     & NumMatch=.FALSE.
            END DO
            Match=Match.OR.NumMatch
          ELSE IF (MenuResps(I).EQ.'date') THEN
            DateMatch=.TRUE.
              IF (UsrResp(3:3).NE.'/') DateMatch=.FALSE.
```

```
                IF (UsrResp(6:6).NE.'/') DateMatch=.FALSE.
              DO J=1,8
                  IF (INDEX(' /0123456789',UsrResp(J:J)).LE.0)
   & DateMatch=.FALSE.
                END DO
                Match=Match.OR.DateMatch
            ELSE
              CALL UpperCaseCh(MenuResps(I))
                CALL StringLength(MenuResps(I),Length)
              Length=MAX(1,Length)
              IF (UsrResp(1:Length).EQ.MenuResps(I)(1:Length)) Match=.TRUE.
            END IF
          END DO

          IF (Match.EQ..FALSE.) THEN
            WRITE(*,*) 'Warning: ''',UsrResp,''' is an invalid response.'
          END IF

        END DO

        RETURN
        END

C ********************************************************************** PowerRound

        REAL*8 FUNCTION PowerRound(Number,PowerOfTen)

        INTEGER PowerOfTen,NoOfDigits,Sign
        REAL*8 Number,Rounded,Divisor,Ten/10.0/
        CHARACTER*80 Test

        IF (Number.LT.0) THEN
          Sign=-1
          Number=-Number
        ELSE
          Sign=1
        END IF

        IF (Number.GT.0) THEN
          NoOfDigits=LOG(Number)/LOG(Ten)-PowerOfTen+1

          IF (NoOfDigits.LE.0) THEN
            Rounded=0.0
          ELSE
            Divisor=Ten**PowerOfTen
            WRITE(UNIT=Test(1:),FMT=10)
   & Number/Divisor
10          FORMAT(F<NoOfDigits+2>.0)
            CALL ConvertChToDP(Test,Rounded)
            Rounded=Rounded*Divisor
          END IF

          IF (Sign.LT.0) THEN
            Rounded=-Rounded
          END IF

        ELSE
          Rounded=0.0
```

```
      END IF

      PowerRound=Rounded
      RETURN
      END

C ********************************************************************** PRoundDP

      Subroutine PRoundDP(Number,PowerOfTen,Rounded)

      DOUBLE PRECISION Number,TenRaised,Ten/10.0/,Rounded,Intermediate
      INTEGER PowerOfTen
      INTEGER*4 BigInt

      TenRaised=(Ten)**(PowerOfTen)
      BigInt=Number/TenRaised
      Intermediate=BigInt

      Rounded=Intermediate*TenRaised

      RETURN
      END

C ********************************************************************** PauseCRT

      SUBROUTINE PauseCrt

      CHARACTER*1 DummyVar

      CALL ChInp('Press Return to Continue',DummyVar)

      RETURN
      END

C ********************************************************************** QPInp

      SUBROUTINE QPInp(Prompt,Var16)

C This subroutine prints the Prompt and obtains a quadruple precision Var
C from the user

      CHARACTER*(*) PROMPT
      DOUBLE PRECISION Var8
      REAL*16 Var16

      CALL DPInp(Prompt,Var8)
      Var16=Var8

      RETURN
      END

C ********************************************************************** RInp

      SUBROUTINE RInp(Prompt,Var)

      COMMON /EDITSUBS/DataEntered

C This subroutine prints the Prompt and obtains a real Var from the user
```

```
      CHARACTER*(*) PROMPT
      REAL Var
      CHARACTER*80 Response
      LOGICAL DataEntered

      CALL ChInp(Prompt,Response)
      IF (Response.EQ.' ') THEN
        DataEntered=.FALSE.
      ELSE
        DataEntered=.TRUE.
      END IF
      CALL ConvertChToR(Response,Var)

      RETURN
      END

C ********************************************************************** RXChng

      SUBROUTINE RXChng(R1,R2)

C This subroutine is called by Bub2Sort to exchange two real numbers

      REAL R1,R2,T

      T=R1
      R1=R2
      R2=T

      RETURN
      END

C ********************************************************************** ReOrderDP

      SUBROUTINE ReOrderDP(Matrix,NoElems,Pointer)

      PARAMETER MaxNoOfElems=2000
      REAL*8 Matrix(*),Temp(MaxNoOfElems)
      INTEGER NoElems,Pointer(*),I

      IF (NoElems.GT.MaxNoOfElems) THEN
        WRITE(*,*) 'Error: Number of elements in Matrix is larger '
        WRITE(*,*) '       than the maximum number of elements allowed.'
        WRITE(*,*) '            NoElems=',NoElems
        WRITE(*,*) '       MaxNoOfElems=',MaxNoOfElems
        STOP 'In Routine: ReOrderDP'
      END IF

      DO I=1,NoElems
        Temp(I)=Matrix(I)
      END DO

      DO I=1,NoElems
        Matrix(I)=Temp(Pointer(I))
      END DO

      RETURN
      END
```

```
C ******************************************************************** ReOrderR

      SUBROUTINE ReOrderR(Matrix,NoElems,Pointer)

      PARAMETER MaxNoOfElems=2000
      REAL Matrix(*),Temp(MaxNoOfElems)
      INTEGER NoElems,Pointer(*),I

      IF (NoElems.GT.MaxNoOfElems) THEN
        WRITE(*,*) 'Error: Number of elements in Matrix is larger '
        WRITE(*,*) '       than the maximum number of elements allowed.'
        WRITE(*,*) '            NoElems=',NoElems
        WRITE(*,*) '       MaxNoOfElems=',MaxNoOfElems
        STOP 'In Routine: ReOrderR'
      END IF

      DO I=1,NoElems
        Temp(I)=Matrix(I)
      END DO

      DO I=1,NoElems
        Matrix(I)=Temp(Pointer(I))
      END DO

      RETURN
      END

C ********************************************************** RemoveBlanksFromCh

      SUBROUTINE RemoveBlanksFromCh(String)

      CHARACTER*(*) String
      CHARACTER*1 Blank/' '/

      CALL RemoveCharFromCh(String,Blank)

      RETURN
      END

C ********************************************************** RemoveCharFromCh

      SUBROUTINE RemoveCharFromCh(String,Blank)

      CHARACTER*(*) String,Blank
      INTEGER Length,Char,I

      Length=LEN(String)
      Char=0
      DO I=1,Length
        IF (String(I:I).NE.Blank(1:1)) THEN
          Char=Char+1
          String(Char:Char)=String(I:I)
        END IF
      END DO
      DO I=Char+1,Length
        String(I:I)=' '
      END DO
```

```
      RETURN
      END

C ******************************************************************** RptString

      SUBROUTINE RptString(Repeat,NoOfTimes,String)

      CHARACTER*(*) Repeat,String
      INTEGER NoOfTimes,SLen,RLen,I

      SLen=0
      RLen=LEN(Repeat)

      DO I=1,NoOfTimes
        String(SLen+1:)=Repeat
        SLen=Slen+RLen
        IF (SLen.GE.LEN(String)) RETURN
      END DO

      RETURN
      END

C ******************************************************************** SearchCh

      SUBROUTINE SearchCh(Matrix,NElements,MatchElement,Location,Found,
     & NoLocs)

      CHARACTER*(*) Matrix(*),MatchElement
      INTEGER NElements,Location,NoLocs,I,MatchLength
      LOGICAL Found

      Location=0
      Found=.FALSE.
      NoLocs=0

      IF (NElements.GT.0) THEN
        MatchLength=MIN(LEN(Matrix(1)),LEN(MatchElement))
      END IF

      DO I=1,NElements
        IF (Matrix(I)(1:MatchLength).EQ.
     & MatchElement(1:MatchLength)) THEN
          Found=.TRUE.
          Location=I
          NoLocs=NoLocs+1
        END IF
      END DO

      RETURN
      END

C ******************************************************************** SearchChSub

      SUBROUTINE SearchChSub(Matrix,NElements,StartLoc,EndLoc,
     & MatchElement,Location,Found,NoLocs)

      CHARACTER*(*) Matrix(*),MatchElement
      INTEGER NElements,Location,NoLocs,I,StartLoc,EndLoc
```

```
      LOGICAL Found

      Location=0
      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I)(StartLoc:EndLoc).EQ.MatchElement) THEN
          Found=.TRUE.
          Location=I
          NoLocs=NoLocs+1
        END IF
      END DO

      RETURN
      END

C ********************************************************************** SearchDP

      SUBROUTINE SearchDP(Matrix,NElements,MatchElement,Location,Found,
     & NoLocs)

      INTEGER NElements,Location,NoLocs,I
      DOUBLE PRECISION Matrix(*),MatchElement
      LOGICAL Found

      Location=0
      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I).EQ.MatchElement) THEN
          Found=.TRUE.
          Location=I
          NoLocs=NoLocs+1
        END IF
      END DO

      RETURN
      END

C ********************************************************************** SearchI

      SUBROUTINE SearchI(Matrix,NElements,MatchElement,Location,Found,
     & NoLocs)

      INTEGER Matrix(*),MatchElement
      INTEGER NElements,Location,NoLocs,I
      LOGICAL Found

      Location=0
      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I).EQ.MatchElement) THEN
          Found=.TRUE.
          Location=I
```

```
          NoLocs=NoLocs+1
        END IF
      END DO

      RETURN
      END

C ********************************************************************** SearchL

      SUBROUTINE SearchL(Matrix,NElements,MatchElement,Location,Found,
     & NoLocs)

      LOGICAL Matrix(*),MatchElement
      INTEGER NElements,Location,NoLocs,I
      LOGICAL Found

      Location=0
      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I).EQ.MatchElement) THEN
          Found=.TRUE.
          Location=I
          NoLocs=NoLocs+1
        END IF
      END DO

      RETURN
      END

C ********************************************************************** SearchLEI

      SUBROUTINE SearchLEI(Matrix,NElements,MatchElement,Location)

      INTEGER Matrix(*),MatchElement,NElements,Location,I

      Location=0

      DO I=1,NElements
        IF (Matrix(I).LE.MatchElement) THEN
          Location=I
          RETURN
        END IF
      END DO

      RETURN
      END

C ********************************************************************** SearchR

      SUBROUTINE SearchR(Matrix,NElements,MatchElement,Location,Found,
     & NoLocs)

      REAL Matrix(*),MatchElement
      INTEGER NElements,Location,NoLocs,I
      LOGICAL Found
```

```
      Location=0
      Found=.FALSE.
      NoLocs=0

      DO I=1,NElements
        IF (Matrix(I).EQ.MatchElement) THEN
          Found=.TRUE.
          Location=I
          NoLocs=NoLocs+1
        END IF
      END DO

      RETURN
      END

C ********************************************************************** SetPlotter

      SUBROUTINE SetPlotter(TermType)

      CHARACTER*(*) TermType
      INTEGER IErr

      IF (TermType.EQ.'VT') THEN
        CALL dev('dec_vt240_general',IErr)
      ELSE IF (TermType.EQ.'SEL') THEN
        CALL dev('tektronix_4014_general',IErr)
      ELSE IF (TermType.EQ.'TEK') THEN
        CALL dev('tektronix_4014_general',IErr)
      ELSE IF (Termtype.EQ.'LAS') THEN
        CALL dev('postscript_all_general',IErr)
        CALL HWROT('AUTO')
      ELSE
        WRITE(*,*) 'Error: unknown terminal type.'
        WRITE(*,*) '       TermType=',TermType
        STOP 'In Routine: SetPlotter'
      END IF

      CALL Page(8.5,11.0)
      CALL Area2D(5.0,5.0)

      CALL HWSCAL('SCREEN')

      RETURN
      END

C ********************************************************************** SgnDP

      INTEGER FUNCTION SgnDP(Number)

      DOUBLE PRECISION Number

      IF (Number.LT.0) THEN
        SgnDP=-1
      ELSE IF (Number.GT.0) THEN
        SgnDP=1
      ELSE
        SgnDP=0
      END IF
```

```
      RETURN
      END

C ************************************************************************ SortCh

      SUBROUTINE SortCh(Matrix,NoElems,Pointer)

      INTEGER NoElems,Pointer(*)
      CHARACTER*(*) Matrix(*)

      CALL HeapSortIndexCh(NoElems,Matrix,Pointer)

      RETURN
      END

C ************************************************************************ SortDP

      SUBROUTINE SortDP(Matrix,NoElems,Pointer)

C This subroutine sorts the matrix to pointer.  Upon return from the
C subroutine the elements of pointer(1),pointer(2),... are the locations
C in Matrix of the first element, second element, ...

      INTEGER NoElems,Pointer(*),Top,Bottom,I
      DOUBLE PRECISION Matrix(*),Elem1,Elem2

      LOGICAL Exchanges

      DO I=1,NoElems
        Pointer(I)=I
      END DO

      Bottom=1
      Top=NoElems

      Exchanges=.TRUE.

      DO WHILE (Exchanges)

        Exchanges=.FALSE.
        DO I=Bottom,Top-1
          Elem1=Matrix(Pointer(I))
          Elem2=Matrix(Pointer(I+1))
          IF (Elem1.GT.Elem2) THEN
              CALL IXCHNG(Pointer(I),Pointer(I+1))
              Exchanges=.TRUE.
              Top=I
          END IF
        END DO

        IF (Exchanges) THEN
          Exchanges=.FALSE.
          DO I=Top,Bottom+1,-1
            Elem1=Matrix(Pointer(I-1))
            Elem2=Matrix(Pointer(I))
              IF (Elem1.GT.Elem2) THEN
                CALL IXCHNG(Pointer(I-1),Pointer(I))
                Exchanges=.TRUE.
```

```
            Bottom=I
          END IF
        END DO
      END IF

    END DO

    RETURN
    END

C ************************************************************************** SortI

    SUBROUTINE SortI(Matrix,NoElems,Pointer)

C This subroutine sorts the matrix to pointer.  Upon return from the
C subroutine the elements of pointer(1),pointer(2),... are the locations
C in Matrix of the first element, second element, ...

    INTEGER NoElems,Pointer(*),Top,Bottom,I
    INTEGER Matrix(*),Elem1,Elem2

    LOGICAL Exchanges

    DO I=1,NoElems
      Pointer(I)=I
    END DO

    Bottom=1
    Top=NoElems

    Exchanges=.TRUE.

    DO WHILE (Exchanges)

      Exchanges=.FALSE.
      DO I=Bottom,Top-1
        Elem1=Matrix(Pointer(I))
        Elem2=Matrix(Pointer(I+1))
        IF (Elem1.GT.Elem2) THEN
            CALL IXCHNG(Pointer(I),Pointer(I+1))
            Exchanges=.TRUE.
            Top=I
        END IF
      END DO
      IF (Exchanges) THEN
        Exchanges=.FALSE.
        DO I=Top,Bottom+1,-1
          Elem1=Matrix(Pointer(I-1))
          Elem2=Matrix(Pointer(I))
            IF (Matrix(Pointer(I-1)).GT.Matrix(Pointer(I))) THEN
              CALL IXCHNG(Pointer(I-1),Pointer(I))
              Exchanges=.TRUE.
              Bottom=I
            END IF
        END DO
      END IF

    END DO
```

```
      RETURN
      END

C ************************************************************************ SortR

      SUBROUTINE SortR(Matrix,NoElems,Pointer)

C This subroutine sorts the matrix to pointer.  Upon return from the
C subroutine the elements of pointer(1),pointer(2),... are the locations
C in Matrix of the first element, second element, ...

      INTEGER NoElems,Pointer(*)
      REAL Matrix(*)

      CALL HeapSortIndexR(NoElems,Matrix,Pointer)

      RETURN
      END

C ******************************************************************** StringLength

      SUBROUTINE StringLength(String,Length)

      CHARACTER*(*) String
      INTEGER Length

      Length=LEN(String)
      DO WHILE((String(Length:Length).EQ.' ').OR.
     & (String(Length:Length).EQ.CHAR(0)))
        Length=Length-1
        IF (Length.EQ.0) RETURN
      END DO

      RETURN
      END

C ************************************************************************ TrimCh

      SUBROUTINE TrimCh(String)

      CHARACTER*(*) String
      INTEGER Length,I

      Length=LEN(String)

      DO I=1,Length
        IF ((ICHAR(String(I:I)).LT.32)
     & .OR.(ICHAR(String(I:I)).EQ.255)) String(I:I)=' '
      END DO

      IF (String.EQ.' ') RETURN
      IF (Length.LT.2) RETURN

      DO WHILE (String(1:1).EQ.' ')
        String(1:)=String(2:)
      END DO

      RETURN
```

```
      END
C ******************************************************************** UpperCaseCh

      SUBROUTINE UpperCaseCh(String)

      CHARACTER*(*) String
      INTEGER StringLength,NoCharacters,Char,I

      NoCharacters=0
      StringLength=LEN(String)

      DO I=1,StringLength
        IF (ICHAR(String(I:I)).LT.32) String(I:I)=' '
        IF ((String(I:I).GE.' ').AND.(String(I:I).LE.'~')) THEN
          NoCharacters=NoCharacters+1
          IF ((String(I:I).GE.'a').AND.(String(I:I).LE.'z')) THEN
              String(NoCharacters:NoCharacters)=CHAR(ICHAR(String(I:I))-32)
          ELSE
              String(NoCharacters:NoCharacters)=String(I:I)
          END IF
        END IF
      END DO

      IF (NoCharacters+1.LE.StringLength) String(NoCharacters+1:
     & StringLength)=' '

      RETURN
      END
C ******************************************************************** VecExtremaDP

      SUBROUTINE VecExtremaDP(Matrix,NoOfPts,Maxi,Mini)

C This subroutine locates the minimum and maximum elements in a REAL array

      REAL*8 Matrix(*),Maxi,Mini
      INTEGER NoOfPTs,I

      Maxi=Matrix(1)
      Mini=Matrix(1)

      DO I=2,NoOfPts
        Maxi=MAX(Maxi,Matrix(I))
        Mini=MIN(Mini,Matrix(I))
      END DO

      RETURN
      END
C ******************************************************************** VecExtremaR

      SUBROUTINE VecExtremaR(Matrix,NoOfPts,Maxi,Mini)

C This subroutine locates the minimum and maximum elements in a REAL array

      REAL Matrix(*),Maxi,Mini
      INTEGER NoOfPTs,I
```

```
      Maxi=Matrix(1)
      Mini=Matrix(1)

      DO I=2,NoOfPts
        Maxi=MAX(Maxi,Matrix(I))
        Mini=MIN(Mini,Matrix(I))
      END DO

      RETURN
      END

C ******************************************************************** WriteCenter

      SUBROUTINE WriteCenter(Lun,Width,Line)

      PARAMETER MaxWidth=108

      INTEGER Lun,Width,TrueWidth
      CHARACTER*(*) Line
      CHARACTER*(MaxWidth) In,Out

      IF (Width.GT.MaxWidth) THEN
        TrueWidth=MaxWidth
      ELSE
        TrueWidth=Width
      END IF

      In=Line
      CALL Center(In,TrueWidth,Out)

      WRITE(Lun,10) Out
10    FORMAT(X,A<TrueWidth>)

      RETURN
      END

C ************************************************************************** YorN

      SUBROUTINE Yorn(Prompt$,Yes)

      LOGICAL Yes,No
      CHARACTER Prompt$*(*),Response*1

C This subprogram obtains a Y or N response from the user.

C Prompt$ is the prompt to be used.
C Yes is the logical variable

      Yes=.FALSE.
      No=.FALSE.
      DO WHILE(.NOT.(Yes.OR.No))

        WRITE (*,*) Prompt$
        CALL ChInp('(Y or N)?',Response)

        Yes=(Response.EQ.'Y').OR.(Response.EQ.'y')
        No=(Response.EQ.'N').OR.(Response.EQ.'n')
```

```
      IF (.NOT.(Yes.OR.No)) THEN

        WRITE(*,100) Response
100     FORMAT (' ','Error: response ',A,' is not Y or N.')

      END IF

    END DO

    RETURN
    END

c *
```

```
C ********************************************************************** CalcCF

      SUBROUTINE CalcCF(XI,C,N)

C This subroutine is from Conte and DeBoor, p.287

C Variables-----------------------------------------------------------------

C Input Variables

C XI(1),...,XI(N+1) strictly increasing sequence of breakpoints.
C C(1,I), C(2,I), VAlue and first derivative at XI(I), I=1,...,N+1
C       of the piecewise function.

C Output Variables

C C(1,I), C(2,I), C(3,I), C(4,I) Polynomial coefficients of the function on the
C       interval (XI(I),XI(I+1), I=1,...N.

      INTEGER N,I
      REAL*8 C(4,N+1),XI(N+1),dX,DivDF1,DivDF3

      DO I=1,N
         dX=XI(I+1)-XI(I)
         DivDF1=(C(1,I+1)-C(1,I))/dX
         DivDF3=C(2,I)+C(2,I+1)-2.0*DivDF1
         C(3,I)=(DivDF1-C(2,I)-DivDF3)/dX
         C(4,I)=DivDF3/(dX*dX)
      END DO

      RETURN
      END


C ************************************************************** Compute1DPhis

      SUBROUTINE Compute1DPhis(BasisDeg,Zeta,Phi,PhiZeta)

C This subroutine computes the values of the local 1D weighting functions, and
C the values of their slopes.

C Input Variables

C Zeta        The value of the local variable at which to evaluate the
C             functions. (Range: 0<Zeta<1)
C BasisDeg    degree of basis functions, 3 are hermite cubics

C Output Variables

C Phi()       Phi(1) is the first local weighting function evaluated at
C             zeta. Phi(2) is the second.
C PhiZeta()   These are the derivates (with respect to zeta) evaluated at
C             zeta.

      DOUBLE PRECISION Zeta
      REAL*8 Phi(*),PhiZeta(*)
      INTEGER BasisDeg
```

```
      IF (BasisDeg.EQ.1) THEN

        Phi(1)=1.0D0-Zeta
        Phi(2)=Zeta

        PhiZeta(1)=-1.0D0
        PhiZeta(2)=1.0D0

      ELSE IF (BasisDeg.EQ.2) THEN

        Phi(1)=1.0D0-3.0D0*Zeta+2.0D0*Zeta**2
c 0<z<1          -1<x<1  x=2z-1;z=(x+1)/2
c (2z-1)(z-1)    (1/2)(x)(x-1)
        Phi(2)=4.0D0*Zeta-4.0D0*Zeta**2
c 4(z)(1-z)      (1-x)(1+x)
        Phi(3)=Zeta*(2.0D0*Zeta-1.0D0)
c 2z^2 - z
c z (2z-1)       (1/2)(x)(x+1)

        PhiZeta(1)=-3.0D0+4.0D0*Zeta
        PhiZeta(2)=4.0D0-8.0D0*Zeta
        PhiZeta(3)=-1.0D0+4.0D0*Zeta

      ELSE IF (BasisDeg.EQ.3) THEN

        Phi(1)=1.0-3.0*Zeta**2+2.0*Zeta**3
        Phi(2)=Zeta-2.0*Zeta**2+Zeta**3
        Phi(3)=3.0*Zeta**2-2.0*Zeta**3
        Phi(4)=-Zeta**2+Zeta**3

        PhiZeta(1)=-6.0*Zeta+6.0*Zeta**2
        PhiZeta(2)=1.0-4.0*Zeta+3.0*Zeta**2
        PhiZeta(3)=6.0*Zeta*(1.0-Zeta)
        PhiZeta(4)=-2.0*Zeta+3.0*Zeta**2

      END IF

      RETURN
      END

C ********************************************************** ComputeIntersection

      SUBROUTINE ComputeIntersection(Xa1,Ya1,Xa2,Ya2,Xb1,Yb1,Xb2,Yb2,
     & XInt,YInt,Error)

C     =0 => line segments intersect
C Error=1 => lines intersect, but segments do not.
C     =2 => lines are parallel
C Xa1   X or Y Coordinate, segment a or b, 1st or 2nd point.
C Yb2

      INTEGER Error
      LOGICAL Intersect
      REAL*8 Xa1,Ya1,Xa2,Ya2,Xb1,Yb1,Xb2,Yb2,Ta,Tb
      REAL*8 Denom,XInt,YInt,Temp,PowerRound
      EXTERNAL PowerRound

      Error=2
```

```
      Denom=(Xa1-Xa2)*(Yb2-Yb1)-(Ya1-Ya2)*(Xb2-Xb1)
      Intersect=(Denom.NE.0.0D0)

      IF (Intersect) THEN

C The lines intersect.  Compute the intersection point in parametric form
C (Y,X)=(Y(t),X(t)) for both segment a and b

        Ta = ((Yb2-Yb1)*(Xa1-Xb1)+(xb1-xb2)*(ya1-yb1))/Denom
        Tb = ((ya2-ya1)*(Xa1-Xb1)+(xa1-Xa2)*(Ya1-Yb1))/Denom

        IF ((0.LE.Ta).AND.(Ta.LE.1).AND.(0.LE.Tb).AND.(Tb.LE.1)) THEN
          Error=0
        ELSE
          Error=1
        END IF

C Compute the location of the intersection.

        XInt=Ta*(Xa2-Xa1)+Xa1
        YInt=Ta*(Ya2-Ya1)+Ya1

C Round to the nearest -16 place

        XInt=PowerRound(XInt,-16)
        YInt=PowerRound(YInt,-16)

      END IF

      RETURN
      END

C ************************************************************************ EElliptic

      REAL*8 FUNCTION EElliptic(M)

C Taken from Numerical Recipes, p. 187
C This function returns the complete elliptic integral with
C QQC=kc

      REAL*8 M

      REAL*8 QQc,Qc,A,B,P,F,E,Em,G,Q

C CA   Square-root of desired accuracy.

      PARAMETER CA=0.00000005
      PARAMETER PIO2=1.5707963267948966192313 22

      IF (ABS(M).GE.1) STOP 'Error in Routine EElliptic: M>=1.'
      Qc=SQRT(1.0-ABS(M))

      A=1.0
      B=1-M
      P=1.0
      E=Qc
      Em=1.0
```

```
1       F=A
        A=A+B/P
        G=E/P
        B=2*(B+F*G)
        P=G+P
        G=EM
        EM=QC+EM
        IF (ABS(G-QC).GT.G*CA) THEN
          QC=2*SQRT(E)
          E=QC*EM
          GOTO 1
        END IF

        EElliptic=PIO2*(B+A*EM)/(EM*(EM+P))
        RETURN
        END

C ************************************************************************ EElP

        REAL*8 FUNCTION EElP(M)

C Polynomial Approximation for Complete Elliptic Integral E, from Abramowitz and
C Stegun, p. 587

        REAL*8 M,M1,First,Second
        REAL*8 A(4)/0.44325141463,0.06260601220,0.04757383546,
     & 0.01736506451/
        REAL*8 B(4)/0.24998368310,0.09200180037,0.04069697526,
     & 0.00526449639/
        INTEGER I

        IF (M.GE.1.0) STOP 'In Routine: EElP; Error, M>=1'
        M1=1.0-M

        First=1.0
        Second=0.0
        DO I=1,4
          First=First+A(I)*(M1**I)
          Second=Second+B(I)*(M1**I)
        END DO

        EelP=First-Second*LOG(M1)

        RETURN
        END

C ************************************************************************ ErfCDP

        REAL*8 FUNCTION ErfCDP(X)

C This function returns the complementary errof function of x.  See
C numerical recipes p. 164

        REAL*8 X,GammaQDP,GammaPDP
        EXTERNAL GammaQDP,GammaPDP

        IF (X.LT.0.0) THEN
          ErfCDP=1.0D0+GammaPDP(0.5,X**2)
```

```
      ELSE
        ErfCDP=GammaQDP(0.5,X**2)
      END IF

      RETURN
      END

C ************************************************************************ ErfDP

      REAL*8 FUNCTION ErfDP(x)

C This function from NUMERICAL RECIPES, p. 164

      REAL*8 X,GammaPDP
      EXTERNAL GammaPDP

      IF (X.LT.0) THEN
        ErfDP=-GammaPDP(0.5,x**2)
      ELSE
        ErfDP=GammaPDP(0.5,x**2)
      END IF

      RETURN
      END

C ************************************************************************ Factor

      SUBROUTINE Factor(W,N,D,IPivot,IFlag,MaxDim)

C This subroutine, taken from Conte and DeBoor p165, computes the LU
C factorization of the Matrix W.  The program follows Algorithm 4.2 in
C Conte and DeBoor using scaled partial pivoting.

C Input Variables

C W(N,N)        Contains the matrix to be factored.
C N             The order of the Matrix
C D(N)          Work Vector, holds row sizes.

C Output Variables

C W(N,N)        The LU Factorization of P*W where W is the
C               original input matrix, and P is a permutation matrix
C               specified by IPivot.
C IPivot(N)     The Row IPivot(K) was used to eliminate X(K), K=1,...,N
C IFlag         =1 if an even number of interchanges was carried out,
C               =-1 if an odd number of interchanges was carried out,
C               =0 if the upper trangular factor has one or more zero
C                 diagonal entries.
C               Thus, the determinant is Iflag*W(1,N)

      INTEGER IFlag,IPivot(*),I,IStar,J,K,MaxDim,N
      REAL*8 D(*),W(MaxDim,MaxDim),Awikod,ColMax,Ratio,RowMax,Temp

      IFlag=1

C Intialize IPivot,D
```

```
      DO I=1,N
        IPivot(I)=I

        RowMax=0.0
        DO J=1,N
          RowMax=MAX(RowMax,ABS(W(I,J)))
        END DO

        IF (RowMax.EQ.0.0) THEN
          IFlag=0
          RowMax=1.0
        END IF

        D(I)=RowMax
      END DO

      IF (N.LE.1) RETURN

C Compute Factorization

      DO K=1,N-1

C Determine Pivot Row, IStar

        ColMax=ABS(W(K,K))/D(K)
        IStar=K
        DO I=K+1,N
          Awikod=ABS(W(I,K))/D(I)
          IF (Awikod.GT.ColMax) THEN
            ColMax=Awikod
            IStar=I
          END IF
        END DO

        IF (ColMax.EQ.0.0) THEN
          IFlag=0
        ELSE
          IF (IStar.GT.K) THEN

C Make K the Pivot Row by Interchanging it with the chosen row IStar.

            IFlag=-IFlag

            I=IPivot(IStar)
            IPivot(IStar)=IPivot(K)
            IPivot(K)=I

            Temp=D(IStar)
            D(IStar)=D(K)
            D(K)=Temp

            DO J=1,N
              Temp=W(IStar,J)
              W(IStar,J)=W(K,J)
              W(K,J)=Temp
            END DO

          END IF
```

```
C Eliminate X(K) from Rows K+1,...,N.

            DO I=K+1,N
              W(I,K)=W(I,K)/W(K,K)
              Ratio=W(I,K)
              DO J=K+1,N
                W(I,J)=W(I,J)-Ratio*W(K,J)
              END DO
            END DO
          END IF
        END DO

        IF (W(N,N).EQ.0.0) IFlag=0

        RETURN
        END

C ********************************************************************** GammaPdp

        REAL*8 FUNCTION GammaPdp(A,X)

C This function returns the incomplete gamma function P(a,x).  See Numerical
C Recipes p. 160.

        REAL*8 A,X,GamSer,Gln,GammCF

        IF ((X.LT.0).OR.(A.LE.0)) THEN
          WRITE(*,*) 'Error: X<0 or A<=0'
          WRITE(*,*) '       X=',X,'     A=',A
          STOP 'In Function: GammaPdp'
        END IF

        IF (X.LT.A+1) THEN

C Use the series representation

          CALL GSerDP(GamSer,A,X,Gln)
          GammaPdp=GamSer

        ELSE

C Use the continued fraction representation and takes its complement

          CALL GCFDP(GammCF,A,X,Gln)
          GammaPdp=1.0-GammCF

        END IF

        RETURN
        END

C ********************************************************************** GammaQDP

        REAL*8 FUNCTION GammaQDP(A,X)

C This function returns the incomplete gamma function Q(a,x)=1-P(a,x)  See
C Numerical Recipes p.162
```

```
      REAL*8 A,X,GamSer,Gln,GammCF

      IF ((X.LT.0).OR.(A.LE.0)) THEN
        WRITE(*,*) 'Error: improper A or X values.  Proper Ranges are:'
        WRITE(*,*) '       0<=X, and 0<A'
        WRITE(*,*) '       X=',X
        WRITE(*,*) '       A=',A
        STOP 'In Function: GammaQDP'
      END IF

      IF (X.LT.A+1.) THEN
        CALL GSer(GamSer,A,X,GLN)
        GammaQdp=1.0D0-GamSer
      ELSE
        CALL GCF(GammCF,A,X,Gln)
        GammaQDP=GammCF
      END IF

      RETURN
      END

C ********************************************************************** GammLNDP

      REAL*8 FUNCTION GammLnDP(XX)

C Returns the value LOG(GAMMA(XX)) for XX>0.  Full accuracy is obtained
C for XX>1.  For 0<XX<1, the reflection formula (6.1.4) can be used first.
C See Numerical Recipes p.157

      REAL*8 Cof(6)/76.18009173D0,-86.50532033D0,24.01409822D0,
     & -1.231739516D0,0.120858003D-2,-0.536382D-5/,
     & Stp/2.50662827465D0/,Half/0.5D0/,One/1.0D0/,FPF/5.5D0/,
     & X,Tmp,Ser,XX
      INTEGER J

      X=XX-One
      Tmp=X+Fpf
      Tmp=(X+Half)*LOG(Tmp)-Tmp
      Ser=One

      DO J=1,6
        X=X+One
        Ser=Ser+Cof(J)/X
      END DO

      GammLnDP=Tmp+LOG(Stp*Ser)

      RETURN
      END

C ********************************************************************** GCFDP

      SUBROUTINE GCFDP(GammCF,A,X,GLN)

C Returns the incomplete gamma function Q(a,x) evaluated by its continued
C fraction representation as GammCF.  Also returns LOG Gamma(A) as GLN.
C See Numerical Recipes p. 162
```

```
      REAL*8 GammCF,A,X,Gln,G,GOld,A0,A1,B0,B1,Fac
      REAL*8 An,Ana,Anf,GammLnDP
      INTEGER N
      EXTERNAL GammLnDP

      PARAMETER ITMax=500
      PARAMETER Eps=3.E-7

C Variables

C GOld The previous value of G tested against for convergence.

      Gln=GammLnDP(A)

      GOld=0.

      A0=1.
      A1=X

      B0=0.
      B1=1.

      Fac=1.

      DO N=1,ITMax
        An=FLOAT(N)
        ANA=AN-A

C One step of the recurrence (5.2.5)

        A0=(A1+A0*ANA)*Fac
        B0=(B1+B0*ANA)*Fac

C The next step of the recurrence (5.2.5)

        Anf=AN*Fac
        A1=X*A0+Anf*A1
        B1=X*B0+Anf*B1

C Shall we renormalize?

        IF (A1.NE.0.0) THEN
          Fac=1./A1
          G=B1*Fac
          IF (ABS((G-GOld)/G).LT.Eps) THEN
            GammCF=EXP(-X+A*LOG(X)-Gln)*G
            RETURN
          END IF
          GOld=G
        END IF
      END DO

      WRITE(*,*) 'Error: A too large or ITMax too small'
      WRITE(*,*) '          A=',A
      WRITE(*,*) '          X=',X
      WRITE(*,*) '      ITMax=',ITMAX
      STOP 'In Routine: GCFDP'
```

```
      END
C ********************************************************************** GSerDP
      SUBROUTINE GSerDP(GamSer,A,X,Gln)

C Returns the incomplete Gamma Function P(a,x) evaluated by its series
C representation as GamSer.  Also returns LOG Gamma(a) as Gln.

      INTEGER N
      REAL*8 GamSer,A,X,Gln,Ap,Sum,Del,GammLnDP
      EXTERNAL GammLnDP

      PARAMETER ITMax=100
      PARAMETER Eps=3.0E-7

      Gln=GammLnDP(A)

      IF (X.LT.0) STOP 'In Subroutine: GSerDP'

      IF (X.EQ.0.0) THEN
        GamSer=0.0
        RETURN
      END IF

      AP=A
      Sum=1.0/A
      Del=Sum

      DO N=1,ITMax
        AP=AP+1.
        Del=Del*X/AP
        Sum=Sum+Del
        IF (Abs(Del).LT.ABS(Sum)*Eps) THEN
          GamSer=Sum*EXP(-X+A*LOG(X)-GLN)
          RETURN
        END IF
      END DO

      STOP 'In Routine: GSerDP, A too large or ITMax too small'
      END
C ********************************************************************** GauLeg
      SUBROUTINE GauLeg(Np,Points,Weights)

C This subroutine returns the Gauss-Legendre integration points and weights for
C a Np-point gaussian quadrature integration over the interval 0<x<1.
C The value of the integral is approximated by the
C     sum(F(Points(i))*Weights(i),i=1,Np)

C (Modified from Press, Flannery, Teuukolsky, and Vetterling,
C NUMERICAL RECIPES: The Art of Scientific Computing, 1986, Cambridge
C University Press, p. 125-126)

      REAL*8 Points(*),Weights(*),Z,PP,Z1,Z2,P1,P2,P3
      INTEGER Np,I,J
```

```
      PARAMETER Eps=3.D-14

C Loop over the desired roots

      DO I=1,(Np+1)/2
        Z=COS(3.141592654D0*(I-0.25D0)/(Np+0.5D0))

C Starting with the above approximation to the Ith root, we enter the main
C loop of refinement by Newton's method.

1       CONTINUE

          P1=1.D0
          P2=0.D0

          DO J=1,Np
            P3=P2
            P2=P1
            P1=((2.D0*J-1.D0)*Z*P2-(J-1.D0)*P3)/J
          END DO

C P1 is now the desired Legendre polynomial.  We next compute PP, its
C derivative, by a standard relation involving also P2, the polynomial of
C one lower order.

          PP=Np*(Z*P1-P2)/(Z*Z-1.D0)
          Z1=Z
          Z=Z1-P1/PP
        IF (ABS(Z-Z1).GT.Eps) GOTO 1

        Points(Np+1-I)=Z
        Weights(Np+1-I)=2.D0/((1.D0-Z*Z)*PP*PP)

      END DO

      DO I=1,NP/2
        Points(I)=-Points(Np+1-I)
        Weights(I)=Weights(Np+1-I)
      END DO

C The following DO-LOOP convert the gauss points and weights from an interval
C of -1 to 1 to an interval from 0.0 to 1.0

      DO I=1,NP
        Points(I)=(Points(I)+1.0D0)/2.0D0
        Weights(I)=Weights(I)/2.0D0
      END DO

      RETURN
      END

C ************************************************************************ GauLog

      SUBROUTINE GauLog(Np,Points,Weights)

      INTEGER Np
      REAL*8 Points(*),Weights(*)
```

```
C This subroutine returns the gauss points and weights for integrating
C Ln (1/x) * f(x) from 0 to 1. The integral is the sum of
C Weights(i)*f(Points(i)) for i=1 to Np. The the values for points and
C weights come form GAUSSIAN QUADRATURE FORMULAS, A.H. Stroud and Don Secrest
C Prentice-Hall, Inc., Englewood Cliffs, N.J. C 1966

      IF (Np.EQ.2) THEN

        Points(1)=0.11200880616697618295
        Points(2)=0.60227690811873810275

        Weights(1)=0.71853931903038444066
        Weights(2)=0.28146068096961555933

      ELSE IF (Np.EQ.3) THEN

        Points(1)=0.06389079308732540499 6
        Points(2)=0.36899706371561876554
        Points(3)=0.76688030393894145542

        Weights(1)=0.51340455223236332512
        Weights(2)=0.39198004120148755480
        Weights(3)=0.09461540656614912006 4

      ELSE IF (Np.EQ.4) THEN

        Points(1)=0.04144848019938322080 3
        Points(2)=0.24527491432060225193
        Points(3)=0.55616545356027583718
        Points(4)=0.84898239453298517464

        Weights(1)=0.38346406814513512485
        Weights(2)=0.38687531777476262733
        Weights(3)=0.19043512695014241536
        Weights(4)=0.03922548712995983245 2

      ELSE IF (Np.EQ.5) THEN

        Points(1)=0.02913447215197205330 3
        Points(2)=0.17397721332089762870
        Points(3)=0.41170252028490204317
        Points(4)=0.67731417458282038070
        Points(5)=0.89477136103100828363

        Weights(1)=0.29789347178289445727
        Weights(2)=0.34977622651322418037
        Weights(3)=0.23448829004405241888
        Weights(4)=0.09893045951663314697 6
        Weights(5)=0.01891155214319579648 9

      ELSE IF (Np.EQ.6) THEN

        Points(1)=0.02163400584411694899 5
        Points(2)=0.12958339115495079613
        Points(3)=0.31402044991476550879
        Points(4)=0.53865721735180214454
        Points(5)=0.75691533737740285216
        Points(6)=0.92266885137212023733
```

```
      Weights(1)=0.23876366257854756972
      Weights(2)=0.30828657327394679296
      Weights(3)=0.24531742656321038598
      Weights(4)=0.14200875656647668542
      Weights(5)=0.05545462232488629001 5
      Weights(6)=0.01016895869293227588 6

   ELSE IF (Np.EQ.7) THEN

      Points(1)=0.01671935540825851594 1
      Points(2)=0.10018567791567512158
      Points(3)=0.24629424620793059904
      Points(4)=0.43346349325703310583
      Points(5)=0.63235098804776608846
      Points(6)=0.81111862674010557652
      Points(7)=0.9408481667433477217 6

      Weights(1)=0.19616938942524820752
      Weights(2)=0.27030264424727298214
      Weights(3)=0.23968187300769094830
      Weights(4)=0.16577577481043290656
      Weights(5)=0.08894322713765796443 5
      Weights(6)=0.03319430435657106702 5
      Weights(7)=0.00593278701512592399 91

   ELSE IF (Np.EQ.8) THEN

      Points(1)=0.01332024415089246501 2
      Points(2)=0.07975042901389493840 9
      Points(3)=0.19797102932618805379
      Points(4)=0.35415399435190941967
      Points(5)=0.52945857523491727770
      Points(6)=0.70181452993909996383
      Points(7)=0.84937932044110667604
      Points(8)=0.95332645005635978876

      Weights(1)=0.16441660472800288683
      Weights(2)=0.23752561002330602050
      Weights(3)=0.22684198443191912636
      Weights(4)=0.17575407900607024498
      Weights(5)=0.11292403024675905185
      Weights(6)=0.05785221071778207239 8
      Weights(7)=0.02097907374213297804 3
      Weights(8)=0.00368640710402761901 33

   ELSE IF (Np.EQ.9) THEN

      Points(1)=0.01086933608417547711 3
      Points(2)=0.06498366633800793940 6
      Points(3)=0.16222939802388293870
      Points(4)=0.29374990397167465807
      Points(5)=0.44663188190546803720
      Points(6)=0.60548166277612862084
      Points(7)=0.75411013715716356665
      Points(8)=0.87726582883583825317
      Points(9)=0.96225055941028184144

      Weights(1)=0.14006843874813473426
```

```
      Weights(2)=0.20977220520103044750
      Weights(3)=0.21142714989660272853
      Weights(4)=0.17715623393907998954
      Weights(5)=0.12779922803329549588
      Weights(6)=0.07847890261156217546
      Weights(7)=0.03902250498539909684 7
      Weights(8)=0.01386729554959302329 0
      Weights(9)=0.00240804103639231157 29

   ELSE IF (Np.EQ.10) THEN

      Points(1)=0.0090426309621996506369
      Points(2)=0.05397126622250062950 4
      Points(3)=0.13531182463925077487
      Points(4)=0.24705241628715982422
      Points(5)=0.38021253960933233397
      Points(6)=0.52379231797184320116
      Points(7)=0.66577520551642459722
      Points(8)=0.79419041601196621735
      Points(9)=0.89816109121900353816
      Points(10)=0.96884798871863353939

      Weights(1)=0.12095513195457051498
      Weights(2)=0.18636354256407187032
      Weights(3)=0.19566087327775998271
      Weights(4)=0.17357714218290692084
      Weights(5)=0.13569567299548420166
      Weights(6)=0.09364675853811052598 7
      Weights(7)=0.05578772735141587407 5
      Weights(8)=0.02715981089923333114 5
      Weights(9)=0.00951518260284851499 92
      Weights(10)=0.00163815763359826325 48

   ELSE IF (Np.EQ.11) THEN

      Points(1)=0.0076439411746377066292
      Points(2)=0.04554182825657891854 8
      Points(3)=0.11452229745512458369
      Points(4)=0.21037858122703353088
      Points(5)=0.32669555322169284797
      Points(6)=0.45545324692881343831
      Points(7)=0.58764835635908440789
      Points(8)=0.71396385001256144053
      Points(9)=0.82545321780181180417
      Points(10)=0.91419392161254313794
      Points(11)=0.97386025627558615232

      Weights(1)=0.10565225609910049130
      Weights(2)=0.16657168060062904863
      Weights(3)=0.18056321828775372476
      Weights(4)=0.16727873677378417932
      Weights(5)=0.13869705740163122050
      Weights(6)=0.10383343336504406027
      Weights(7)=0.06953669788873523234 6
      Weights(8)=0.04054160080359632956 8
      Weights(9)=0.01943540247621817278 0
      Weights(10)=0.00673742934245006270 21
      Weights(11)=0.00115248696105747778 32
```

```
ELSE IF (Np.EQ.12) THEN

  Points(1)=0.0065487222790800587892
  Points(2)=0.03894680956044995916 1
  Points(3)=0.09815026310600662886 2
  Points(4)=0.18113858159063157735
  Points(5)=0.28322006766737255470
  Points(6)=0.39843443516343664370
  Points(7)=0.51995262679235266272
  Points(8)=0.64051091671610645430
  Points(9)=0.75286501205183057837
  Points(10)=0.85024002416230220067
  Points(11)=0.92674968322391410104
  Points(12)=0.97775612968999747917

  Weights(1)=0.09319269144393132449 1
  Weights(2)=0.14975182757632236417
  Weights(3)=0.16655745436459300532
  Weights(4)=0.15963355943698765116
  Weights(5)=0.13842483186483562106
  Weights(6)=0.11001657063572116233
  Weights(7)=0.07996182177082897026 4
  Weights(8)=0.05240695482464177065 0
  Weights(9)=0.03007108887376118712 3
  Weights(10)=0.01424924558799827910 7
  Weights(11)=0.00489992458232176093 90
  Weights(12)=0.00083402903805690336469

ELSE IF (Np.EQ.13) THEN

  Points(1)=0.0056747662562426690299
  Points(2)=0.03369010879903253674 8
  Points(3)=0.08503675447417502808 9
  Points(4)=0.15749755947788902873
  Points(5)=0.24756957887684314613
  Points(6)=0.35074431236085520042
  Points(7)=0.46177374676161024622
  Points(8)=0.57495946652556132074
  Points(9)=0.68445988035043004253
  Points(10)=0.78460256881034708051
  Points(11)=0.87018642840788838882
  Points(12)=0.93675782930675139337
  Points(13)=0.98084345181159094850

  Weights(1)=0.08290049679327578781 8
  Weights(2)=0.13536867316574450040
  Weights(3)=0.15377328439229220085
  Weights(4)=0.15145815850998819062
  Weights(5)=0.13604033653728306071
  Weights(6)=0.11317682288163380335
  Weights(7)=0.08737443048004525823 8
  Weights(8)=0.06216023064180486951 7
  Weights(9)=0.04008772893416585189 8
  Weights(10)=0.02272384493997219532 8
  Weights(11)=0.01067123041296844408 7
  Weights(12)=0.00364649227597414007 97
  Weights(13)=0.00061827003485169707685
```

```
ELSE IF (Np.EQ.14) THEN

  Points(1)=0.004966003573868542243 9
  Points(2)=0.02943254011888517828 6
  Points(3)=0.07437629222453576261 0
  Points(4)=0.13813849198918628179
  Points(5)=0.21805564849895907807
  Points(6)=0.31066208391810198317
  Points(7)=0.41187247517775020720
  Points(8)=0.51717930739865432972
  Points(9)=0.62186485972851111970
  Points(10)=0.72122074520810885373
  Points(11)=0.81076598807158985635
  Points(12)=0.88645403803443465718
  Points(13)=0.94485913946181863892
  Points(14)=0.98333102648567848004

  Weights(1)=0.07429122506751041249 8
  Weights(2)=0.12298877246932291429
  Weights(3)=0.14219930656252335570
  Weights(4)=0.14322929764126422201
  Weights(5)=0.13234508377208520926
  Weights(6)=0.11413587573667647525
  Weights(7)=0.09228303807907361321 6
  Weights(8)=0.06975367329393756455 0
  Weights(9)=0.04883032360051356464 4
  Weights(10)=0.03110179606441614111 7
  Weights(11)=0.01746281195019609383 2
  Weights(12)=0.00814242342987593613 32
  Weights(13)=0.00276843641856393733 18
  Weights(14)=0.00046793591404056013532

ELSE IF (Np.EQ.15) THEN

  Points(1)=0.0043831101754754038348
  Points(2)=0.02593589810533061610 2
  Points(3)=0.06559609541231624525 4
  Points(4)=0.12210193407333160332
  Points(5)=0.19339526237400711598
  Points(6)=0.27677283870610202439
  Points(7)=0.36901512713974294381
  Points(8)=0.46652432896470658267
  Points(9)=0.56547347379181730642
  Points(10)=0.66196291901245642138
  Points(11)=0.75217888337878579878
  Points(12)=0.83254803386618958925
  Points(13)=0.89988205012089808432
  Points(14)=0.95150618874340990272
  Points(15)=0.98536446812213193892

  Weights(1)=0.06700997891649371360 3
  Weights(2)=0.11226415028670574182
  Weights(3)=0.13176017703967990373
  Weights(4)=0.13521764906193472513
  Weights(5)=0.12788179864568037040
  Weights(6)=0.11353290749021942129
  Weights(7)=0.09520523978435865851 1
  Weights(8)=0.07538931416739595433 9
```

```
      Weights(9)=0.05607842449265371799 2
      Weights(10)=0.03876829537501823111 0
      Weights(11)=0.02445148326875007596 0
      Weights(12)=0.01362463013823884690 5
      Weights(13)=0.00631644759859076119 98
      Weights(14)=0.00213888991594447134 78
      Weights(15)=0.00036061381833540664685

    ELSE IF (Np.EQ.16) THEN

      Points(1)=0.0038978344871159159240
      Points(2)=0.02302894561687323982 0
      Points(3)=0.05828039830624041234 8
      Points(4)=0.10867836509105403648
      Points(5)=0.17260945490984393776
      Points(6)=0.24793705447057849514
      Points(7)=0.33209454912991715598
      Points(8)=0.42218391058194860011
      Points(9)=0.51508247338146260347
      Points(10)=0.60755612044772872408
      Points(11)=0.69637565322821406115
      Points(12)=0.77843256587326540520
      Points(13)=0.85085026971539108323
      Points(14)=0.91108685722227190541
      Points(15)=0.95702557170354215759
      Points(16)=0.98704780024798447675

      Weights(1)=0.06079171004359123285 1
      Weights(2)=0.10291567751758214438
      Weights(3)=0.12235566204600919355
      Weights(4)=0.12756924693701598871
      Weights(5)=0.12301357460007091542
      Weights(6)=0.11184724485548572262
      Weights(7)=0.09659638515212434125 2
      Weights(8)=0.07935666435147313878 2
      Weights(9)=0.06185049458196520709 5
      Weights(10)=0.04543524650772666862 8
      Weights(11)=0.03109897475158180640 9
      Weights(12)=0.01945976592736084207 8
      Weights(13)=0.01077625496320552564 5
      Weights(14)=0.00497254289008764171 25
      Weights(15)=0.00167820111005119451 50
      Weights(16)=0.00028235376466843632177

    ELSE

      WRITE(*,*) 'Error: Np is too large or small!-',Np,'=Np'
      STOP 'In Routine: GauLog'

    END IF

    RETURN
    END

C ******************************************************************** GaussPtsWts

    SUBROUTINE GaussPtsWts(Np,Points,Weights)
```

```
C This subroutine returns the Gauss-Legendre integration points and weights for
C a Np-point gaussian quadrature integration over the intervall 0<x<1.
C The value of the integral is approximated by the
C     sum(F(Points(i))*Weights(i),i=1,Np)

C (Modified from Conte and de Boor, ELEMENTARY NUMERICAL ANALYSIS, 3d ed.
C McGraw Hill p.316)

C Input Variables

C Np    The number of points approximating the interval

C Output Variables

C Points(NP)    The points to use in approximating the integral
C Weights(NP)   The weights to use.

C Internal Variables

C I     Loop counter

      DOUBLE PRECISION Points(*),Weights(*)
      INTEGER Np,I

      IF (Np.EQ.1) THEN

        Points(1)=0.0
        Weights(1)=2.0

      ELSE IF (Np.EQ.2) THEN

        Points(2)=0.577350269189626
        Weights(2)=1.0

      ELSE IF (Np.EQ.3) THEN

        Points(2)=0.0
        Points(3)=0.774596669241483
        Weights(2)=0.8888888888888889
        Weights(3)=0.5555555555555556

      ELSE IF (Np.EQ.4) THEN

        Points(3)=0.339981043584856
        Points(4)=0.861136311594053
        Weights(3)=0.652145154862546
        Weights(4)=0.347854845137454

      ELSE IF (Np.EQ.5) THEN

        Points(3)=0.0
        Points(4)=0.538469310105683
        Points(5)=0.906179845938664
        Weights(3)=0.568888888888889
        Weights(4)=0.478628670499366
        Weights(5)=0.236926885056189

      ELSE IF (Np.EQ.6) THEN
```

```
          Points(4)=0.238619186083197
          Points(5)=0.661209386466265
          Points(6)=0.932469514203152
          Weights(4)=0.467913934572691
          Weights(5)=0.360761573048139
          Weights(6)=0.171324492379170

        ELSE IF (Np.GT.6) THEN

          WRITE(*,600) NP
600       FORMAT(X,'The number of points (',I3,') is greater than 6.',
     & /,X,'Please call this GaussPtsWts with Np <= 6.',
     & /,X,'The GauLeg routine will compute the gauss points ',
     & /,X,'for an Np')
          STOP 'In Routine GaussPtsWts'

        END IF

        DO I=1,NP/2
          Points(I)=-Points(Np+1-I)
          Weights(I)=Weights(Np+1-I)
        END DO

C The following DO-LOOP convert the gauss points and weights from an interval
C of -1 to 1 to an interval from 0.0 to 1.0

        DO I=1,NP
          Points(I)=(Points(I)+1.0D0)/2.0D0
          Weights(I)=Weights(I)/2.0D0
        END DO

        RETURN
        END

C ********************************************************************** GaussQuad

        SUBROUTINE GaussQuad(NoOfGauss,Points,Weights,Funk,Sum)

C This subroutine will perform the gaussian quadrature tsummation.
C (Note, can be used for integration by gaussian quadrature of
C f(x) or ln(1/x)f(X) by changing the weights passed to the subroutine

C Passed Variables

        INTEGER NoOfGauss
        REAL*8 Sum,Points(*),Weights(*)

C Passed Functions

        REAL*8 Funk
        EXTERNAL Funk

C Internal Variables

        INTEGER Gs

C Subroutine Code
```

```
      Sum=0
      DO Gs=1,NoOfGauss
        Sum=Sum+Funk(Points(Gs))*Weights(Gs)
      END DO

      RETURN
      END

C ********************************************************************** Interp1D

      SUBROUTINE Interp1D(X1,F1,X2,F2,Xd,Fd)

      REAL*8 X1,X2,Xd,F1,F2,Fd

      IF (X2.EQ.X1) THEN
        Fd=F1
      ELSE
        Fd=F1+(F2-F1)*(Xd-X1)/(X2-X1)
      END IF

      RETURN
      END

C ********************************************************************** Interp2D

      SUBROUTINE Interp2D(X1,Y1,X2,Y2,F11,F12,F21,F22,Xd,Yd,Fd)

      REAL*8 X1,Y1,X2,Y2,F11,F12,F21,F22,Xd,Yd,Fd
      REAL*8 FX1,FX2,FD1,F1Y,F2Y,FD2

      CALL Interp1D(X1,F11,X2,F21,XD,FX1)
      CALL Interp1D(X1,F12,X2,F22,Xd,FX2)
      CALL Interp1D(Y1,FX1,Y2,FX2,Yd,Fd1)

      CALL Interp1D(Y1,F11,Y2,F12,Yd,F1Y)
      CALL Interp1D(Y1,F21,Y2,F22,Yd,F2Y)
      CALL Interp1D(X1,F1Y,X2,F2Y,Xd,Fd2)

      FD=(Fd1+Fd2)/2.0

      RETURN
      END

C ********************************************************************** KElliptic

      REAL*8 FUNCTION KElliptic(M)

C Taken from Numerical Recipes, p. 187
C This function returns the complete elliptic integral with
C QQC=k

C CA   Square-root of desired accuracy.

      PARAMETER CA=0.00000005
      PARAMETER PIO2=1.57079632679489661923132 2

      REAL*8 M,PP,AA,BB
```

```
      REAL*8 Qc,A,B,P,E,EM,F,G

      IF (ABS(M).GE.1) THEN
        WRITE(*,*) 'Error: M>=1  ',M,'=M'
        STOP 'In Routine: KElliptic'
      END IF

      QC=SQRT(1.0-ABS(M))

      A=1
      B=1
      P=1
      E=QC
      EM=1.0

1     F=A
      A=A+B/P
      G=E/P
      B=2*(B+F*G)
      P=G+P
      G=EM
      EM=QC+EM
      IF (ABS(G-QC).GT.G*CA) THEN
        QC=2*SQRT(E)
        E=QC*EM
        GOTO 1
      END IF

      KElliptic=PIO2*(B+A*EM)/(EM*(EM+P))
      RETURN
      END

C ************************************************************************** KElP

      REAL*8 FUNCTION KElP(M)
      REAL*8 M

      REAL*8 First,Second,M1

      CALL KelPCoef(M,M1,First,Second)
      KelP=First-Second*LOG(M1)

      RETURN
      END

C ********************************************************************** KElPCoef

      SUBROUTINE KElPCoef(M,M1,First,Second)

C Polynomial Approximation to Complete Elliptic Integral K, taken from
C Abramowitz and Stegun, p 587

      REAL*8 M,M1,First,Second
      REAL*8 A(5)/1.38629436112,0.09666344259,0.03590092383,
     & 0.03742563713,0.01451196212/
      REAL*8 B(5)/0.5,0.12498593597,0.06880248576,
     & 0.03328355346,0.00441787012/
      INTEGER I
```

```
      IF (M.GE.1.0) THEN
        WRITE(*,*) 'Error: M>=1      M=',M
        M=M/0.0
        STOP 'In Routine: KEIP'
      END IF

      M1=1.0-M

      First=0.0
      Second=0.0
      DO I=1,5
        First=First+A(I)*(M1**(I-1))
        Second=Second+B(I)*(M1**(I-1))
      END DO

      RETURN
      END

C ********************************************************************** LUBksbDP
C ***********************************************************************LUBksb

      SUBROUTINE LUBksb(A,N,Np,Indx,B)

C This routine from Numerical Recipes, p. 37.

C Solves the set of N linear equations A.x=B. Here A is input, not
C as the matrix A but rather as its LU decomposition, determined by the
C routien LUDcmp. Indx is input as the permutation vector returned by
C LUDcmp. B is input as the right-hand side vectore B, and returns with
C the solution vecter X. A,N,Np and Indx are not modified by this routine
C and can be left in place for successive calls with different right-hand
C sides B. This routine takes into account the possiblity that B will begin
C with many zero elements, so it is efficient for use in matrix inversion.

      INTEGER N,Np,Indx(Np),II,I,LL,J
      REAL A(Np,Np),B(Np),Sum

C When II is set ot a positive value, it will become the index of the first
C nonvanishing element of B. We now do the forward substitution, equation
C 2.3.6. The only new wrinkle is to unscramble the permutation as we go.

      II=0

      DO I=1,N
        LL=Indx(I)
        Sum=B(ll)
        B(ll)=B(I)
        IF (ii.NE.0) THEN
          DO J=ii,i-1
            Sum=Sum-A(I,J)*B(J)
          END DO
        ELSE IF (Sum.NE.0) THEN

C A nonzero element was encountered, so from no on we will have
C to do the sums in the loop above.

          II=i
        END IF
```

```
      B(I)=Sum

    END DO

C Now we do the backsubstitution, equation 2.3.7

    DO I=N,1,-1
      Sum=B(I)
      IF (I.LT.N) THEN
        DO J=I+1,N
          Sum=Sum-A(I,J)*B(J)
        END DO
      END IF

C Store a compononent of the solution vector X.

      B(I)=Sum/A(I,I)
    END DO

    RETURN
    END


    SUBROUTINE LUBksbDP(A,N,Np,Indx,B)

C This routine from Numerical Recipes, p. 37.

C Solves the set of N linear equations A.x=B. Here A is input, not
C as the matrix A but rather as its LU decomposition, determined by the
C routien LUDcmp. Indx is input as the permutation vector returned by
C LUDcmp. B is input as the right-hand side vectore B, and returns with
C the solution vecter X. A,N,Np and Indx are not modified by this routine
C and can be left in place for successive calls with different right-hand
C sides B. This routine takes into account the possiblity that B will begin
C with many zero elements, so it is efficient for use in matrix inversion.

    INTEGER N,Np,Indx(Np),II,I,LL,J
    REAL*8 A(Np,Np),B(Np),Sum

C When II is set ot a positive value, it will become the index of the first
C nonvanishing element of B. We now do the forward substitution, equation
C 2.3.6. The only new wrinkle is to unscramble the permutation as we go.

    II=0

    DO I=1,N
      LL=Indx(I)
      Sum=B(ll)
      B(ll)=B(I)
      IF (ii.NE.0) THEN
        DO J=ii,i-1
          Sum=Sum-A(I,J)*B(J)
        END DO
      ELSE IF (Sum.NE.0) THEN

C A nonzero element was encountered, so from no on we will have
C to do the sums in the loop above.
```

```
          II=i
        END IF
        B(I)=Sum

      END DO

C Now we do the backsubstitution, equation 2.3.7

      DO I=N,1,-1
        Sum=B(I)
        IF (I.LT.N) THEN
          DO J=I+1,N
            Sum=Sum-A(I,J)*B(J)
          END DO
        END IF

C Store a compononent of the solution vector X.

        B(I)=Sum/A(I,I)
      END DO

      RETURN
      END

C ************************************************************************ LUDcmp

      SUBROUTINE LUDcmp(A,N,Np,Indx,D)

C This routine from NUMERICAL RECIPES by W.H. Press, B.P. Flannery,
C S.A. Teukolsky, W.T. Vetterling.

C Given an NxN matrix A, with physical dimension Np, this routine replaces
C it by the LU decomposition of a row-wise permutation of itself. A and
C N are input. A is output, arranged as in eqaution (2.3.14); Indx is an
C output vector which records the row permutation effected by the partial
C pivoting; D is output as + or - 1 depending on whether the number of
C row interchanges was even or odd, respectively. This routine is used
C in combination with LUBKSB to solve linear equations or to invert a
C matrix.

C NMaxLargest expected N
C Tiny  A small number
C VV() The implicit scaling of each row.

      PARAMETER NMax=100
      PARAMETER Tiny=1.0E-20

      INTEGER N,Np,Indx(Np),I,J,K,IMax
      REAL A(Np,Np),VV(NMax),AAMax,Sum,D,Dum

      IF (N.GT.NMax) THEN
        WRITE(*,*) 'Error: matrix dimension too large.'
        WRITE(*,*) '          N=',N
        WRITE(*,*) '       NMax=',NMax
        STOP 'In Routine: LUDcmp'
      END IF

      D=1
```

```
C The following DO-loop loops over rows to get the implicit scaling
C information.

      DO I=1,N

        AAMax=0.
        DO J=1,N
          AAMax=MAX(ABS(A(I,J)),AAMax)
        END DO

        IF (AAMax.EQ.0.0) THEN
          WRITE(*,*) 'Error: Singular Matrix.'
          STOP 'In Routine: LUDcmp'
        END IF

        VV(I)=1.0/AAMax

      END DO

C The following DO-loop loops over the columns of Crout's method.

      DO J=1,N

C The following DO-loop computes equation (2.3.12) except for i=j.

        DO I=1,J-1

          Sum=A(I,J)

          DO K=1,I-1
            Sum=Sum-A(I,K)*A(K,J)
          END DO

          A(I,J)=Sum

        END DO

C The following DO-loop searches for the larges pivot element

        AAMax=0.0
        DO I=J,N
          Sum=A(I,J)
          DO K=1,J-1
            Sum=Sum-A(I,K)*A(K,J)
          END DO
          A(I,J)=Sum

          Dum=VV(I)*ABS(Sum)
          IF (Dum.GT.AAMax) THEN
            IMax=I
            AAMax=Dum
          END IF

        END DO

C The following IF-THEN block makes the row-exchange if it is required.

        IF (J.NE.IMax) THEN
```

```
          DO K=1,N
            Dum=A(Imax,K)
            A(Imax,K)=A(J,K)
            A(J,K)=Dum
          END DO

          D=-D
          VV(IMax)=VV(J)

        END IF

        Indx(J)=IMax
        IF (A(J,J).EQ.0) A(J,J)=Tiny

        IF (J.NE.N) THEN
          Dum=1.0/A(J,J)
          DO I=J+1,N
            A(I,J)=A(I,J)*Dum
          END DO
        END IF
      END DO

      RETURN
      END

C ********************************************************************** LUDcmpDP

      SUBROUTINE LUDcmpDP(A,N,Np,Indx,D)

C This routine from NUMERICAL RECIPES by W.H. Press, B.P. Flannery,
C S.A. Teukolsky, W.T. Vetterling.

C Given an NxN matrix A, with physical dimension Np, this routine replaces
C it by the LU decomposition of a row-wise permutation of itself.  A and
C N are input.  A is output, arranged as in eqaution (2.3.14); Indx is an
C output vector which records the row permutation effected by the partial
C pivoting; D is output as + or - 1 depending on whether the number of
C row interchanges was even or odd, respectively.  This routine is used
C in combination with LUBKSB to solve linear equations or to invert a
C matrix.

C NMax Largest expected N
C Tiny  A small number
C VV()  The implicit scaling of each row.

      PARAMETER NMax=1000
      PARAMETER Tiny=1.0E-20

      INTEGER N,Np,Indx(Np),I,J,K,IMax
      REAL*8 A(Np,Np),VV(NMax),AAMax,Sum,D,Dum

      IF (N.GT.NMax) THEN
        WRITE(*,*) 'Error: matrix dimension too large.'
        WRITE(*,*) '          N=',N
        WRITE(*,*) '       NMax=',NMax
        STOP 'In Routine: LUDcmp'
      END IF
```

```
      D=1

C The following DO-loop loops over rows to get the implicit scaling
C information.

      DO I=1,N

        AAMax=0.
        DO J=1,N
          AAMax=MAX(ABS(A(I,J)),AAMax)
        END DO

        IF (AAMax.EQ.0.0) THEN
          WRITE(*,*) 'Error: Singular Matrix.'
          STOP 'In Routine: LUDcmp'
        END IF

        VV(I)=1.0/AAMax

      END DO

C The following DO-loop loops over the columns of Crout's method.

      DO J=1,N

C The following DO-loop computes equation (2.3.12) except for i=j.

        DO I=1,J-1

          Sum=A(I,J)

          DO K=1,I-1
            Sum=Sum-A(I,K)*A(K,J)
          END DO

          A(I,J)=Sum

        END DO

C The following DO-loop searches for the larges pivot element

        AAMax=0.0
        DO I=J,N
          Sum=A(I,J)
          DO K=1,J-1
            Sum=Sum-A(I,K)*A(K,J)
          END DO
          A(I,J)=Sum

          Dum=VV(I)*ABS(Sum)
          IF (Dum.GT.AAMax) THEN
            IMax=I
            AAMax=Dum
          END IF

        END DO

C The following IF-THEN block makes the row-exchange if it is required.
```

```
      IF (J.NE.IMax) THEN

        DO K=1,N
          Dum=A(Imax,K)
          A(Imax,K)=A(J,K)
          A(J,K)=Dum
        END DO

        D=-D
        VV(IMax)=VV(J)

      END IF

      Indx(J)=IMax
      IF (A(J,J).EQ.0) A(J,J)=Tiny

      IF (J.NE.N) THEN
        Dum=1.0/A(J,J)
        DO I=J+1,N
          A(I,J)=A(I,J)*Dum
        END DO
      END IF
    END DO

    RETURN
    END

C ************************************************************************ MatInv

    SUBROUTINE MatInv(A,Inv,Indx,Col,N,Np)

C This routine from Numerical Recipes p. 38

C Matrix A will be destroyed!

    INTEGER N,Np,Indx(Np),I,J
    REAL A(Np,Np),Inv(Np,Np),Col(Np),D

    DO I=1,N
      Col(I)=0.0
    END DO

    CALL LUDcmp(A,N,Np,Indx,D)

    DO I=1,N

      DO J=1,N
        Col(J)=0.0
      END DO

      Col(I)=1.0
      CALL LUBksb(A,N,Np,Indx,Col)

      DO J=1,N
        Inv(J,I)=Col(J)
      END DO

    END DO
```

```
      RETURN
      END

C ******************************************************************* MatVecMultDP

      SUBROUTINE MatVecMultDP(MaxDim,Dim,Mat,Vec,Res)

      INTEGER MaxDim,Dim,I,J
      REAL*8 Mat(MaxDim,MaxDim),Vec(*),Res(*)

      DO I=1,Dim
        Res(I)=0.D0
        DO J=1,Dim
          Res(I)=Res(I)+Mat(I,J)*Vec(J)
        END DO
      END DO

      RETURN
      END

C ******************************************************************* MatZeroDP

      SUBROUTINE MatZeroDP(RowDim,NoOfRows,NoOfColumns,Matrix)

      INTEGER NoOfRows,NoOfColumns,I,J,RowDim
      REAL*8 Matrix(RowDim,*)

      DO I=1,NoOfRows
        DO J=1,NoOfColumns
          Matrix(I,J)=0.0
        END DO
      END DO

      RETURN
      END

C ******************************************************************* PCubic

      REAL*8 FUNCTION PCubic(XBar,Xi,C,N)

C This function is from Conte and DeBoor p.285. Returns the value at XBar of
C the piecewise cubic function on N intervals with breakpoint sequence Xi and
C coefficients C.

      INTEGER N,I,J
      REAL*8 C(4,N),XBar,Xi(N+1),dX
      DATA I/1/

      IF (I.GT.N+1) I=N+1

      IF (XBar.GE.XI(I)) THEN
         DO J=I,N
              IF (XBar.LT.XI(J+1)) GOTO 30
         END DO
         J=N
      ELSE
         DO J=I-1,1,-1
              IF (XBar.GE.XI(J)) GOTO 30
```

```
          END DO
          J=1
        END IF

30      I=J
        dX=XBar-XI(I)
        PCubic=C(1,I)+dX*(C(2,I)+dX*(C(3,I)+dX*C(4,I)))

        RETURN
        END

C ******************************************************************** PrintMatDP

        SUBROUTINE PrintMatDP(MaxDim,Dim,Mat,Title)

        INTEGER Dim,MaxDim,Length,I,J,K
        REAL*8 Mat(MaxDim,*)
        CHARACTER*(*) Title
        LOGICAL FirstLine

        CALL StringLength(Title,Length)
        IF (Length.GT.0) THEN
          WRITE(*,*) ' '
          WRITE(*,*) Title(1:Length)
          WRITE(*,*) ' '
        END IF

C   12345678    1234567    1234567    1234567    1234567
C Row      Col 1      Col 2      Col 3      Col 4      Col 5
C 12312345123456789a
C iii    ddddddddddd ddddddddddd ddddddddddd ddddddddddd ddddddddddd
C        ddddddddddd ddddddddddd ddddddddddd ddddddddddd ddddddddddd

10      FORMAT(X,'Row',8X,'Col 1',7X,'Col 2',7X,'Col 3',7X,'Col 4',7X,
     & 'Col 5',/)
20      FORMAT(X,I3,3X,5(:,2X,G10.4))
30      FORMAT(X,3X,3X,5(:,2X,G10.4))

        WRITE(*,10)

        DO I=1,Dim
          FirstLine=.FALSE.
          DO J=0,Dim-5,5
            IF (.NOT.FirstLine) THEN
              WRITE(*,20) I,(Mat(I,J+K),K=1,5)
              FirstLine=.TRUE.
            ELSE
              WRITE(*,30) (Mat(I,J+K),K=1,5)
            END IF
          END DO

          IF (J+5.LT.Dim) J=J+5

            IF (.NOT.FirstLine) THEN
                WRITE(*,20) I,(Mat(I,K),K=J+1,Dim)
            ELSE
                WRITE(*,30) (Mat(I,K),K=J+1,Dim)
            END IF
```

```
          IF (FirstLine) WRITE(*,*) ' '

        END DO

        RETURN
        END

C ******************************************************************** PrintMatR

        SUBROUTINE PrintMatR(MaxDim,Dim,Mat,Title)

        INTEGER Dim,MaxDim,Length,I,J,K
        REAL Mat(MaxDim,*)
        CHARACTER*(*) Title
        LOGICAL FirstLine

        CALL StringLength(Title,Length)
        IF (Length.GT.0) THEN
          WRITE(*,*) ' '
          WRITE(*,*) Title(1:Length)
          WRITE(*,*) ' '
        END IF

C    12345678     1234567     1234567     1234567     1234567
C Row       Col 1       Col 2       Col 3       Col 4       Col 5
C 12312345123456789a
C iii     dddddddddd  dddddddddd  dddddddddd  dddddddddd  dddddddddd
C         dddddddddd  dddddddddd  dddddddddd  dddddddddd  dddddddddd

10      FORMAT(X,'Row',8X,'Col 1',7X,'Col 2',7X,'Col 3',7X,'Col 4',7X,
     & 'Col 5'/)
20      FORMAT(X,I3,3X,5(:,2X,G10.4))
30      FORMAT(X,3X,3X,5(:,2X,G10.4))

        WRITE(*,10)

        DO I=1,Dim
          FirstLine=.FALSE.
          DO J=0,Dim-5,5
            IF (.NOT.FirstLine) THEN
              WRITE(*,20) I,(Mat(I,J+K),K=1,5)
              FirstLine=.TRUE.
            ELSE
              WRITE(*,30) (Mat(I,J+K),K=1,5)
            END IF
          END DO

          IF (J+5.LT.Dim) J=J+5

            IF (.NOT.FirstLine) THEN
                WRITE(*,20) I,(Mat(I,K),K=J+1,Dim)
            ELSE
                WRITE(*,30) (Mat(I,K),K=J+1,Dim)
            END IF

          IF (FirstLine) WRITE(*,*) ' '

        END DO
```

```
      RETURN
      END

C ******************************************************************** ReadSIMEQ

      SUBROUTINE ReadSIMEQ(FileName)

      CHARACTER*(*) FileNAme
      INCLUDE 'SIMEQ.INC'
      INTEGER I,J

      OPEN(UNIT=1,FILE=FileNAme,STATUS='OLD',FORM='FORMATTED')
      READ(1,*) N1,IFlag

      DO I=1,N1
        READ(1,*) IPivot(I)
      END DO

      DO I=1,N1
        DO J=1,N1
          READ(1,*) W1(I,J)
        END DO
      END DO

      CLOSE(1)

      RETURN
      END

C ************************************************************************ Slnpd

      SUBROUTINE Slnpd(A,B,Determinant,N,Nx)

C Solultion of linear systems of equations by the gauss elimination method
C providing for interchanging rows when encountering a
C zero diagonal coefficient

C A       System Matrix (Ax=B)
C B       Originally contains the independent coefficients.  After solution
C         it contains the values of the system unknowns.(x)
C N       Actual number of unknowns
C Nx      Row and Column dimension of A
C Determinant   The determinant of the matrix A
C Tolerance     Values below this are interpreted as zero.

      INTEGER Nx,N
      REAL*8 A(Nx,Nx),B(Nx),Determinant,Tolerance/0.000001/

      INTEGER K,JLoc,J,L,I
      REAL*8 C

      Determinant=0.0

      DO K=1,N-1
        IF (ABS(A(K,K)).LT.Tolerance) THEN

C Try to interchange rows to get non-zero diagonal coefficients
```

```
          JLoc=0
          DO J=K+1,N
              IF ((ABS(A(J,K)).GT.Tolerance).AND.(JLoc.NE.0)) THEN
                JLoc=J
              END IF
          END DO

          IF (JLoc.EQ.0) THEN

            WRITE(6,2) K
2           FORMAT(X,'**** Singularity in row ',I5)
            RETURN

          END IF

          DO L=K,N
              CALL DPXchng(A(K,L),A(J,L))
          END DO
          CALL DPXchng(B(K),B(J))

        END IF

C Divide row by the diagonal coefficient

        DO J=K+1,N
          A(K,J)=A(K,J)/A(K,K)
        END DO
        B(K)=B(K)/A(K,K)

C Eliminate unknown X(K) from row i

        DO I=K+1,N
          C=A(I,K)
          DO J=K+1,N
              A(I,J)=A(I,J)-C*A(K,J)
          END DO
          B(I)=B(I)-C*B(K)
        END DO
      END DO

C Compute Last Unknown

      IF (ABS(A(N,N)).LT.Tolerance) THEN
        WRITE(6,2) N
        RETURN
      END IF

      B(N)=B(N)/A(N,N)

C Apply backsubstitution process to compute remaining unknowns

      DO L=1,N-1
        K=N-L
        DO J=K+1,N
          B(K)=B(K)-A(K,J)*B(J)
        END DO
      END DO
```

```
C Compute Determinant

      Determinant=1.0
      DO I=1,N
        Determinant=Determinant*A(I,I)
      END DO

      RETURN
      END

C ******************************************************************** SolveRelEq

      SUBROUTINE SolveRelEq(B,X)

      REAL*8 B(*),X(*)

      INCLUDE 'SIMEQ.INC'

      IF (IFlag.NE.0) THEN
        CALL Subst(W1,IPivot,B,N1,X,MaxNoOfEqus)
      ELSE
        WRITE(*,200)
200     FORMAT(X,'Warning: the system of linear equations is',
     & ' not solvable.')
        STOP 'In Routine: SolveRelEq'
      END IF

      RETURN
      END

C ******************************************************************** SolveSimEq

      SUBROUTINE SolveSimEq(W,B,N,X,MaxDim)

C Variables

C W     The (N,N) matrix of coefficients.
C B     The (N) vector, WX=B
C X     The Solution (N)
C MaxDim The maximum dimension of the matrices.

      INCLUDE 'SIMEQ.INC'

      INTEGER N,MaxDim,I,J
      REAL*8 W(MaxDim,MaxDim),D(MaxNoOfEqus),B(*),X(*)

      IF (N.GT.MaxNoOfEqus) THEN
        WRITE(*,100) N,MaxNoOfEqus
100     FORMAT(X,'Warning: There are ',I4,'simultaneous equations',
     & /,X,'This routine SolveSimEq can only solve up to ',I4,'.')
        STOP 'In Routine: SolveSimEq'
      END IF

C Copy the Maxtrix to a dummy matrix so that original matrix is not destroyed
C by call to Factor.  Also, this allows a routine SolveRelEq to be used
C to solve for related problems W.x=b'#b

      N1=N
```

```
      DO I=1,N
        DO J=1,N
          W1(I,J)=W(I,J)
        END DO
      END DO

      CALL Factor(W1,N1,D,IPivot,IFlag,MaxNoOfEqus)

      IF (IFlag.NE.0) THEN
        CALL Subst(W1,IPivot,B,N1,X,MaxNoOfEqus)
      ELSE
        WRITE(*,200)
200     FORMAT(X,'Warning: the system of linear equations is',
     & ' not solvable.')
        STOP 'In Routine: SolveSimEq'
      END IF

      RETURN
      END

C ***************************************************************** SolveTriDiag

      SUBROUTINE SolveTriDiag(Sub,Diag,Sup,B,N,X)

C This subroutine solves a tridiagonal linear system of equations:
C    Sub(I)*X(I-1) + Diag(I)*X(I) + Sup(I)*X(I+1) = B(I), I=1,...,N
C (With Sub(1) and Sup(N) taken to be zero) by factorization and
C substitution.

C (Taken from Conte and de Boor, ELEMENTARY NUMERICAL ANALYSIS, 3d. ed.
C  McGraw Hill, p.155)

C Input Variables

C Sub(N)        The subdiagonal of the Matrix.
C Diag(N)       The diagonal of the Matrix
C Sup(N)        The superdiagonal of the matrix.
C B(N)          The load vector.
C N             The number of equations

C Output Variables

C X(N)          The solution

C Internal Variables

C I       Loop counter
C Beta(100)     The load vector after the elimination
C Gamma(100)    The diagonal after the elimination

      REAL*8 Sub(*),Diag(*),Sup(*),B(*),X(*)
      REAL*8 Beta(100),Gamma(100)
      INTEGER N,I

      IF (N.GT.100) THEN
        WRITE(*,100) N
100     FORMAT(X,'Warning: too many simultaneous equations.',
     & /,X,'N=',I3,' and N must be <=100')
```

```
      STOP 'In Routine: SolveTriDiag'
    END IF

    IF (N.LE.1) THEN
      X(1)=B(1)/Diag(1)
      RETURN
    END IF

    Gamma(1)=Diag(1)
    Beta(1)=B(1)

    DO I=2,N
      Gamma(I)=Diag(I)-Sub(I)*Sup(I-1)/Gamma(I-1)
      Beta(I)=B(I)-Sub(I)*Beta(I-1)/Gamma(I-1)
    END DO

    X(N)=Beta(N)/Gamma(N)
    DO I=N-1,1,-1
      X(I)=(Beta(I)-Sup(I)*X(I+1))/Gamma(I)
    END DO

    RETURN
    END

C ************************************************************************ Sparse

    SUBROUTINE Sparse(B,N,X,Rsq)

C This subroutine solves the linear system A.X=b for the vector X of length
C N, given the right-hand vector B, and given two subroutines, ASub(Xin,Xout)
C and ATSub(XIn,XOut), which respectively calculate Ax and ATx for x given
C as their first arguments, returning the result in their second arguments.
C These subroutines should take every advantage of the sparseness of
C the matrix A. On input, X shoul be set to a first guess of the desired
C solution (all zero components is fine). On output, X is the solution
C vector, and Rsq is the sum of the squares of the components of the residual
C vector A.X-b. If this is not small, then the matrix is numerically singular
C and the solution represents a least-squares best approximation.

C NMaxThe maximum expected N
C Eps  RMS Accuracy desired
C Eps2 Criterion for sum-squared residuals
C IRst THe number of restarts attempted internally.

    PARAMETER NMax=700
    PARAMETER Eps=1.0E-6

    INTEGER N,IRst,J,Iter
    REAL*8 B(*),X(*),G(NMax),H(NMax),Xi(NMax),XJ(NMax)
    REAL*8 Eps2,Rp,Bsq,ANum,ADen,Rsq,GG,DGG,Gam

    Eps2=N*Eps*Eps

    Irst=0
1   IRst=IRst+1

    CALL ASub(X,XI)
```

```
      Rp=0
      Bsq=0
      DO J=1,N
        Bsq=Bsq+B(J)**2
        Xi(J)=XI(J)-B(J)
        RP=RP+XI(J)**2
      END DO

      CALL ATSub(Xi,G)

      DO J=1,N
        G(J)=-G(J)
        H(J)=G(J)
      END DO

      DO Iter=1,10*N

        CALL ASub(H,XI)

        ANum=0.
        ADen=0.

        DO J=1,N
          ANum=ANum+G(J)*H(J)
          ADen=ADen+XI(J)**2
        END DO

        IF (ADen.EQ.0.) THEN
          WRITE(*,*) 'Error: Singular Matrix'
          WRITE(*,*) '       Iteration number ',Iter
          WRITE(*,*) '       X Vector:'
          WRITE(*,*) (H(J),J=1,N)
          STOP 'In Routine: SPARSE'
        END IF

        ANum=ANum/Aden
        DO J=1,N
          XI(J)=X(J)
          X(J)=X(J)+ANum*H(J)
        END DO

        CALL ASub(X,Xj)

        Rsq=0.

        DO J=1,N
          XJ(J)=XJ(J)-B(J)
          Rsq=Rsq+XJ(J)**2
        END DO

C The following IF statement tests for convergence.

        IF ((Rsq.EQ.RP).OR.(Rsq.LE.BSQ*Eps2)) RETURN

        IF (Rsq.GT.Rp) THEN

C Not improving.  Do a restart
```

```
          DO J=1,N
            X(J)=XI(J)
          END DO

C Return if too many restarts.  This is the normal return when we run into
C roundoff error before satisfying the return above.

          IF (IRst.GE.3) RETURN

          GOTO 1
        END IF

        Rp=Rsq

        CALL ATSub(Xj,Xi)

        GG=0.
        Dgg=0.

        DO J=1,N
          GG=GG+G(J)**2
          DGG=DGG+(Xi(J)+G(J))*XI(j)
        END DO

C A Rare but normal return.

        IF (GG.EQ.0) RETURN

        Gam=Dgg/GG
        DO J=1,N
          G(J)=-Xi(j)
          H(J)=G(J)+Gam*H(J)
        END DO

      END DO

      WRITE(*,*) 'Error: too many iterations.'
      WRITE(*,*) '       Number of iterations=',10*N

      STOP 'In Routine: Sparse'

      END

C ************************************************************************ Spline

      SUBROUTINE Spline(Xi,C,N)

C This subroutine from Conte and DeBoor p.290

C Variables-----------------------------------------------------------------

C Input Variables

C XI(1),...,XI(N+1)  Strictly increasing sequence of breakpoints
C C(1,I),C(2,I)      Value and 1st derivative at XI(I), I=1,...,N+1
C        of the cubic spline

C Output Variables
```

```
C C(1,I),C(2,I),C(3,I),C(4,I) Polynomial Coefficients of the spline
C       on the interval (XI(I),XI(I+1)), I=1,...,N.

        PARAMETER Np1Max=1000
        INTEGER N,M,Offset
        REAL*8 C(4,N+1),XI(N+1),D(Np1Max),Diag(Np1Max),G
        DATA Diag(1)/1.0/,D(1)/0.0/

        IF (N+1.GT.Np1Max) THEN
          WRITE(*,*) 'Error: N is too large.  N=',N
          WRITE(*,*) '        Maximum N Allowed=',Np1Max-1
          STOP 'In Routine: Spline'
        END IF

        Offset=0
        DO M=2,N+1
          IF (XI(M)-XI(M-1).LE.0) THEN
            Offset=Offset+1
            WRITE(*,*) 'Warning: Sequence of breakpoints is',
     & ' not strictly increasing'
            WRITE(*,*) ' Used: XI(',M,')=',XI(M),C(1,M)
            WRITE(*,*) '       XI(',M-1,')=',XI(M-1),C(1,M-1)
          END IF
          XI(M-Offset)=XI(M)
          C(1,M-Offset)=C(1,M)
          C(2,M-Offset)=C(2,M)
          C(3,M-Offset)=C(3,M)
          C(4,M-Offset)=C(4,M)
        END DO
        N=N-Offset

        DO M=2,N+1
           D(M)= XI(M)-XI(M-1)
           IF (D(M).LE.0) THEN
           END IF
           Diag(M)=(C(1,M)-C(1,M-1))/D(M)
        END DO

        DO M=2,N
           C(2,M)=3.0*(D(M)*Diag(M+1)+D(M+1)*Diag(M))
           Diag(M)=2.0*(D(M)+D(M+1))
        END DO

        DO M=2,N
           G=-D(M+1)/Diag(M-1)
           Diag(M)=Diag(M)+G*D(M-1)
           C(2,M)=C(2,M)+G*C(2,M-1)
        END DO

        DO M=N,2,-1
           C(2,M)=(C(2,M)-D(M)*C(2,M+1))/Diag(M)
        END DO

        RETURN
        END

C ************************************************************************** Subst
```

```
      SUBROUTINE Subst(W,IPivot,B,N,X,MaxDim)

C This subroutine uses the factorization from FACTOR to solve the system
C of linear equations W*X=B (where W is the original matrix input to FACTOR.
C Taken from Conte and DeBoor, p164.

C Input Variables

C W(N,N)       The matrix output from FACTOR
C IPivot(N)    The pivoting matrix output from FACTOR
C N            The dimension of the matrix.
C B(N)         The right side of the system to be solved.

C Output Variables

C X(N)         The vector satisfying W*X=B

      INTEGER IPivot(*),I,IP,J,MaxDim,N
      REAL*8 B(*),W(MaxDim,MaxDim),X(*),Sum

      IF (N.LE.1) THEN
        X(1)=B(1)/W(1,1)
        RETURN
      END IF

      Ip=IPivot(1)
      X(1)=B(IP)
      DO I=2,N
        Sum=0.0
        DO J=1,I-1
          Sum=W(I,J)*X(J)+Sum
        END DO
        IP=IPivot(I)
        X(I)=B(IP)-Sum
      END DO

      X(N)=X(N)/W(N,N)
      DO I=N-1,1,-1
        Sum=0.0
        DO J=I+1,N
          Sum=W(I,J)*X(J)+Sum
        END DO
        X(I)=(X(I)-Sum)/W(I,I)
      END DO

      RETURN
      END

C ********************************************************************** VecNormDP

      REAL*8 FUNCTION VecNormDP(NoOfElem,Vec)

C This subroutine Computes the norm of Vec

      INTEGER I,NoOfElem
      REAL*8 Vec(*),Norm

      Norm=0.0
```

```
      DO I=1,NoOfElem
        Norm=Norm+Vec(I)*Vec(I)
      END DO

      VecNormDP=SQRT(Norm)

      RETURN
      END

C ******************************************************************* VecScaMultDP

      SUBROUTINE VecScaMultDP(Dim,Vec,Sca,Res)

      INTEGER Dim,I,J
      REAL*8 Vec(*),Res(*),Sca

      DO I=1,Dim
        Res(I)=Vec(I)*Sca
      END DO

      RETURN
      END

C ********************************************************************** VecSumDP

      REAL*8 FUNCTION VecSumDP(NoOfElems,Vector)

C This subroutine sums the elements from 1 to NoOfElems in Vector(*). The
C result is stored in Sum

      REAL*8 Vector(*),Sum
      INTEGER NoOfElems,I

      Sum=0.0D0
      DO I=1,NoOfElems
        Sum=Sum+Vector(I)
      END DO

      VecSumDP=Sum

      RETURN
      END

C ********************************************************************** VecZeroDP

      SUBROUTINE VecZeroDP(NoOfElems,Vector)

      REAL*8 Vector(*)
      INTEGER NoOfElems,I

      DO I=1,NoOfElems
        Vector(I)=0.D0
      END DO

      RETURN
      END

C ******************************************************************* WriteSIMEQ
```

```
      SUBROUTINE WriteSIMEQ(FileName)

      CHARACTER*(*) FileNAme
      INCLUDE 'SIMEQ.INC'
      INTEGER I,J

      OPEN(UNIT=1,FILE=FileNAme,STATUS='UNKNOWN',FORM='FORMATTED')
      WRITE(1,*) N1,',',IFlag

      DO I=1,N1
        WRITE(1,*) IPivot(I)
      END DO

      DO I=1,N1
        DO J=1,N1
          WRITE(1,*) W1(I,J)
        END DO
      END DO

      CLOSE(1)

      RETURN
      END

c *
```

LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
INFORMATION RESOURCES DEPARTMENT
BERKELEY, CALIFORNIA 94720