APPENDIX A7 – Fortran elasto – plastic finite element program

SUBROUTINE **ALGOR**

```
Subroutine algor(fixed,iincs,iiter,kresl,mtotv,nalgo,ntotv)
!**********************************************************************
!
!***** This subroutine sets equation resolution index,kresl
!
!**********************************************************************
!
implicit none
!
        integer :: kresl,nalgo,iiter,itotv,ntotv,iincs,mtotv
        double precision :: fixed(mtotv)
!
! KRESL=1 - indicates reformulation of the elem. stiffnesses accompanied
!                 by full eqn. resolution
! KRESL=2 - the element stiffnesses are not to be modified & so only 1 eqn.
!                 resolution takes place
!
        kresl=2
        if (nalgo.eq.1.and.iincs.eq.1.and.iiter.eq.1) kresl=1
        if (nalgo.eq.2) kresl=1
        if (nalgo.eq.3.and.iiter.eq.1) kresl=1
        if (nalgo.eq.4.and.iincs.eq.1.and.iiter.eq.1) kresl=1
        if (nalgo.eq.4.and.iiter.eq.2) kresl=1
        if (iiter.eq.1) return
!
        do 100 itotv=1,ntotv
                fixed(itotv)=0.0
        100 continue
!
return
end
```

SUBROUTINE **BMATPS**

```
Subroutine bmatps(bmatx,cartd,nnode,shap,gpcod,ntype,kgasp)
!**********************************************************************
!
!***** This subroutine evaluates the strain displacement matrix B
!                 at any position within an element
!
!**********************************************************************
!
implicit none
!
        integer :: ngash,mgash,kgasp
        double precision :: bmatx(4,18),cartd(2,9),shap(9),gpcod(2,9)
        integer :: ju,iu,ntype,nnode,inode
```

```
!
        ngash=0
!
        do 10 inode=1,nnode !'nnode' is no. of nodes per element
                mgash=ngash+1
                ngash=mgash+1
                bmatx(1,mgash)=cartd(1,inode)
                bmatx(1,ngash)=0.0
                bmatx(2,mgash)=0.0
                bmatx(2,ngash)=cartd(2,inode)
                bmatx(3,mgash)=cartd(2,inode)
                bmatx(3,ngash)=cartd(1,inode)
                if (ntype.ne.3) goto 10
                bmatx(4,mgash)=shap(inode)/gpcod(1,kgasp)
                bmatx(4,ngash)=0.0
        10 continue

return
end
```

SUBROUTINE **CHECK1**

```
Subroutine check1(ndofn,nelem,ngaus,nmats,nnode,npoin,nstre,ntype,nvfix, &
                                ncrit,nalgo,nincs)
!**********************************************************************
!
!***** This subroutine checks the main control data
!
!**********************************************************************
!
implicit none
!
        integer :: ieror,keror,nstre,nalgo,ngaus,ncrit,nmats,ndofn,ntype,nincs
        integer :: nvfix,nnode,nelem,npoin
        integer ::  neror(24)
                do 10 ieror=1,12 !See definition on page 201
                        10 neror(ieror)=0 !zero all error codes
!
!*** create the diagnostic messages
!
        if (npoin.le.0) neror(1)=1
        if (nelem*nnode.lt.npoin) neror(2)=1
        if (nvfix.lt.2.or.nvfix.gt.npoin) neror(3)=1
        if (nincs.lt.1) neror(4)=1
        if (ntype.lt.1.or.ntype.gt.3) neror(5)=1
        if (nnode.lt.4.or.nnode.gt.9) neror(6)=1
        if (ndofn.lt.2.or.ndofn.gt.5) neror(7)=1
        if (nmats.lt.1.or.nmats.gt.nelem) neror(8)=1
        if (ncrit.lt.1.or.ncrit.gt.4) neror(9)=1
        if (ngaus.lt.2.or.ngaus.gt.3) neror(10)=1
        if (nalgo.lt.1.or.nalgo.gt.4) neror(11)=1
        if (nstre.lt.3.or.nstre.gt.5) neror(12)=1
!
```

```
!*** either return, or else print the errors diagnosed
!
        keror=0
        do 20 ieror=1,12
                if (neror(ieror).eq.0) goto 20
                keror=1
                write (6,900) ieror
                900 format (//a31 "*** diagnosis by check1", "error",i3)
        20 continue
        if (keror.eq.0) return
!
!*** otherwise echo all the remaining data without further comment
!
        call echo
end !should terminate program
```

SUBROUTINE **CHECK2**

```
Subroutine check2(coord,iffix,lnods,matno,melem,mfron,mpoin,mtotv, &
                        mvfix,ndfro,ndofn,nelem,nmats,nnode,nofix,npoin,nvfix)
!**********************************************************************
!
!***** This subroutine checks the remainder of the input data
!***** Description given on page 215 'finite element programming'-hinton and owen
!
!**********************************************************************
!
implicit none
!
        integer :: ielem,nelem,ipoin,npoin,kpoin,jpoin,idime,nmats,inode,node
        integer :: mpoin,mtotv,melem,mvfix,ieror,kstar,kzero,nnode,ndofn,klast,nlast
        integer :: ivfix,nvfix,nfron,kfron,mfron,kount,nloca,idofn,kvfix,jvfix,keror
        double precision :: sigma
        double precision :: coord(mpoin,2)
        integer :: matno(melem),ndfro(melem),neror(24),nofix(mvfix),iffix(mtotv)
        integer :: lnods(melem,9)

!
!*** check against two identical nonzero nodal coordinates
!
        do 5 ieror=13,24
                5 neror(ieror)=0
        do 10 ielem=1,nelem !set the NDFRO array to zero. This will store the
                                                ! contribution to the frontwidth of each element
                10 ndfro(ielem)=0
        do 40 ipoin=2,npoin !enter loop over all nodal points, except the first
                kpoin=ipoin-1 !enter loop over all nodal points up to,but not including,
                                                !the node being currently checked
                do 30 jpoin=1,kpoin
! check if the coordinates of the current node are equal to those of one or more
!of the preceding nodes. If not,goto 30,thereby bypassing the error counter
                        do 20 idime=1,2
                                if (coord(ipoin,idime).ne.coord(jpoin,idime)) goto 30
```

```
                        20 continue
                        neror(13)=neror(13)+1
                30 continue
        40 continue
!
!*** check the list of element property numbers
!
        do 50 ielem=1,nelem
                50      if (matno(ielem).le.0.or.matno(ielem).gt.nmats) neror(14)=neror(14)+1
!
!*** check for impossible node numbers
!
        do 70 ielem=1,nelem
                do 60 inode=1,node
!if any element nodal connection no. is zero inc. the count for error 15 by 1
!Array lnods stores the nodal connections which forms each element
                        if (lnods(ielem,inode).eq.0) neror(15)=neror(15)+1
!If any element nodal connection no. is -ve or > the specified tot. no. of
!nodes in the structure,NPOIN, than inc. counter for error 16 by 1
                60 if (lnods(ielem,inode).lt.0.or.lnods(ielem,inode).gt.npoin) &
                neror(16)=neror(16)+1
        70 continue
!
!*** check for any repetition of a node number within an element
!
        do 140 ipoin=1,npoin !enter loop over each node in the structure
!kstar used to denote the element in which the node currently considered 1st appears
                kstar=0
                do 100 ielem=1,nelem
!kzero used to indicate how many times a particular node appears in any element
                        kzero=0
                        do 90 inode=1,nnode !enter loop over each node in the elem.
                                if(lnods(ielem,inode).ne.ipoin) goto 90
                                kzero=kzero+1
                                if (kzero.gt.1) neror(17)=neror(17)+1
!
!*** seek first,last and intermediate appearances of node ipoin
!
                                if (kstar.ne.0) goto 80
                                kstar=ielem
!
!*** calculate increase or decrease in frontwidth at each element stage
!the no. of unknowns in the front is the frontwidth
                                ndfro(ielem)=ndfro(ielem)+ndofn
                                80 continue
!
!*** and change the sign of the last appearance of each node
!
                klast=ielem
                nlast=inode
                        90 continue
                100 continue
                if (kstar.eq.0) goto 110
                if (klast.lt.nelem) ndfro(klast+1)=ndfro(klast+1)-ndofn
        !each last appearance is labelled by introducing a -ve sign before the
        !nodal connection no. in the LNODS array
```

```
                lnods(klast,nlast)=-ipoin
                goto 140
!
!*** check that coordinates for an unused node have not been specified
!
                110 write(6,900) ipoin
                900 format(/"check why node",i4,"never appears")
                neror(18)=neror(18)+1
                sigma=0.0
                do 120 idime=1,2
                        120 sigma=sigma+abs(coord(ipoin,idime))
                if (sigma.ne.0.0) neror(19)=neror(19)+1
!
!*** check that an unused node number is not a restrained node
!
                do 130 ivfix=1,nvfix
                        130 if (nofix(ivfix).eq.ipoin) neror(20)=neror(20)+1
        140 continue
!
!*** calculate the largest frontwidth
!
!Initialize to zero the current frontwidth,NFRON, and the maximum
!encountered to date,KFRON
        nfron=0
        kfron=0
        do 150 ielem=1,nelem
                nfron=nfron+ndfro(ielem)
        150 if (nfron.gt.kfron) kfron=nfron
        write (6,905) kfron
        905 format (//"maximum frontwidth encountered =",i5)
        if (kfron.gt.mfron) neror(21)=1
!
!*** continue checking the data for the fixed values
!
        do 170 ivfix=1,nvfix
                if (nofix(ivfix).le.0.or.nofix(ivfix).gt.npoin) neror(22)=neror(22)+1
                kount=0
                nloca=(nofix(ivfix)-1)*ndofn
                do 160 idofn=1,ndofn
                        nloca=nloca+1
                        160 if (iffix(nloca).gt.0) kount=1
                if (kount.eq.0) neror(23)=neror(23)+1
                kvfix=ivfix-1
        do 170 jvfix=1,kvfix
                170 if (ivfix.ne.1.and.nofix(ivfix).eq.nofix(jvfix)) &
                        neror(24)=neror(24)+1
        keror=0
        do 180 ieror=13,24
                if (neror(ieror).eq.0) goto 180
                keror=1
                write(6,910) ieror,neror(ieror)
                910 format(//"*** diagnosis by check2, error",i3,6x,"associated &
                number",i5)
        180 continue
!if any errors have been detected branch to call subroutine ECHO
if (keror.ne.0) goto 200
```

```
!
!*** return all nodal connection numbers to positive values
!
        do 190 ielem=1,nelem
        do 190 inode=1,nnode
                190 lnods(ielem,inode)=iabs(lnods(ielem,inode))
return
200 call echo
end
```

SUBROUTINE **CONVER**

```
Subroutine conver(eload,iiter,lnods,melem,mevab,mtotv,nchek &
                ,ndofn,nelem,nevab,nnode,ntotv,pvalu,stfor, &
                tload,tofor,toler)
!**********************************************************************
!
!***** This subroutine checks for convergence of the iteration process
!
!**********************************************************************
!
implicit none
!
        integer :: itotv,nposi,ndofn,idofn,locno,nnode,inode,kevab,nelem,ielem
        integer :: iiter,nevab,ievab,ntotv,nchek,melem,mevab,mtotv
        double precision :: pvalu,toler,ratio,agash,refor,remax,retot,resid
        double precision :: eload(melem,mevab),stfor(mtotv),tofor(mtotv),tload(melem,mevab)
        integer :: lnods(melem,9)
!
        nchek=0 !initialise the convergence indicator to zero
        resid=0.0 !initialise to zero the norm. of the residual forces
        retot=0.0 !initialise to zero the norm. of the total applied loads
        remax=0.0
!initialise the arrays which will contain the eqv. nodal forces & the applied loads
!for each nodal point
        do 5 itotv=1,ntotv
                stfor(itotv)=0.0
                tofor(itotv)=0.0
        5 continue
!
!Assemble the eqv. nodal forces & the applied load contributions of each element
!to give the total nodal values
        do 40 ielem=1,nelem
                kevab=0
        do 40 inode=1,nnode
                locno=iabs(lnods(ielem,inode))
        do 40 idofn=1,ndofn
                kevab=kevab+1
                nposi=(locno-1)*ndofn+idofn
                stfor(nposi)=stfor(nposi)+eload(ielem,kevab)
        40      tofor(nposi)=tofor(nposi)+tload(ielem,kevab)
!
        do 50 itotv=1,ntotv
```

```
                refor=tofor(itotv)-stfor(itotv) !calculate the nodal residual force according to 2.4
                resid=resid+refor*refor !evaluate the norm of the residual forces
                retot=retot+tofor(itotv)*tofor(itotv) !evaluate the norm of the total applied forces
                agash=abs(refor)                                ! evaluate the max. residual force
        50      if (agash.gt.remax) remax=agash ! existing in the structure as an additional check
                                                                                ! on the nonlinear
solution
!
! compute the residual nodal forces for each element, for application as forces
! for the next iteration according to 2.12
        do 10 ielem=1,nelem
        do 10 ievab=1,nevab
        10      eload(ielem,ievab)=tload(ielem,ievab)-eload(ielem,ievab)
!
        resid=sqrt(resid)
        retot=sqrt(retot)
        ratio=100.0*resid/retot !compute the residual sum ratio
        if (ratio.gt.toler) nchek=1 !convergence has not yet occured
        if (iiter.eq.1) goto 20
        if (ratio.gt.pvalu) nchek=999 !check to see whether problem is converging
20      pvalu=ratio
        write(6,30) nchek,ratio,remax
30      format (3x,"convergence code =",i4,3x,"norm of residual sum ratio =",e14.6,3x, &
                "maximum residual =",e14.6)
return
end
```

SUBROUTINE **DBE**

```
Subroutine dbe(bmatx,dbmat,dmatx,mevab,nevab,nstre,nstr1)
!**********************************************************************
!
!***** This subroutine multiplies the D matrix by the B matrix
!
!**********************************************************************
!
implicit none
!
        integer :: istre,nstre,ievab,nevab,jstre,nstr1,mevab
        double precision :: bmatx(nstr1,mevab),dbmat(nstr1,mevab),dmatx(nstr1,nstr1)

!
        do 2 istre=1,nstre
        do 2 ievab=1,nevab
                dbmat(istre,ievab)=0.0
        do 2 jstre=1,nstre
                dbmat(istre,ievab)=dbmat(istre,ievab)+dmatx(istre,jstre)*bmatx(jstre,ievab)
        2       continue
return
end
```

SUBROUTINE **DIMEN**

```
Subroutine dimen(mbufa,melem,mevab,mfron,mmats,mpoin,mstif,mtotg, &
                                 mtotv,mvfix,ndofn,nprop,nstre)
!**********************************************************************
!
!***** This subroutine presets variables associated with dynamic dimensioning
!
!**********************************************************************
!
implicit none
!
integer :: mbufa,melem,mfron,mmats,mpoin,mstif,mtotg,ndofn,mtotv
integer :: nstre,mvfix,nprop,mevab

!
        mbufa=10
        melem=1000 !40 to increase size of model
        mfron=110 !set the max. possible frontwidth,MFRON,equal to the same
                                        !value defined in subroutine FRONT
        mmats=5
        mpoin=3400 !150 to increase size of model
        mstif=(mfron*mfron-mfron)/2+mfron          !removed .0  in order to be integer
        mtotg=melem*9
        ndofn=2
        mtotv=mpoin*ndofn
        mvfix=500 !30 to increase size of model
        nprop=7
        mevab=ndofn*9
return
end
```

SUBROUTINE **ECHO**

```
Subroutine echo
!**********************************************************************
!
!***** If data errors have been detected by subroutines check1 or
!       check2, this subroutine reads and writes the remaining data cards
!
!**********************************************************************
!
implicit none

        character(80) :: ntitl
!
        write (6,900)
        900 format (//a50 "** now follows a listing of post disaster data **" /)
        10 read(5,905) ntitl
                905 format (80a1)
                write (6,910) ntitl
                910 format (20x,80a1)
```

```
        goto 10
end !should terminate the program
```

SUBROUTINE **FLOWPL**

```
Subroutine flowpl(avect,abeta,dvect,ntype,props,lprop,nstr1,mmats)
!
!**********************************************************************
!
!***** This subroutine evaluates the plastic D vector (dD) see pg243
!
!**********************************************************************
!
implicit none
!
        integer :: lprop,mmats,ntype,nstr1,istr1
        double precision :: young,poiss,hards,fmul1,abeta,denom,fmul3,fmul2
        double precision :: avect(4),dvect(4),props(mmats,7)
!
!       write(60,*) 'flowpl ntype',ntype

        young=props(lprop,1)
        poiss=props(lprop,2)
        hards=props(lprop,6)
        fmul1=young/(1.0+poiss)
        if (ntype.eq.1) goto 60 !for plane stress
!
! for plane strain and axisymmetric problems
        fmul2=young*poiss*(avect(1)+avect(2)+avect(4))/((1.0+poiss)* &
                (1.0-2.0*poiss))
        dvect(1)=fmul1*avect(1)+fmul2
        dvect(2)=fmul1*avect(2)+fmul2
        dvect(3)=0.5*avect(3)*young/(1.0+poiss)
        dvect(4)=fmul1*avect(4)+fmul2
        goto 70
!
! for plane stress
        60 fmul3=young*poiss*(avect(1)+avect(2))/(1.0-poiss*poiss)
        dvect(1)=fmul1*avect(1)+fmul3
        dvect(2)=fmul1*avect(2)+fmul3
        dvect(3)=0.5*avect(3)*young/(1.0+poiss)
        dvect(4)=fmul1*avect(4)+fmul3
        70 denom=hards
        do 80 istr1=1,nstr1
        80 denom=denom+avect(istr1)*dvect(istr1)
        abeta=1.0/denom
return
end
```

SUBROUTINE **FRONT**

```
Subroutine front(asdis,eload,eqrhs,equat,estif,fixed,iffix,iincs, &
                 iiter,gload,gstif,locel,lnods,kresl,mbufa,melem, &
                 mevab,mfron,mstif,mtotv,mvfix,nacva,namev,ndest, &
                 ndofn,nelem,nevab,nnode,nofix,npivo,npoin, &
                 ntotv,tdisp,tload,treac,vecrv)
!**********************************************************************
!
!***** This subroutine solves the equation by the frontal method
!
!**********************************************************************
!
implicit none
!
!       dimension       asdis(mtotv),eload(melem,mevab),eqrhs(mbufa),equat(mfron,mbufa), &
!                       estif(mevab,mevab),fixed(mtotv),iffix(mtotv),npivo(mbufa), &
!                       vecrv(mfron),gload(mfron),gstif(mstif),lnods(melem,9),locel(mevab), &
!                       nacva(mfron),namev(mbufa),ndest(mevab),nofix(mvfix),noutp(2), &
!                       tdisp(mtotv),tload(melem,mevab),treac(mvfix,ndofn), &
!locel1(mevab),nacva1(mfron),ndest1(mevab) !mine for testing
!mine
integer :: mbufa,melem,mfron,mmats,mpoin,mstif,mtotg,ndofn,mtotv,ielem,kelva
integer :: nstre,mvfix,nprop,mevab,nbufa,ibufa,ifron,istif,inode,kevab
integer :: ngash,jdest,idest,jevab,ievab,kexis,lfron,jfron,kboun,ntotv,itotv
integer :: ievla,nloca,ngish,mgash,idofn,ipoin,ngush,nikno,nevab,locno,nposi
integer :: nfron,kresl,nlast,nnode,nelem,klast,npoin,iiter,i,iincs,j !,nfunc
integer :: mus,joo,ioo,mar
integer :: iffix(mtotv),npivo(mbufa),lnods(melem,9),locel(mevab)
integer :: nacva(mfron),namev(mbufa),ndest(mevab),nofix(mvfix),noutp(2)
integer :: locel1(mevab),nacva1(mfron),ndest1(mevab),nfron1 !,nfunc(100,100)
double precision :: cureq,pivot
double precision :: asdis(mtotv),eload(melem,mevab),eqrhs(mbufa),equat(mfron,mbufa)
double precision :: estif(mevab,mevab),fixed(mtotv),vecrv(mfron),gload(mfron),gstif(mstif)
double precision :: tdisp(mtotv),tload(melem,mevab),treac(mvfix,ndofn)
!
        character (6) :: bmark !used for file bacspacing in back substitution

!               nfunc(i,j)=(j*j-j)/2+i
!
!***     change the sign of the last appearance of each node
!!

!write(34,*) 'lnods'
!do ioo=1,40
! write(34,567) (lnods(ioo,joo),joo=1,9)
!enddo
        567 format (9i4)
        if (iincs.gt.1.or.iiter.gt.1) goto 455 !performed only once in the solution

        do 140 ipoin=1,npoin !npoin=tot. no. of nodes in the structure
                klast=0 !set counter for element number to zero
                do 130 ielem=1,nelem !enter loop over each element
                        do 120 inode=1,nnode !enter loop over each node within an element
                                if (lnods(ielem,inode).ne.ipoin) goto 120
                                klast=ielem
                                nlast=inode
```

```
                    120      continue
              130      continue
              if (klast.ne.0) lnods(klast,nlast)=-ipoin
! after searching through each element, the last appearance of a node in an element
! is recorded with a -ve sign in the LNODS array i.e the nodal composition of each elem.
       140      continue
!write(34,*) 'lnods after registering last appearance of each node'
!do ioo=1,40
! write(34,567) (lnods(ioo,joo),joo=1,9)
!enddo
!write(34,*) 'end lnods'

       455      continue

!
!***    start by initializing everything that matters to zero
!
       do 450 ibufa=1,mbufa
450 eqrhs(ibufa)=0.0
       do 150 istif=1,mstif
150 gstif(istif)=0.0 !stiffness array
       do 160 ifron=1,mfron
       gload(ifron)=0.0 !load vector
       vecrv(ifron)=0.0 !vector of solved variables
       nacva(ifron)=0 !active variable array
       do 160 ibufa=1,mbufa
160 equat(ifron,ibufa)=0.0 !reduced eqn. vector
!
!***    and prepare for disc reading and writing operations
!
!write(34,*) "kresl", kresl
!write(34,*) "nelem", nelem
!
!write(34,*) 'ifron =',ifron
!write(34,*) 'mfron =',mfron
       nbufa=0
       if (kresl.gt.1) nbufa=mbufa
       rewind 1
       rewind 2
       rewind 3
       rewind 4
       rewind 8
!
!***    enter main element assembly-reduction loop
!
       nfron=0 !set the current frontwidth to zero
       kelva=0 !set the counter over all eliminated variables in the structure to zero
!""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""!
do 320 ielem=1,nelem !enter the main loop over each element
       write(34,*) 'ielem= ',ielem
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!! ASSEMBLE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
              if (kresl.gt.1) goto 400
! *** KRESL=1 - indicates reformulation of the elem. stiffnesses accompanied
! ***           by full eqn. resolution
```

```
! *** KRESL=2 - the element stiffnesses are not to be modified & so only 1 eqn.
! ***                 resolution takes place
!
                    kevab=0 !set the counter over the variables per element to zero
                    read(1) estif !read the stiffness matrix of the element from disk file
!                   write(34,*) 'estif estif estif estif estif estif',ielem !mine
!           write(34,*) estif   !mine
!enter loops over the element nodes (at element level) and dof per node respectively
                    do 170 inode=1,nnode
                    do 170 idofn=1,ndofn
                                nposi=(inode-1)*ndofn+idofn !locate,on an elem level, the position of a
!                               particular dof of a particular node of the element under consideration
                                locno=lnods(ielem,inode) !determine the nodal no. of the elem. node under
                                                                                    ! consideration
                                if (locno.gt.0) locel(nposi)=(locno-1)*ndofn+idofn !compute the global
!  location (i.e. correspon. to whole structure) of the element dof under consideration
! and store in the local array LOCEL
                                if (locno.lt.0) locel(nposi)=(locno+1)*ndofn-idofn !if it is a last appearance
!(identified by a -ve value of LOCNO) place a -ve sign before the relevant entry in LOCEL
                    170 continue !loop to consider the next variable (dof)
!write(34,*) 'locel'
! write(34,568) (locel(joo),joo=1,18)
!           568 format (18i4)
!
!***        start by looking for existing destinations
!
                    do 210 ievab=1,nevab !enter loop over each variable of an element
                                nikno=iabs(locel(ievab)) !compute the global location of the variable
                                kexis=0 !set counter to be used in identification of existing
!                                                   destination locations to zero
!           write(34,*) 'deluge ifron',ifron,ievab
!           if (nfron.eq.0) goto 181 !mine
                                do 180 ifron=1,nfron !enter loop over each element in the front
!                               write(34,*) 'deluge'
                                        if (nikno.ne.nacva(ifron)) goto 180
                                        kevab=kevab+1
                                        kexis=1
                                        ndest(kevab)=ifron
                                180 continue
!                               181 continue  !mine
                                if (kexis.ne.0) goto 210
!
!***        we now seek new empty places for destination vector
!
                                do 190 ifron=1,mfron
                                        if (nacva(ifron).ne.0) goto 190
                                        nacva(ifron)=nikno
                                        kevab=kevab+1
                                        ndest(kevab)=ifron
                                        goto 200
                                190 continue
!
!***        the new places may demand an increase in current frontwidth
!
                                200 if (ndest(kevab).gt.nfron) nfron=ndest(kevab)
           210 continue !loop to locate the remaining element variables in the front
```

```
!
!write(34,*) 'ndest'
! write(34,568) (ndest(joo),joo=1,18)
!write(34,*) 'nacva'
! write(34,569) (nacva(joo),joo=1,80)
!           569 format (80i4)
!write(34,*) 'nfron =',nfron
!
!write to file for storage if there is no need for matrix reformulation
!
           write (8) locel,ndest,nacva,nfron
!
!          rewind(8) !mine for testing
!          read (8)  locel1,ndest1,nacva1,nfron1 !mine for testing
!          write(34,*) 'nacva1 tests' !mine for testing
!write (34,*) locel1,ndest1,nacva1,nfron1 !mine for testing
!
!
!local array LOCEL contains the global location of the element dof under consideration
!NDEST is the destination vector, local array, computed separately on the intro. of
! a new element
!NACVA is the active variable array
!nfron is the current frontwidth
! NOTE FOR THE NEXT LINE
! *** KRESL=1 - indicates reformulation of the elem. stiffnesses accompanied
! ***              by full eqn. resolution
! *** KRESL=2 - the element stiffnesses are not to be modified & so only 1 eqn.
! ***              resolution takes place
!
!write(34,*) 'kresl',kresl
           400 if (kresl.gt.1) read(8) locel,ndest,nacva,nfron
!***************** ASSEMBLY STAGE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!***       assemble element loads
!
           do 220 ievab=1,nevab !loop over each element variable
                    idest=ndest(ievab) !set IDEST  equal to the destination vector of the variable
!                             under consideration i.e. IDEST defines the variable position in NACVA
!                             which in turn gives the position in  the global stiffness and load vectors
                    gload(idest)=gload(idest)+eload(ielem,ievab) !insert the element loads in the
!                             correct position  in the global load vector as on pg 182 corresponding to
!                             nacva i.e. loads are now next to corresponding global variables
!
!***       assemble the element stiffnesses - but not in resolution
!
                    if (kresl.gt.1) goto 402 !if KRESL=2 avoid assemling the element stiffness
                    !                                                                  but assemble load vector
           do 222 jevab=1,ievab
                    jdest=ndest(jevab)
! store the element stiffnesses in the correct locations in the one dimensional
! global stiffness sub matrix (half the array (upper half) is stored see pg 179)
!
!                   ngash=nfunc(idest,jdest) !original
!                   ngish=nfunc(jdest,idest) !original
           ngash=(jdest*jdest-jdest)/2+idest
           ngish=(idest*idest-idest)/2+jdest
                    if (jdest.ge.idest) gstif(ngash)=gstif(ngash)+estif(ievab,jevab)
```

```
                    if (jdest.lt.idest) gstif(ngish)=gstif(ngish)+estif(ievab,jevab)
!
          222 continue
402 continue
220 continue
!
!write(34,*) 'gstif'
!write(34,*) gstif
!write(34,*) 'gload'
!write(34,*) gload


!!!!!!!!!!!!!! ELIMINATION STAGE """""""""""""""""""""""""""""
!***      re-examine each element node, to enquire which can be eliminated $$$$$
          do 310 ievab=1,nevab !enter loop over each element variable to check which ones
                                                                    !can be eliminated
!write(34,*) 'loop',ievab

!
                    nikno=-locel(ievab) !store the variable no. as NIKNO with a change of sign.
                    ! therefore for elimination we are only interested in a +ve value of NIKNO
                    !
                    if (nikno.le.0) goto 310 ! if NIKNO is not +ve loop to the next variable
!
!***      find positions of variables in sub matrix that are ready for elimination
!
          do 300 ifron=1,nfron
!if (nacva(ifron).ne.nikno) cycle  !my version
          if (nacva(ifron).ne.nikno) goto 300 !search until you locate var. targetted
                    !
          for elimination in nacva
                    nbufa=nbufa+1
!
!***      write equations to disk or to tape when memory buffer size(mbufa) is full
!see pg 195
                    if (nbufa.le.mbufa) goto 406
                    nbufa=1
                    if (kresl.gt.1) goto 408 !for no reformulation of stiffness matrix
                    write (2) equat,eqrhs,npivo,namev !!!!!!!!!
!
                    goto 406
                    408 write (4) eqrhs
                    read(2) equat,eqrhs,npivo,namev !original
!read(2) equat,dummy,idumm,namev !mine
          406 continue
!
!***      extract the coefficients of the new equation for elimination
!
          if (kresl.gt.1) goto 404 !for no reformulation of stiffness matrix i.e just manipulate
!                                                                   the corresponding right hand sides
          do 230 jfron=1,mfron !over whole front so that all gstif is scanned
!                   if (ifron.lt.jfron) nloca=nfunc(ifron,jfron) !find the loc. of the eliminated var.
!                   if (ifron.ge.jfron) nloca=nfunc(jfron,ifron) ! in the global stiffness submatrix
                    if (ifron.lt.jfron) nloca=(jfron*jfron-jfron)/2+ifron
                    if (ifron.ge.jfron) nloca=(ifron*ifron-ifron)/2+jfron
!
```

```
! extract the appropriate stiffness coef. & insert into the eqn. to be eliminated
! which will later be transferred to disk
                    equat(jfron,nbufa)=gstif(nloca)

!to make its space available for susequent use we replace the stiffness coef. by zero
          230 gstif(nloca)=0.0
          404 continue
!***      and extract the corresponding right hand sides
!
          eqrhs(nbufa)=gload(ifron)
          gload(ifron)=0.0
          kelva=kelva+1 !increase the counter of eliminated vars. by one
          namev(nbufa)=nikno
          npivo(nbufa)=ifron
!
!***      deal with pivot
!

          pivot=equat(ifron,nbufa) !no. on the diagonal of the stiffness matrix stored as pivot
          if (pivot.gt.0.0) goto 235
!write(34,*) 'hi3'

          write (6,900) nikno,pivot
          900 format (3x,"negative or zero pivot encountered for variable " &
                    ,i4," of value" ,e17.6)
stop
          235 continue
          equat(ifron,nbufa)=0.0 !for temporary convenience for eliminating free var.

!
!***      enquire whether present variable is free or prescribed
!
if (iffix(nikno).eq.0) goto 250
!
!***      deal with a prescribed deflection
!
          do 240 jfron=1,nfron
                    240 gload(jfron)=gload(jfron)-fixed(nikno)*equat(jfron,nbufa)
          goto 280
!
!***      eliminate a free variable - deal with the right hand side first
!
250 do 270 jfron=1,nfron
                    gload(jfron)=gload(jfron)-equat(jfron,nbufa)*eqrhs(nbufa)/pivot
!
!***      now deal with the coefficients in core
!

                    if (kresl.gt.1) goto 418 !for the 2nd & subsequent load cases avoid reducing
!                                                                                   the stiffess terms

                    if (equat(jfron,nbufa).eq.0.0) goto 270 !avoids time wasting on zero operations
!                   nloca=nfunc(0,jfron) !original
                    nloca=(jfron*jfron-jfron)/2
                    cureq=equat(jfron,nbufa)
                    do 260 lfron=1,jfron
```

```
                        ngash=lfron+nloca
                    260 gstif(ngash)=gstif(ngash)-cureq*equat(lfron,nbufa)/pivot
                    418 continue
            270 continue
!write(34,*) 'melamax1'

280 equat(ifron,nbufa)=pivot !the pivot term must now be replaced in EQUAT as the reduced
!                                                                    equation has not yet been written to
disk
!
!***        record the new vacant space, and reduce frontwidth if possible
!
            nacva(ifron)=0
!write(34,*) 'nacva',nacva

            goto 290
!
!***        complete the element loop in the forward elimination
!
!write(34,*) 'melamax'

            300 continue
!write(34,*) 'mela',nacva(nfron)
            290 if (nacva(nfron).ne.0) goto 310
!write(34,*) 'weird'
            nfron=nfron-1
            if (nfron.gt.0) goto 290

            310 continue !looping across all variables
!write(34,*) 'damage nbufa',nbufa
!write(34,*) (nacva(ioo), ioo=1,18)
320 continue !looping across all elements
!!!
!!
!
!here downwards are my addition
!                   if (kresl.gt.1) goto 4081 !for no reformulation of stiffness matrix
!                   write (2) equat,eqrhs,npivo,namev !!!!!!!!!
!write(34,*) 'equat,eqrhs,npivo,namev'
!write(34,*) equat
!write(34,*) eqrhs
!write(34,*) npivo
!write(34,*) namev
!
!                   goto 4061
!                   4081 write (4) eqrhs
!                   read(2) equat,eqrhs,npivo,namev !original
!read(2) equat,dummy,idumm,namev !mine
!4061 continue
!!!
!!
!
!here upwards are my addition
```

```
!
!mine start
! write(34,*) "assembly and elimination part ready 1"
!mine end
!
!""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""!

      if (kresl.eq.1) write(2) equat,eqrhs,npivo,namev !!!!!!!!

      backspace 2

!***  enter back substitution phase. loop backwards through variables
      do 340 ievla=1,kelva
!
!***  read a new block of equations - if needed
!
          if (nbufa.ne.0) goto 412

          backspace 2
          read(2) equat,eqrhs,npivo,namev !!!!!!
          backspace 2

          nbufa=mbufa
          if (kresl.eq.1) goto 412
          backspace 4
          read(4) eqrhs
          backspace 4
      412 continue
!
!***  prepare to back substitute from the current equation
!
          ifron=npivo(nbufa)
          nikno=namev(nbufa)
          pivot=equat(ifron,nbufa)
          if (iffix(nikno).ne.0) vecrv(ifron)=fixed(nikno)
          if (iffix(nikno).eq.0) equat(ifron,nbufa)=0.0

!
!***  back-substitute in the current equation
!
          do 330 jfron=1,mfron
              330 eqrhs(nbufa)=eqrhs(nbufa)-vecrv(jfron)*equat(jfron,nbufa)
!
!***  put the final values where they belong
!
          if (iffix(nikno).eq.0) vecrv(ifron)=eqrhs(nbufa)/pivot
          if (iffix(nikno).ne.0) fixed(nikno)=-eqrhs(nbufa)
          nbufa=nbufa-1
          asdis(nikno)=vecrv(ifron) !insert the var. displacement into the global
                                                        !displacement vector ASDIS

      340 continue

!***  add displacements to previous total values
!
      do 345 itotv=1,ntotv
```

```
        345 tdisp(itotv)=tdisp(itotv)+asdis(itotv)
!
!***    store reactions for printing later
!
kboun=1
        do 370 ipoin=1,npoin
                nloca=(ipoin-1)*ndofn
                do 350 idofn=1,ndofn
                        ngush=nloca+idofn
                        if (iffix(ngush).gt.0) goto 360
                350 continue
                goto 370
                360 do 510 idofn=1,ndofn
                        ngash=nloca+idofn
                        510 treac(kboun,idofn)=treac(kboun,idofn)+fixed(ngash) !calc. partial reaction
                kboun=kboun+1
        370 continue
!
!***    add reactions into the total load array
!
        do 700 ipoin=1,npoin
                do 710 ielem=1,nelem
                do 710 inode=1,nnode
                        nloca=iabs(lnods(ielem,inode))
                710 if (ipoin.eq.nloca) goto 720
!
                720 do 730 idofn=1,ndofn
                        ngash=(inode-1)*ndofn+idofn
                        mgash=(ipoin-1)*ndofn+idofn
                730 tload(ielem,ngash)=tload(ielem,ngash)+fixed(mgash) !total reactions
        700 continue
!
return


end
```

SUBROUTINE **FUNCTION**

```
function nfunc(i,j)
!
        integer :: nfunc(100,100)  !100 was put arbitrary by me
        integer, intent(in) :: i,j

        nfunc(i,j)=(j*j-j)/2+i
!
end function
```

SUBROUTINE **GAUSSQ**

```
Subroutine gaussq(ngaus,posgp,weigp)
!**********************************************************************
!
!***** This subroutine sets up the Gauss-Legendre integration constants
!
!**********************************************************************
!
implicit none
!
        integer :: ngaus,jgash,igash,kgaus
        double precision :: posgp(4),weigp(4)
!
        if (ngaus.gt.2) goto 4
2       posgp(1)=-0.577350269189626
        weigp(1)=1.0
        goto 6
4       posgp(1)=-0.774596669241483
        posgp(2)=0.0
        weigp(1)=0.555555555555556
        weigp(2)=0.888888888888889
6       kgaus=ngaus/2
        do 8 igash=1,kgaus
         jgash=ngaus+1-igash
         posgp(jgash)=-posgp(igash)
         weigp(jgash)=weigp(igash)
        8       continue
        return
        end
```

SUBROUTINE **INCREM**

```
Subroutine increm(eload,fixed,iincs,melem,mevab,miter, &
                  mtotv,mvfix,ndofn,nelem,nevab,noutp, &
                  nofix,ntotv,nvfix,presc,rload,tfact,tload,toler)
!**********************************************************************
!
!***** This subroutine increments the applied load
!
!**********************************************************************
!
implicit none
!
        integer :: iincs,miter,nloca,nvfix,ivfix,ntotv,itotv,nevab,ievab
        integer :: ngash,idofn,nelem,ielem,ndofn,melem,mevab,mvfix,mtotv
        double precision :: facto,toler
        integer :: iffix(mtotv),noutp(2),nofix(mvfix)
        double precision :: eload(melem,mevab),fixed(mtotv),presc(mvfix,ndofn),rload(melem,mevab)
        double precision :: tload(melem,mevab),tfact
!
                write(6,900) iincs ! load increment no. currently being applied
900     format(5x,"increment number ",i5)
!FACTO is applied load factor for this increment
!TOLER is convergence toler. factor
```

```
!MITER max. no. of iteration allowed for the load increment
!noutp(1) Par. controlling output of results after 1st iteration
!noutp(2) Par. controlling output of converged results
                    read(5,950) facto,toler,miter,noutp(1),noutp(2)
950                 format (2f10.5,3i5)
                    tfact=tfact+facto
                    write(6,960) tfact,toler,miter,noutp(1),noutp(2)
960                 format (5x,"load factor =",f10.5,5x,"convergence tolerance &
                    =",f10.5,5x,"max. no. of iterations =",i5,//"initial output parameter &
                    =",i5,5x,"final output parameter =",i5)
!
!Accumulate the incremental loading into array ELOAD for equation solution
!and also into TLOAD to record the total load applied to the structure
do 80 ielem=1,nelem
do 80 ievab=1,nevab
          eload(ielem,ievab)=eload(ielem,ievab)+rload(ielem,ievab)*facto
80        tload(ielem,ievab)=tload(ielem,ievab)+rload(ielem,ievab)*facto
!
! *** interpret fixity data in vector form
!
          do 100 itotv=1,ntotv !zero the global vector of prescribed displacements
100       fixed(itotv)=0.0
          do 110 ivfix=1,nvfix !nvfix = total no. of nodal points at which one
                                                                    !or more dof's are restrained
          nloca=(nofix(ivfix)-1)*ndofn
          do 110 idofn=1,ndofn
          ngash=nloca+idofn
          fixed(ngash)=presc(ivfix,idofn)*facto
!Insert any prescribed values, factored by the load increment factor,
!into the appropriate position in the global vector 'fixed(itotv)'
110       continue
return
end
```

SUBROUTINE **INPUT**

```
Subroutine input(coord,iffix,lnods,matno,melem,mevab,mfron,mmats, &
                                 mpoin,mtotv,mvfix,nalgo, &
                                 ncrit,ndfro,ndofn,nelem, &
                                 nevab,ngaus,ngau2, &
                                 nincs,nmats,nnode,nofix,npoin,nprop,nstre,nstr1, &
                                 ntotg,ntotv,ntype,nvfix,posgp,presc,props,weigp)
!**********************************************************************
!
!***** This subroutine accepts most of the input data
!
!**********************************************************************
!
implicit none
!
          character (12)      :: title
          integer                         ::          npoin,nelem,nvfix,ntype,nnode,nmats,ngaus, &
                                                      nalgo,ncrit,nincs,nstre,ifpre
```

```
      integer :: inode,mtotv,melem,mvfix,ndofn,mmats,nprop,mpoin,nevab
      integer :: nstr1,ntotv,ngau2,ntotg,ielem,numel,ipoin,idime,ivfix
      integer :: idofn,nloca,ifdof,ngash,imats,numat,iprop,mfron,mevab
!
      integer ::iffix(mtotv),lnods(melem,9),matno(melem),ndfro(melem),nofix(mvfix)
      double precision :: presc(mvfix,ndofn),props(mmats,nprop),weigp(4),posgp(4)
      double precision ::                    coord(mpoin,2)
!
      rewind 1
      rewind 2
      rewind 3
      rewind 4
      rewind 8
      read(5,920) title
      write(6,920) title
      920 format (12a6)
!
!*** read the first data card, and echo it immediately
!


      read(5,900) npoin,nelem,nvfix,ntype,nnode,nmats,ngaus, &
                                    nalgo,ncrit,nincs,nstre
      900 format (11i5)
      nevab=ndofn*nnode
      nstr1=nstre+1
      if (ntype.eq.3) nstr1=nstre
      ntotv=npoin*ndofn
      ngau2=ngaus*ngaus
      ntotg=nelem*ngau2
      write(6,901) npoin,nelem,nvfix,ntype,nnode,nmats,ngaus,nevab, &
            nalgo,ncrit,nincs,nstre
      901 format (//"npoin =",i4,4x,"nelem =",i4,4x,"nvfix =",i4,4x, &
                                    "ntype =",i4,4x,"nnode =",i4,// &
                                    "nmats =",i4,4x,"ngaus =",i4, &
                                    4x,"nevab =",i4,4x,"nalgo =",i4// &
                                    "ncrit =",i4,4x,"nincs =",i4,4x,"nstre =" ,i4)
      call check1(ndofn,nelem,ngaus,nmats,nnode,npoin,nstre,ntype,nvfix,ncrit,nalgo,nincs)
!
!*** read the element nodal connections, and the property numbers
!
      write (6,902)
      902 format (//"element",3x,"Mat. property",6x,"node numbers")
      do 2 ielem=1,nelem
            read (5,900) numel,matno(numel),(lnods(numel,inode),inode=1,nnode)
      2 write (6,903) numel,matno(numel),(lnods(numel,inode),inode=1,nnode)
      903 format (1x,i5,i9,10x,8i5)
!
!*** zero all the nodal coordinates, prior to reading some of them
!
      do 4 ipoin=1,npoin
      do 4 idime=1,2
      4 coord(ipoin,idime)=0.0
!
!*** read some nodal coprdinates one after each other,
!        finishing with the last node of all
```

```
!
          write (6,904)
          904 format (//"node",10x,"x",10x,"y")
          6 read (5,905) ipoin,(coord(ipoin,idime),idime=1,2)
          905 format (i5,6f10.5)
          if (ipoin.ne.npoin) goto 6
!
!*** interpolate coordinates of mid-side nodes
!
          call nodexy(coord,lnods,melem,mpoin,nelem,nnode)
          do 10 ipoin=1,npoin
                    10 write (6,906) ipoin,(coord(ipoin,idime),idime=1,2)
                    906 format (1x,i5,3f10.3)
!
!*** read the fixed values
!
          write (6,907)
          907 format (//"node",8x,"code",8x,"fixed values")
          do 8 ivfix=1,nvfix
                    read (5,9081) nofix(ivfix),ifpre,(presc(ivfix,idofn),idofn=1,ndofn)
!              read (5,908) nofix(ivfix),ifpre,(presc(ivfix,idofn),idofn=1,ndofn)
                    write (6,908) nofix(ivfix),ifpre,(presc(ivfix,idofn),idofn=1,ndofn)
                    nloca=(nofix(ivfix)-1)*ndofn
                    ifdof=10**(ndofn-1)
          do 8 idofn=1,ndofn
                    ngash=nloca+idofn
                    if (ifpre.lt.ifdof) goto 8
                    iffix(ngash)=1
                    ifpre=ifpre-ifdof
          8 ifdof=ifdof/10
          9081 format (i4,i5,5f10.6)
          908 format (1x,i4,5x,i5,5x,5f10.6)
!
!*** read the available selection of element properties
!
          16 write(6,910)
          910 format (//"number",4x,"element properties")
          do 18 imats=1,nmats
                    read(5,900) numat
                    read(5,930) (props(numat,iprop),iprop=1,nprop)
                    930 format (8f10.5)
          18 write (6,911) numat,(props(numat,iprop),iprop=1,nprop)
          911 format (1x,i4,3x,8e14.6)
!
!*** set up Gaussian integration constants
!
          call gaussq(ngaus,posgp,weigp)
          call check2(coord,iffix,lnods,matno,melem,mfron,mpoin,mtotv, &
                              mvfix,ndfro,ndofn,nelem,nmats,nnode,nofix,npoin,nvfix)
return
end
```

SUBROUTINE **INVAR**

```
Subroutine invar(devia,lprop,mmats,ncrit,props,sint3,steff,stemp, &
                                theta,varj2,yield,ntype)
!
!*********************************************************************
!
!***** This subroutine evaluates the stress invariants and the current
!***** value of the yield function (see pg 239 Owen & Hinton for more details)
!
!*********************************************************************
!
implicit none
!
        integer :: mmats,ncrit,lprop,ntype
        double precision :: root3,smean,varj2,varj3,steff,sint3,theta,yield,phira,snphi,sphi
        double precision :: devia(4),props(mmats,7),stemp(4)
!
        root3=1.73205080757
!       compute the deviatoric stresses
if (ntype.eq.1) stemp(4)=0.0  !this line is my addition (corrected bug for plane stress)
!write(60,*) 'STEMP-Z1',stemp(4)
        smean=(stemp(1)+stemp(2)+stemp(4))/3.0 !smean=(sigmaX+sigmaY+sigmaZ)/3
        devia(1)=stemp(1)-smean !sigmaX-smean
        devia(2)=stemp(2)-smean !sigmaY-smean
        devia(3)=stemp(3)                       !tauXY
        devia(4)=stemp(4)-smean !sigmaZ-smean
!write(60,*) 'NTYPE now',ntype
!write(60,*) 'STEMP-Z2',stemp(4)

!if (ntype.eq.1) then !for plane stress  THese wrong LINEs IS MY ADDITION
! smean=(stemp(1)+stemp(2))/2.0 !smean=(sigmaX+sigmaY+sigmaZ)/3
!       devia(1)=stemp(1)-smean !sigmaX-smean
!       devia(2)=stemp(2)-smean !sigmaY-smean
!       devia(3)=stemp(3)                       !tauXY
!       devia(4)=0.0 !sigmaZ-smean
!endif
        !here upwards is my addition
!
!
!       calculate the second deviatoric stress invariant J2'
! see added note in Mendelson pg41 for a proof of these eqns.
        varj2=devia(3)*devia(3)+0.5*(devia(1)*devia(1)+devia(2)*devia(2) &
                +devia(4)*devia(4))
!       calculate the third deviatoric stress invariant J3'
        varj3=devia(4)*(devia(4)*devia(4)-varj2)
!       compute sqrt(J2')
        steff=sqrt(varj2)
! Evaluate SIN3theta according to (7.61)
        if (steff.eq.0.0) goto 10
        sint3=-3.0*root3*varj3/(2.0*varj2*steff)
        if (sint3.gt.1.0) sint3=1.0
        goto 20
10  sint3=0.0
20  continue
        if (sint3.lt.-1.0) sint3=-1.0
        if (sint3.gt.1.0) sint3=1.0
```

```
          theta=asin(sint3)/3.0 !compute theta
!
          goto (1,2,3,4) ncrit !branch according to the yield criterion being employed
          ! in order to calculate the current value of the yield function
!*** tresca
1 yield=2.0*cos(theta)*steff
return
!*** von mises
2         yield=root3*steff
return
!*** mohr-coulomb
3         phira=props(lprop,7)*0.017453292
          snphi=sin(phira)
          yield=smean*snphi+steff*(cos(theta)-sin(theta)*snphi/root3)
          return
!*** drucker-prager
4         phira=props(lprop,7)*0.017453292
          snphi=sin(phira)
          yield=6.0*smean*snphi/(root3*(3.0-snphi))+steff
return
end
```

SUBROUTINE **JACOB2**

```
Subroutine jacob2(cartd,deriv,djacb,elcod,gpcod,ielem,kgasp,nnode,shap)
!**********************************************************************
!
!***** This subroutine evaluates the Jacobian matrix and the cartesian
!         shape function derivatives

!
!**********************************************************************
!
implicit none
!
          integer :: idime,kgasp,inode,nnode,jdime,ielem
          double precision :: djacb
          double precision :: cartd(2,9),deriv(2,9),elcod(2,9),gpcod(2,9),shap(9), &
                                                    xjaci(2,2),xjacm(2,2)

!
!***       calculate coordinates of sampling point
!
          do 2 idime=1,2
          gpcod(idime,kgasp)=0.0
          do 2 inode=1,nnode
          gpcod(idime,kgasp)=gpcod(idime,kgasp)+elcod(idime,inode)*shap(inode)
2         continue
!
! *** create Jacobian matrix xjacm
!
          do 4 idime=1,2
          do 4 jdime=1,2
```

```
            xjacm(idime,jdime)=0.0
            do 4 inode=1,nnode
            xjacm(idime,jdime)=xjacm(idime,jdime)+deriv(idime,inode)*elcod(jdime,inode)
            4 continue
!write(34,*) "xjacm(1,1) =", xjacm(1,1)
!write(34,*) "xjacm(2,2) =", xjacm(2,2)
!write(34,*) "xjacm(1,2) =", xjacm(1,2)
!write(34,*) "xjacm(2,1) =", xjacm(2,1)

!
! *** calculate determinant and inverse of Jacobian matrix
!
            djacb=xjacm(1,1)*xjacm(2,2)-xjacm(1,2)*xjacm(2,1)
            if (djacb) 6,6,8
6           write(6,600) ielem
!write(34,*) 'djacb= ',djacb
            stop
8           continue
!write(34,*) "detj =", djacb
            xjaci(1,1)=xjacm(2,2)/djacb
            xjaci(2,2)=xjacm(1,1)/djacb
            xjaci(1,2)=-xjacm(1,2)/djacb
            xjaci(2,1)=-xjacm(2,1)/djacb !!**********************
!write(34,*) "xjaci(1,1) =", xjaci(1,1)
!write(34,*) "xjaci(2,2) =", xjaci(2,2)
!write(34,*) "xjaci(1,2) =", xjaci(1,2)
!write(34,*) "xjaci(2,1) =", xjaci(2,1)
!
! *** calculate cartesian derivatives
!
            do 10 idime=1,2
            do 10 inode=1,nnode
            cartd(idime,inode)=0.0
            do 10 jdime=1,2
            cartd(idime,inode)=cartd(idime,inode)+xjaci(idime,jdime)* &
                                    deriv(jdime,inode)
10          continue
!WRITE(34,*) "cartd11 =",cartd(1,1)
!WRITE(34,*) "cartd12 =",cartd(1,2)
!WRITE(34,*) "cartd13 =",cartd(1,3)
!WRITE(34,*) "cartd14 =",cartd(1,4)
!WRITE(34,*) "cartd15 =",cartd(1,5)
!WRITE(34,*) "cartd16 =",cartd(1,6)
!WRITE(34,*) "cartd17 =",cartd(1,7)
!WRITE(34,*) "cartd18 =",cartd(1,8)
!WRITE(34,*) "cartd21 =",cartd(2,1)
!WRITE(34,*) "cartd22 =",cartd(2,2)
!WRITE(34,*) "cartd23 =",cartd(2,3)
!WRITE(34,*) "cartd24 =",cartd(2,4)
!WRITE(34,*) "cartd25 =",cartd(2,5)
!WRITE(34,*) "cartd26 =",cartd(2,6)
!WRITE(34,*) "cartd27 =",cartd(2,7)
!WRITE(34,*) "cartd28 =",cartd(2,8)

600         format (//,"program halted in subroutine jacob2",/,11x, &
                    "zero or negative area",/,10x,"element number",i5)
```

```
        return
        end
```

SUBROUTINE **LINEAR**

```
Subroutine linear(cartd,dmatx,eldis,lprop,mmats,ndofn,nnode,nstre, &
                                ntype,props,stran,stres,kgasp,gpcod,shap)

!
!**********************************************************************
!
!***** This subroutine evaluates stresses and strains assuming linear
! **** elastic behaviour
!
!**********************************************************************
!
implicit none
!
        integer :: mmats,lprop,idofn,ndofn,jdofn,inode,nnode,kgasp,istre,nstre,jstre
        integer :: ntype
        double precision :: poiss,bgash
        double precision :: agash(2,2),cartd(2,9),dmatx(4,4),eldis(2,9),props(mmats,7),stran(4)
        double precision :: stres(4),gpcod(2,9),shap(9)

!write(60,*) 'ntype=', ntype

poiss=props(lprop,2) !Poisson's ratio of the material
!
! Calculate the cartesian derivatives of the Gauss point displacement components
do 20 idofn=1,ndofn
do 20 jdofn=1,ndofn
bgash=0.0
do 10 inode=1,nnode
10 bgash=bgash+cartd(jdofn,inode)*eldis(idofn,inode)
20 agash(idofn,jdofn)=bgash
!
!*** calculate the strains
!
stran(1)=agash(1,1)
stran(2)=agash(2,2)
stran(3)=agash(1,2)+agash(2,1)
stran(4)=0.0
do 30 inode=1,nnode
30 stran(4)=stran(4)+eldis(1,inode)*shap(inode)/gpcod(1,kgasp)
!
!*** and the corresponding stresses (sigma=D x epsilon)
!
do 40 istre=1,nstre
stres(istre)=0.0
do 40 jstre=1,nstre
40 stres(istre)=stres(istre)+dmatx(istre,jstre)*stran(jstre)
if (ntype.eq.1) stres(4)=0.0 !for plane stress
!
```

```
!write(60,*) 'stresses x y xy z'
!write(60,*) stres(1)
!write(60,*) stres(2)
!write(60,*) stres(3)
!write(60,*) stres(4)
!
if (ntype.eq.2) stres(4)=poiss*(stres(1)+stres(2))
return
end
```

SUBROUTINE **LOADPS**

```
Subroutine loadps(coord,lnods,matno,melem,mmats,mpoin,nelem, &
                              nevab,ngaus,nnode,npoin,nstre,ntype,posgp, &
                              props,rload,weigp,ndofn)
!**********************************************************************
!
!***** This subroutine evaluates the consistent nodal forces for each element
! see chapter 7 of owen and hinton 'finite element programming' for good explanation
!
!**********************************************************************
!
implicit none
!
        character (12)      :: title
!
        integer :: melem,mpoin,mmats,ielem,nelem,ievab,nevab,iplod,igrav,iedge
        integer :: lodpt,idofn,inode,nnode,nloca,ngash,npoin,lprop,lnode,idime
        integer :: kgasp,igaus,ngaus,jgaus,ntype,mgash,nedge,nodeg,ncode
        integer :: neass,iodeg,jnode,kount,knode,ndofn,nstre
        double precision :: twopi,theta,gravy,thick,dense,gxcom,gycom,exisp,etasp,djacb,dvolu
        double precision :: pxcom,pycom,radus
        integer :: lnods(melem,9),matno(melem),noprs(4)
        double precision :: cartd(2,9),coord(mpoin,2),deriv(2,9),dgash(2),dmatx(4,4)
        double precision :: elcod(2,9),pgash(2),point(2),posgp(4),press(4,2)
        double precision :: props(mmats,7),rload(melem,18),shap(9),stran(4)
        double precision :: stres(4),weigp(4),gpcod(2,9)

!
        twopi=6.283185308
        do 10 ielem=1,nelem
        do 10 ievab=1,nevab
                10      rload(ielem,ievab)=0.0
        read(5,901) title
        901     format(12a6)
                write(6,903) title
        903     format(12a6)
!
!*** read data controlling loading types to be inputted
!Non zero values of these respective items indicate that point loads,
!gravity loading or distributes edge loadings are to be considered
!
!the consistent nodal loads are evaluated for each element separately
```

```
!and stored in the array RLOAD(IELEM,IEVAB) where IELEM indcates the element
! and IEVAB ranges over the degrees of freedom of the element
!
          read(5,919) iplod,igrav,iedge
          write(6,919) iplod,igrav,iedge
          919       format(3i5)
!
!*** read nodal point loads
!
          if (iplod.eq.0) goto 500
          20        read(5,931) lodpt,(point(idofn),idofn=1,2)
                    write(6,931) lodpt,(point(idofn),idofn=1,2)
                    931       format(i5,2f10.3)
!
!*** associate the nodal point loads with an element
!
          do 30 ielem=1,nelem
          do 30 inode=1,nnode
                    nloca=iabs(lnods(ielem,inode))
          30        if(lodpt.eq.nloca) goto 40
40        do 50 idofn=1,2
                    ngash=(inode-1)*2+idofn
          50        rload(ielem,ngash)=point(idofn)
          if (lodpt.lt.npoin) goto 20
500       continue
          if (igrav.eq.0) goto 600
!
!*** gravity load section
!
!***      read gravity angle and gravitational constant
!
          read(5,906) theta,gravy
          906       format(2f10.3)
          write(6,911) theta,gravy
          911       format(1x,"gravity angle =", f10.3,"gravity constant=",f10.3)
          theta=theta/57.295779514
!
!***      loop over each element
!
          do 90 ielem=1,nelem
!
!***      set up preliminary constants
!
                    lprop=matno(ielem)
                    thick=props(lprop,3)
                    dense=props(lprop,4)
                    if (dense.eq.0.0) goto 90
                    gxcom=dense*gravy*sin(theta)
                    gycom=dense*gravy*cos(theta)
!
!***      compute coordinates of the element nodal points
!
                    do 60 inode=1,nnode
                              lnode=iabs(lnods(ielem,inode))
                    do 60 idime=1,2
                    60        elcod(idime,inode)=coord(lnode,idime)
```

```
!
!***    enter loops for area numerical integration
!
                kgasp=0
                do 80 igaus=1,ngaus
                do 80 jgaus=1,ngaus
                        exisp=posgp(igaus)
                        etasp=posgp(jgaus)
!
!***    compute the shape functions at the sampling points and elemental volume
!
                        call sfr2(deriv,etasp,exisp,nnode,shap)
                        kgasp=kgasp+1
                        call jacob2(cartd,deriv,djacb,elcod,gpcod,ielem,kgasp,nnode,shap)
                        dvolu=djacb*weigp(igaus)*weigp(jgaus)
                        if (thick.ne.0.0) dvolu=dvolu*thick
                        if (ntype.eq.3) dvolu=dvolu*twopi*gpcod(1,kgasp)
!
!***    calculate loads and associate with element nodal loads
!
                        do 70 inode=1,nnode
                                ngash=(inode-1)*2+1
                                mgash=(inode-1)*2+2
                                rload(ielem,ngash)=rload(ielem,ngash)+gxcom*shap(inode)*dvolu
                        70 rload(ielem,mgash)=rload(ielem,mgash)+gycom*shap(inode)*dvolu
                80      continue
        90      continue
600     continue
        if (iedge.eq.0) goto 700
!
!***    distributed edge loads section
!
        read(5,932) nedge
        932     format (i5)
        write(6,912) nedge
        912     format(5x,"no. of loaded edges =",i5)
        write(6,915)
        915     format (5x,"list of loaded edges and applied loads")
        nodeg=3 !calculate the no. of nodes along an element edge
        ncode=nnode
        if (nnode.eq.4) nodeg=2
        if (nnode.eq.9) ncode=8
!
!***    loop over each loaded edge
!
        do 160 iedge=1,nedge
!
!***    read data locating the loaded edge and applied load
!
        read(5,902) neass,(noprs(iodeg),iodeg=1,nodeg)
902     format (4i5)
        write (6,913) neass,(noprs(iodeg),iodeg=1,nodeg)
913     format (i10,5x,3i5)
        read (5,914) ((press(iodeg,idofn),idofn=1,2),iodeg=1,nodeg)
        write (6,914) ((press(iodeg,idofn),idofn=1,2),iodeg=1,nodeg)
914     format (6f10.3)
```

```
      etasp=-1.0 !set eta to -1 in order to adjust the shape functions
      ! to make them resemble that of the beam so that we integrate along
      !the prescribed edge. see owen & hinton page 156
!
!***  calculate the coordinates of the nodes of the element edge
!
      do 100 iodeg=1,nodeg
      lnode=noprs(iodeg)
      do 100 idime=1,2
      100   elcod(idime,iodeg)=coord(lnode,idime) !ELCOD is a local array
!
!***  enter loop for linear numerical integration along the loaded edges
      do 150 igaus=1,ngaus
      exisp=posgp(igaus) !set up the coordinate XIn at the Gaussian sampling points
      !The order of integration rule is defined by NGAUS
      !the sampling point positions & weighting factors are stored respectively
      !in arrays POSGP and WEIGP
!
!***  evaluate the shape functions at the sampling points corresponding
! to XIn ad eta=-1
!
      call sfr2(deriv,etasp,exisp,nnode,shap)
!
!***  calculate the components of the equivalent nodal loads
!
      do 110 idofn=1,2 !enter loops over no. of dof per node
      pgash(idofn)=0.0 !zero two local arrays
      dgash(idofn)=0.0
      do 110 iodeg=1,nodeg !enter loops over no. of nodes per element face
            pgash(idofn)=pgash(idofn)+press(iodeg,idofn)*shap(iodeg)
            110   dgash(idofn)=dgash(idofn)+elcod(idofn,iodeg)*deriv(1,iodeg)
      dvolu=weigp(igaus) !gaussian weighting function
      pxcom=dgash(1)*pgash(2)-dgash(2)*pgash(1)
      pycom=dgash(1)*pgash(1)+dgash(2)*pgash(2)
      if (ntype.ne.3) goto 115
      radus=0.0
      do 125 iodeg=1,nodeg
125   radus=radus+shap(iodeg)*elcod(1,iodeg)
      dvolu=dvolu*twopi*radus
115   continue
!
!***  associate the equivalent nodal edge loads with an element
!
      do 120 inode=1,nnode
            nloca=iabs(lnods(neass,inode))
      120   if (nloca.eq.noprs(1)) goto 130
      130   jnode=inode+nodeg-1
      kount=0
      do 140 knode=inode,jnode
            kount=kount+1
            ngash=(knode-1)*ndofn+1
            mgash=(knode-1)*ndofn+2
            if (knode.gt.ncode) ngash=1
            if (knode.gt.ncode) mgash=2
            rload(neass,ngash)=rload(neass,ngash)+shap(kount)*pxcom*dvolu
      140   rload(neass,mgash)=rload(neass,mgash)+shap(kount)*pycom*dvolu
```

```
150     continue
160     continue
700     continue
        write (6,907)
907     format (5x,"total nodal forces for each element")
        do 290 ielem=1,nelem
        290     write(6,905) ielem,(rload(ielem,ievab),ievab=1,nevab)
905     format (1x,i4,5x,8e12.4/(10x,8e12.4))

!write(34,*) 'heqq'
        return
        end
```

SUBROUTINE **MODPS**

```
Subroutine modps(dmatx,lprop,mmats,ntype,props)
!**********************************************************************
!
!***** This subroutine evaluates the D matrix
!
!**********************************************************************
!
implicit none
!
        integer :: mmats,istr1,jstr1,lprop,ntype
        double precision :: young,poiss,const,conss
        double precision :: dmatx(4,4),props(mmats,7)
!
        young=props(lprop,1)
        poiss=props(lprop,2)
        do 10 istr1=1,4
        do 10 jstr1=1,4
        10      dmatx(istr1,jstr1)=0.0
        if (ntype.ne.1) goto 4
!
!***     D matrix for plane stress case
!
        const=young/(1.0-poiss*poiss)
        dmatx(1,1)=const
        dmatx(2,2)=const
        dmatx(1,2)=const*poiss
        dmatx(2,1)=const*poiss
        dmatx(3,3)=(1.0-poiss)*const/2.0
!write(34,*) "dmatx(1,1) =", dmatx(1,1)
!write(34,*) "dmatx(2,2) =", dmatx(2,2)
!write(34,*) "dmatx(1,2) =", dmatx(1,2)
!write(34,*) "dmatx(2,1) =", dmatx(2,1)
!write(34,*) "dmatx(3,3) =", dmatx(3,3)
return
4       if (ntype.ne.2) goto 6
!
!***     D matrix for plane strain case
!
```

```
            const=young*(1.0-poiss)/((1.0+poiss)*(1.0-2.0*poiss))
            dmatx(1,1)=const
            dmatx(2,2)=const
            dmatx(1,2)=const*poiss/(1.0-poiss)
            dmatx(2,1)=const*poiss/(1.0-poiss)
            dmatx(3,3)=(1.0-2.0*poiss)*const/(2.0*(1.0-poiss))
return
6           if (ntype.ne.3) goto 8
!
!***        D matrix for axisymmetric case
!
            const=young*(1.0-poiss)/((1.0+poiss)*(1.0-2.0*poiss))
            conss=poiss/(1.0-poiss)
            dmatx(1,1)=const
            dmatx(2,2)=const
            dmatx(3,3)=const*(1.0-2.0*poiss)/(2.0*(1.0-poiss))
            dmatx(1,2)=const*conss
            dmatx(1,4)=const*conss
            dmatx(2,1)=const*conss
            dmatx(2,4)=const*conss
            dmatx(4,1)=const*conss
            dmatx(4,2)=const*conss
            dmatx(4,4)=const
!
8 continue
return
end
```

SUBROUTINE **NODEXY**

```
Subroutine nodexy(coord,lnods,melem,mpoin,nelem,nnode)
!*********************************************************************
!
!***** This subroutine interpolates the mid side nodes of straight
!     sides of elements and the midside node of 9 noded elements
!
!*********************************************************************
!
implicit none
!
            integer :: melem,mpoin,ielem,nelem,nnode,nnod1,inode,nodst,igash,nodfn,midpt
            integer :: nodmd,kount,lnode,lnod1,lnod3,lnod5,lnod7
            double precision :: total
            integer :: lnods(melem,9)
            double precision :: coord(mpoin,2)
            if (nnode.eq.4) return !for 4-noded elements there are no midside nodes
!
!*** loop over each element
!
            do 30 ielem=1,nelem
!
!*** loop over each element edge
!
```

```
                    nnod1=9
                    if (nnode.eq.8) nnod1=7
                    do 20 inode=1,nnod1,2
                              if (inode.eq.9) goto 50
!
!*** compute the node number of the 1st node
!
                              nodst=lnods(ielem,inode)
                              igash=inode+2
                              if (igash.gt.8) igash=1
!
!*** compute the node number of the last node
!
                              nodfn=lnods(ielem,igash)
                              midpt=inode+1
!
!*** compute the node number of the intermediate node
!
                              nodmd=lnods(ielem,midpt)
                              total=abs(coord(nodmd,1))+abs(coord(nodmd,2))
!
!*** if the coordinates of the intermediate node are both zero
!          interpolate by a straight line to calc. its coords
!
                              if (total.gt.0.0) goto 20
                              kount=1
                              10        coord(nodmd,kount)=(coord(nodst,kount)+coord(nodfn,kount))/2.0
                              kount=kount+1
                              if (kount.eq.2) goto 10
                    20        continue
                    goto 30
                    50        lnode=lnods(ielem,inode) !for 9 node Lagrangian element
                    total=abs(coord(lnode,1))+abs(coord(lnode,2))
                    if (total.gt.0.0) goto 30
                    lnod1=lnods(ielem,1)
                    lnod3=lnods(ielem,3)
                    lnod5=lnods(ielem,5)
                    lnod7=lnods(ielem,7)
                    kount=1
                    40        coord(lnode,kount)=(coord(lnod1,kount)+coord(lnod3,kount) &
                              +coord(lnod5,kount)+coord(lnod7,kount))/4.0
                    kount=kount+1
                    if (kount.eq.2) goto 40
          30        continue
return
end
```

SUBROUTINE **OUTPUT**

```
Subroutine output (iiter,mtotg,mtotv,mvfix,nelem,ngaus,nofix, &
                              noutp,npoin,nvfix,strsg,tdisp,treac,epstn, &
                              ntype,nchek)
!*****************************************************************
```

```
!
!***** This subroutine outputs displacements, reactions and stresses
! **** evaluates the equivalent nodal forces
!
!**********************************************************************
!
implicit none
!
        integer :: mvfix,mtotv,mtotg,koutp,iiter,nchek,ntype,ipoin,npoin,ngash
        integer :: ngish,igash,ivfix,nvfix,idofn,kgaus,ielem,nelem,kelgs,igaus
        integer :: ngaus,jgaus,istr1,istre
        double precision :: xgash,xgish,xgesh,xgosh
        integer :: nofix(mvfix),noutp(2)
        double precision :: tdisp(mtotv),treac(mvfix,2),epstn(mtotg),strsg(4,mtotg),strsp(3)

!
        koutp=noutp(1)
        if (iiter.gt.1) koutp=noutp(2)
        if (iiter.eq.1.and.nchek.eq.0) koutp=noutp(2)
!
!*** output displacements
!
        if (koutp.lt.1) goto 10
        write(6,900)
900 format (5x,"displacements")
        if (ntype.ne.3) write(6,950)
950 format (6x,"node",6x,"x-disp.",7x,"y-disp.")
        if (ntype.eq.3) write(6,955)
955 format(6x,"node",6x,"r-disp.",7x,"z-disp.")
        do 20 ipoin=1,npoin
        ngash=ipoin*2
ngish=ngash-2+1
20 write(6,910) ipoin,(tdisp(igash),igash=ngish,ngash)
910 format (i10,3e14.6)
!
10 continue
!
!*** output reactions
!
if (koutp.lt.2) goto 30
        write(6,920)
920 format(5x,"reactions")
        if (ntype.ne.3) write(6,960)
960 format (6x,"node",6x,"x-reac.",7x,"y-reac.")
        if (ntype.eq.3) write(6,965)
965 format (6x,"node",6x,"r-reac.",7x,"z-reac.")
        do 40 ivfix=1,nvfix
40 write (6,910) nofix(ivfix),(treac(ivfix,idofn), idofn=1,2)
30 continue
!
!*** output stresses
!
if (koutp.lt.3) goto 50
if (ntype.ne.3) write (6,970)
970 format (1x,"g.p.",6x,"xx-stress",5x,"yy-stress",5x,"xy-stress", &
                5x,"zz-stress",6x,"max p.s.",6x,"min p.s.",3x,"angle",3x,"e.p.s.")
```

```
        if (ntype.eq.3) write(6,975)
975 format (1x,"g.p.",6x,"rr-stress",5x,"zz-stress",5x,"rz-stress",5x &
                "tt-stress",6x,"max p.s.",6x,"min p.s.",3x,"angle",3x,"e.p.s.")
        kgaus=0
        do 60 ielem=1,nelem !loop over each element
                kelgs=0
                write(6,930) ielem
                930 format (1h0,5x,13helement no. =,i5)
!evaluate the principal stresses & direction for each Gauss point according to 7.97
        do 60 igaus=1,ngaus
        do 60 jgaus=1,ngaus
                kgaus=kgaus+1
                kelgs=kelgs+1
                xgash=(strsg(1,kgaus)+strsg(2,kgaus))*0.5
                xgish=(strsg(1,kgaus)-strsg(2,kgaus))*0.5
                xgesh=strsg(3,kgaus)
                xgosh=sqrt(xgish*xgish+xgesh*xgesh)
                strsp(1)=xgash+xgosh  !principal stress 1
                strsp(2)=xgash-xgosh        !principal stress 2
                if (xgish.eq.0.0) xgish=0.1e-20
                strsp(3)=atan(xgesh/xgish)*28.647889757    !principal stress 3
! Output the cartesian stress components, the principal stresses & direction & the total
! effective plastic strain for each Gauss point. This latter quantity gives an
! immediate indication whether the Gauss point has yielded or not, since it will be zero
! for all elastic points
        60 write (6,940) kelgs,(strsg(istr1,kgaus),istr1=1,4), &
                (strsp(istre),istre=1,3),epstn(kgaus)
        940 format (i5,2x,6e14.6,f8.3,e14.6)
50 continue
!

return
end
```

PROGRAM **PLAST**

```
program plast
!**********************************************************************
!
!***** Program for the elasto-plastic analysis of plane stress,
! **** plane strain and axisymmetric solids
!
!**********************************************************************
!
implicit none
!
double precision :: asdis(6800),coord(3400,2),eload(1000,18),estif(18,18)
double precision :: eqrhs(10),equat(110,10),fixed(6800),gload(110),gstif(3240)
integer :: iffix(6800),lnods(1000,9),locel(18),matno(1000)
integer :: nacva(110),namev(10),ndest(18),ndfro(1000),nofix(500)
integer :: noutp(2),npivo(10)
double precision :: posgp(4),presc(500,2),props(5,7),rload(1000,18)
double precision :: stfor(6800),treac(500,2),vecrv(110),weigp(4)
```

```
double precision :: strsg(4,9000),tdisp(6800),tload(1000,18),tofor(6800), &
                                                    epstn(9000),effst(9000)
integer :: iincs,nvfix,ntype,ntotv,ntotg,nstr1,npoin,nnode,nmats,nincs,ngau2
integer :: ngaus,nevab,nelem,ncrit,nalgo,nstre,nprop,ndofn,mvfix,mtotv,mtotg
integer :: mstif,mpoin,mmats,mfron,mevab,melem,mbufa,nchek,lprop,kresl,iiter,miter
double precision :: tfact,pvalu,facto,toler,plwork,elwork


!! *** Original array dimensions
!           dimension asdis(300),coord(150,2),eload(40,18),estif(18,18), &
!                     eqrhs(10),equat(80,10),fixed(300),gload(80),gstif(3240), &
!                     iffix(300),lnods(40,9),locel(18),matno(40), &
!                     nacva(8),namev(10),ndest(18),ndfro(40),nofix(30), &
!                     noutp(2),npivo(10), &
!                     posgp(4),presc(30,2),props(5,7),rload(40,18), &
!                     stfor(300),treac(30,2),vecrv(80),weigp(4), &
!                     strsg(4,360),tdisp(300),tload(40,18), &
!                     tofor(300),epstn(360),effst(360)
!
!  *** Array dimensions changed to the following to increase model size permitted
!   *  * change also values of mvfix, melem, mpoin in subroutine 'dimen'
!    * *
!double precision :: asdis(6800),coord(3400,2),eload(1000,18),estif(18,18)
!double precision :: eqrhs(10),equat(80,10),fixed(6800),gload(80),gstif(3240)
!integer :: iffix(6800),lnods(1000,9),locel(18),matno(1000)
!integer :: nacva(80),namev(10),ndest(18),ndfro(1000),nofix(500)
!integer :: noutp(2),npivo(10)
!double precision :: posgp(4),presc(500,2),props(5,7),rload(1000,18)
!double precision :: stfor(6800),treac(500,2),vecrv(80),weigp(4)
!double precision :: strsg(4,9000),tdisp(6800),tload(1000,18),tofor(6800), &
!                                                    epstn(9000),effst(9000)
!
!*** open required disc files
!
           open (unit=5,file="inpdata.txt", status="old")
           open (unit=6,file="outdata.txt", status="old")
           open (unit=1,file="scratch.txt", status="old",form="unformatted")
           open (unit=2,file="unknown2.txt", status="old",form="unformatted")
           open (unit=3,file="unknown3.txt", status="old",form="unformatted")
           open (unit=4,file="unknown4.txt", status="old",form="unformatted")
           open (unit=8,file="unknown8.txt", status="old",form="unformatted")
           open (unit=34,file="unknown34.txt", status="old") !this is a text file for testing
           open (unit=60,file="elplwork.txt", status="old") !Work output file

!
!*** preset variables associated with dynamic dimensioning
!
           call dimen(mbufa,melem,mevab,mfron,mmats,mpoin,mstif,mtotg,mtotv,mvfix, &
                     ndofn,nprop,nstre)
!
!*** call the subroutine which reads most of the problem data
!
           call input (coord,iffix,lnods,matno,melem,mevab,mfron,mmats, &
                              mpoin,mtotv,mvfix,nalgo, &
                              ncrit,ndfro,ndofn,nelem,nevab,ngaus,ngau2, &
                              nincs,nmats,nnode,nofix,npoin,nprop,nstre, &
```

```
                              nstr1,ntotg,ntotv, &
                              ntype,nvfix,posgp,presc,props,weigp)
!
!*** call the subroutine which computes the consistent load vectors
!           for each element after reading the relevant input data
!
            call loadps(coord,lnods,matno,melem,mmats,mpoin,nelem, &
                  nevab,ngaus,nnode,npoin,nstre,ntype,posgp, &
                  props,rload,weigp,ndofn)
!
!*** initialise certain arrays
!
            call zero(eload,melem,mevab,mpoin,mtotg,mtotv,ndofn,nelem, &
                  nevab,ngaus,nstr1,ntotg,epstn,effst, &
                  ntotv,nvfix,strsg,tdisp,tfact, &
                  tload,treac,mvfix)
!
!*** loop over each increment
!
            do 100 iincs=1,nincs
!
!read data for current increment
!
                  call increm(eload,fixed,iincs,melem,mevab,miter,mtotv, &
                        mvfix,ndofn,nelem,nevab,noutp,nofix,ntotv, &
                        nvfix,presc,rload,tfact,tload,toler)
!
!*** loop over each iteration
!
!We now try to reach convergence for the present load increment
                  do 50 iiter=1,miter !'miter' is the maximum no. of iterations
!
!*** call routine which selects solution algorithm variable 'kresl'
!
                              call algor (fixed,iincs,iiter,kresl,mtotv,nalgo,ntotv)
!
!*** check whether a new evaluation of the stiffness matrix is required
!           otherwise use the previous one
!
                              if (kresl.eq.1) call stiffp(coord,epstn,iincs,lnods,matno, &
                              mevab,mmats,mpoin,mtotv,nelem,nevab,ngaus,nnode, &
                              nstre,nstr1,posgp,props,weigp,melem,mtotg, &
                              strsg,ntype,ncrit)
!
!*** solve equations
!
                              call front(asdis,eload,eqrhs,equat,estif,fixed,iffix,iincs, &
                                    iiter,gload,gstif,locel,lnods,kresl,mbufa,melem, &
                                    mevab,mfron,mstif,mtotv,mvfix,nacva,namev,ndest, &
                                    ndofn,nelem,nevab,nnode,nofix,npivo,npoin, &
                                    ntotv,tdisp,tload,treac,vecrv)
!
!*** calculate residual forces
!
                              call residu(asdis,coord,effst,eload,facto,iiter,lnods, &
                                    lprop,matno,melem,mmats,mpoin,mtotg,mtotv,ndofn, &
```

```
                                    nelem,nevab,ngaus,nnode,nstr1,ntype,posgp,props, &
                                    nstre,ncrit,strsg,weigp,tdisp,epstn)
!
!*** check for convergence
!
                        call conver(eload,iiter,lnods,melem,mevab,mtotv,nchek &
                                ,ndofn,nelem,nevab,nnode,ntotv,pvalu,stfor, &
                                tload,tofor,toler)
!
!*** output results if required
!
                        if (iiter.eq.1.and.noutp(1).gt.0) &
                        call output(iiter,mtotg,mtotv,mvfix,nelem,ngaus,nofix, &
                                noutp,npoin,nvfix,strsg,tdisp,treac,epstn, &
                                ntype,nchek)
!
!*** if solution has converged stop iterating and output results for load increment
!
                        if (nchek.eq.0) goto 75
                50 continue
!
!***
!
                if (nalgo.eq.2) goto 75 !for tangential stiffness method
                stop
                75              call work(plwork,elwork,coord,props,posgp,weigp,epstn,effst,matno, &
                                        lnods,nelem,nnode,ngaus,ntype,mpoin,mtotg,mmats,ndofn, &
                                        melem,iincs) !calculate the elastic & plastic work so far
!N.B.: The subroutine calculates the work for a bi-linear hardening material model
!
                                    call output(iiter,mtotg,mtotv,mvfix,nelem,ngaus,nofix, &
                                    noutp,npoin,nvfix,strsg,tdisp,treac,epstn, &
                                    ntype,nchek)

        100 continue
stop
end

! Function for converting 2x2 matrix into 1D matrix, used in subroutine FRONT
!       function nfunc(i,j)
!               integer :: i,j,nfunc
!               nfunc(i,j)=(j*j-j)/2+i
!       end function nfunc
```

SUBROUTINE **RESIDU**

```
Subroutine residu(asdis,coord,effst,eload,facto,iiter,lnods, &
                        lprop,matno,melem,mmats,mpoin,mtotg,mtotv,ndofn, &
                        nelem,nevab,ngaus,nnode,nstr1,ntype,posgp,props, &
                        nstre,ncrit,strsg,weigp,tdisp,epstn)
!**********************************************************************
!
!***** This subroutine reduces the stresses to the yield surface and
```

```
! **** evaluates the equivalent nodal forces
!
!*********************************************************************
!
implicit none
!
        integer :: mtotv,mpoin,mtotg,melem,mmats,ielem,nelem,ievab,nevab,kgaus
        integer :: lprop,ncrit,inode,nnode,lnode,nposn,ndofn,idofn,ntype
        integer :: kgasp,igaus,ngaus,jgaus,nstre,istr1,nstr1
        integer :: mstep,istep,mgash,istre,iiter
        double precision :: root3,twopi,uniax,hards,frict,thick,exisp,etasp,preys,sint3
        double precision :: steff,theta,varj2,yield,espre,escur,rfact,astep,reduc,abeta,agash
        double precision :: dlamd,bgash,curys,bring,facto,djacb,dvolu
        double precision ::  asdis(mtotv),avect(4),cartd(2,9),coord(mpoin,2), &
                devia(4),dvect(4),effst(mtotg),elcod(2,9),eldis(2,9), &
                eload(melem,18),posgp(4),props(mmats,7), &
                stran(4),stres(4),strsg(4,mtotg), &
                weigp(4),dlcod(2,9),desig(4),sigma(4),sgtot(4), &
                dmatx(4,4),deriv(2,9),shap(9),gpcod(2,9), &
                epstn(mtotg),tdisp(mtotv),bmatx(4,18)
        integer :: lnods(melem,9),matno(melem)
!
        root3=1.73205080757
        twopi=6.283185308
!
        do 10 ielem=1,nelem         !zero the array in which the equivalent nodal forces
        do 10 ievab=1,nevab         !calculated in step h, will be stored
         10 eload(ielem,ievab)=0.0
!
        kgaus=0 !zero the Gauss point counter over all elements
        do 20 ielem=1,nelem !loop over each element
                lprop=matno(ielem) !identify the element material property number
                uniax=props(lprop,5) !identify the initial uniaxial yield stress, the linear strain
                hards=props(lprop,6)        !hardening parameter H',
                frict=props(lprop,7)        !and the friction angle for Mohr Coulomb
                if (ncrit.eq.3) uniax=props(lprop,5)*cos(frict*0.017453292) !used for Mohr Coulomb
                if (ncrit.eq.4) uniax=6.0*props(lprop,5)*cos(frict*0.017453292)/ &
                        (root3*(3.0-sin(frict*0.017453292)))          !used for drucker-prager
!
!*** compute coordinate and incremental displacements of the
!*** element nodal points
!store the element nodal coords. in array ELCOD and the nodal displacements due to the application
!of the residual forces or load increment for iteration 1 in array ELDIS
                do 30 inode=1,nnode
                        lnode=iabs(lnods(ielem,inode))
                        nposn=(lnode-1)*ndofn
                do 30 idofn=1,ndofn
                        nposn=nposn+1
                        elcod(idofn,inode)=coord(lnode,idofn)
                30 eldis(idofn,inode)=asdis(nposn)
!!
                call modps(dmatx,lprop,mmats,ntype,props) !evaluate the elastic D matrix
                thick=props(lprop,3) !identify the element thickness
                kgasp=0 !zero the local Gauss point counter
                do 40 igaus=1,ngaus !enter the loops for numerical integration and evaluate the local
                do 40 jgaus=1,ngaus !coords.(ksi,neta) at the sampling point
```

```
                        exisp=posgp(igaus)
                        etasp=posgp(jgaus)
                        kgaus=kgaus+1 !increment the local
                        kgasp=kgasp+1 !and global gauss point counters
                        call sfr2(deriv,etasp,exisp,nnode,shap) !evaluate the shape functs. & their
derivatives
!evaluate the Gauss point coords. GPCOD(IDIME,KGASP), the det. of the jacobian matrix
!and the cartesian derivatives of the shape functions
                        call jacob2(cartd,deriv,djacb,elcod,gpcod,ielem,kgasp,nnode,shap)
                        dvolu=djacb*weigp(igaus)*weigp(jgaus)
!calculate the elemental volume for numerical integration - for plane stress/strain the default
!value of the thickness is 1.0
                        if (ntype.eq.3) dvolu=dvolu*twopi*gpcod(1,kgasp)
                        if (thick.ne.0.0) dvolu=dvolu*thick
!Compute the strain matrix B for the Gauss point
                        call bmatps (bmatx,cartd,nnode,shap,gpcod,ntype,kgasp)
!compute the strains & stress increment STRES(ISTR1), assuming elastic behaviour.
                        call linear (cartd,dmatx,eldis,lprop,mmats,ndofn,nnode,nstre, &
                                        ntype,props,stran,stres,kgasp,gpcod,shap)
!Compute the yield stress for the previous iteration as sigmaYo + H' x equiv. plast. strain (at
!
                the gauss point) of previous iteration
                        preys=uniax+epstn(kgaus)*hards
!       store increment of the stress vector as DESIG(ISTR1) and total stress vector as SIGMA(ISTR1)
                        do 150 istr1=1,nstr1
                                desig(istr1)=stres(istr1)
                        150 sigma(istr1)=strsg(istr1,kgaus)+stres(istr1)
!testing stuff
!  do 634 istr1=1,nstr1
!                       634 write(60,*) 'stress field 0011',sigma(istr1)
!end test stuff
!
!Evaluate the effective stress (due to the vector SIGMA(ISTR1)) at the gauss point & store as YIELD
                        call invar(devia,lprop,mmats,ncrit,props,sint3,steff,sigma, &
                                theta,varj2,yield,ntype)
!write (60,*) 'YIELD',yield
!write (60,*) 'PREYS',preys

!Check if the Gauss point has yielded on the previous iteration
!               effst(kgaus) gives stress adjusted to satisfy the yield criterion for (r-1)th iteration
                        espre=effst(kgaus)-preys
!write (60,*) 'espre',espre
                        if (espre.ge.0.0) goto 50 !if it has yielded previously goto 50
!If the Gauss point was previously elastic, check to see if it has yielded during this iteration
!       yield is the current effective stress (for value see calc. in previous lines)
                        escur=yield-preys !preys=previous yield stress
                        if (escur.le.0.0) goto 60 !if it has not yielded during the current iter. goto 60
                        rfact=escur/(yield-effst(kgaus)) !Calc. R for gauss pt which yields during current
iter
                        goto 70
! check to see if a Gauss point which had previously yielded is unloading during this iteration
! if yes goto 60
                        50 escur=yield-effst(kgaus) !effst is the effective stress of the previous iteration
                        if (escur.le.0.0) goto 60 !if point is unloading goto 60
                        rfact=1.0 !otherwise set R=1
!Evaluate the no. of steps into which the excess stress is to be divided according to 7.94 pg253
```

```
                        70 mstep=escur*8.0/uniax+1.0
                        astep=mstep
                        reduc=1.0-rfact
!
! For yielded gauss pts. only compute the portion of the total stress which satisfies the yield
! criterion & store in SGTOT & evaluate the portion which needs to be reduced bit by bit in mstep s
! & store in STRES (for more details see step e pg251)
                        do 80 istr1=1,nstr1
                                sgtot(istr1)=strsg(istr1,kgaus)+reduc*stres(istr1)
                        80 stres(istr1)=rfact*stres(istr1)/astep
!
! Loop over each stress reduction step
                        do 90 istep=1,mstep
                        ! ** Compute the vectors [a] & [dD] **
                                call invar (devia,lprop,mmats,ncrit,props,sint3,steff,sgtot, &
                                        theta,varj2,yield,ntype)
!here yield gives value of eff. stress from which the step leaves to
!       cause a wee inc. in plastic strain
!thus here we make use of the prandtl-reuss eqns., see crux of theorem on Mendelson pg100
                                call yieldf (avect,devia,lprop,mmats,ncrit,nstr1, &
                                        props,sint3,steff,theta,varj2) !This subroutine evaluates
                                        !
        the flow vector 'a' defined in (7.74)
                                call flowpl (avect,abeta,dvect,ntype,props,lprop,nstr1,mmats) !This
subroutine
                                !
        evaluates the plastic D vector (dD) see pg243
! Compute dlambda according to 7.45
                                agash=0.0
                                do 100 istr1=1,nstr1
                                100 agash=agash+avect(istr1)*stres(istr1)
                                dlamd=agash*abeta
                                if (dlamd.lt.0.0) dlamd=0.0
!
! See note on Owen & Hinton pg257 RSDU 93-96
                                bgash=0.0
                                do 110 istr1=1,nstr1
                                        bgash=bgash+avect(istr1)*sgtot(istr1)
                                110 sgtot(istr1)=sgtot(istr1)+stres(istr1)-dlamd*dvect(istr1)
! sgtot(istr1) starts loop as stress vector which satisfies the yield criterion
! but after each loop sgtot gives the stress vector after wee plastic strain occurs
! see eqn. given in RSDU 93-96 pg257
                                epstn(kgaus)=epstn(kgaus)+dlamd*bgash/yield !compute the eff.
plastic strain
                                !
                        (7.51) pg228 & (7.96) pg257
                        90 continue !return to loop over next stress reduction step.see RSDU98 pg257
                        ! after the m loops, sgtot gives sigma(r) on pg257 & as in figs7.10 pg251
                        ! and epstn(kgaus) gives the accumulated eff. plastic strain at the gauss point
! *Note 1:so in effect we have relaxed a wee bit the excess stresses by calculating an estimated
! value for plastic strain - we repeat the process for enough iterations until convergence
! is satisfied
                        call invar(devia,lprop,mmats,ncrit,props,sint3,steff,sgtot, &
                                theta,varj2,yield,ntype) !compute the effective stress 'yield'
!write(60,*) 'yield1=',yield
                                !
```

```
                              curys=uniax+epstn(kgaus)*hards !evaluate current yield stress from the
                              !                                                    accumulated eff. plastic strain at
the gauss point
! *Note 2: notice that after calculating the eff. plastic strain we must be sure that we are
! obeying the material model - ie calculate the proper yield stress for the calculated amount
! of eff. plastic strain - then again we perform another iteration to try to get equilibrium
! of external forces with internal stresses mentioned in note 1 above
! Factor the stresses to ensure that they lie on the yield surface

                              ! testing
!write(60,*) 'curys=',curys

!do 134 istr1=1,nstr1
!                             134 write(60,*) 'stress field 1',strsg(istr1,kgaus)
                              !end testing

                              bring=1.0
                              if (yield.gt.curys) bring=curys/yield
                              do 130 istr1=1,nstr1
                              130 strsg(istr1,kgaus)=bring*sgtot(istr1)
                              ! strsg gives current stress vector which satisfies the yield criteria
                              !
                              ! testing
!do 133 istr1=1,nstr1
!                             133 write(60,*) 'stress field',strsg(istr1,kgaus)
                              !end testing
                              !
                              effst(kgaus)=bring*yield !store the eff. stress in array EFFST to be
                              !                                                          used for next
iteration where it becomes (sigma^r-1)
!write(60,*) 'bring=',bring
!write(60,*) 'yield=',yield
!write(60,*) 'curys=',curys
!write(60,*) 'residu effst(',kgaus,')=',effst(kgaus)
!
!*** alternative location of stress reduction loop termination card
! 90 continue  !see RSDU108 pg257
!
! For elastic gauss points compute sigma and store in effst
                              goto 190
                              60 do 180 istr1=1,nstr1
                                    180 strsg(istr1,kgaus)=strsg(istr1,kgaus)+desig(istr1)
                              effst(kgaus)=yield !to be used for next iteration where it becomes (sigma^r-1)
!
!*** calculate the equivalent nodal forces and associate with the element nodes
!
                              190 mgash=0
                              do 140 inode=1,nnode
                              do 140 idofn=1,ndofn
                                    mgash=mgash+1
                              do 140 istre=1,nstre
                              140 eload(ielem,mgash)=eload(ielem,mgash)+bmatx(istre,mgash)* &
                                    strsg(istre,kgaus)*dvolu
                        40 continue !Termination of loop for numerical integration
                  20 continue             !Termination of loop over each element
return
```

```
end
```

SUBROUTINE **SFR2**

```
Subroutine sfr2(deriv,etasp,exisp,nnode,shap)
!**********************************************************************
!
!***** This subroutine evaluates shape functions and their derivatives
!           for linear, quadratic lagrangian and serendipity isoparametric
!           2D elements
!
!**********************************************************************
!
implicit none
!
           integer :: nnode
           double precision :: s,t,exisp,etasp,st,s2,t2,ss,tt,sst,stt,st2,s1,t1,s9,t9
           double precision :: deriv(2,9),shap(9)
           s=exisp
           t=etasp
           if (nnode.gt.4) goto 10
           st=s*t
!
!*** Shape functions for 4 noded element
!
           shap(1)=(1-t-s+st)*0.25
           shap(2)=(1-t+s-st)*0.25
           shap(3)=(1+t+s+st)*0.25
           shap(4)=(1+t-s-st)*0.25
!
!***       shape function derivatives
!
           deriv(1,1)=(-1+t)*0.25
           deriv(1,2)=(+1-t)*0.25
           deriv(1,3)=(+1+t)*0.25
           deriv(1,4)=(-1-t)*0.25
           deriv(2,1)=(-1+s)*0.25
           deriv(2,2)=(-1-s)*0.25
           deriv(2,3)=(+1+s)*0.25
           deriv(2,4)=(+1-s)*0.25
           return
10         if (nnode.gt.8) goto 30
           s2=s*2.0
           t2=t*2.0
           ss=s*s
           tt=t*t
           st=s*t
           sst=s*s*t
           stt=s*t*t
           st2=s*t*2.0
!
!***       shape functions for 8 noded element
!
```

```
            shap(1)=(-1.0+st+ss+tt-sst-stt)/4.0
            shap(2)=(1.0-t-ss+sst)/2.0
            shap(3)=(-1.0-st+ss+tt-sst+stt)/4.0
            shap(4)=(1.0+s-tt-stt)/2.0
            shap(5)=(-1.0+st+ss+tt+sst+stt)/4.0
            shap(6)=(1.0+t-ss-sst)/2.0
            shap(7)=(-1.0-st+ss+tt+sst-stt)/4.0
            shap(8)=(1.0-s-tt+stt)/2.0
!
!***        shape function derivatives
!
            deriv(1,1)=(t+s2-st2-tt)/4.0
            deriv(1,2)=-s+st
            deriv(1,3)=(-t+s2-st2+tt)/4.0
            deriv(1,4)=(1.0-tt)/2.0
            deriv(1,5)=(t+s2+st2+tt)/4.0
            deriv(1,6)=-s-st
            deriv(1,7)=(-t+s2+st2-tt)/4.0
            deriv(1,8)=(-1.0+tt)/2.0
            deriv(2,1)=(s+t2-ss-st2)/4.0
            deriv(2,2)=(-1.0+ss)/2.0
            deriv(2,3)=(-s+t2-ss+st2)/4.0
            deriv(2,4)=-t-st
            deriv(2,5)=(s+t2+ss+st2)/4.0 !!!!!! might be wrong
            deriv(2,6)=(1.0-ss)/2.0
            deriv(2,7)=(-s+t2+ss-st2)/4.0
            deriv(2,8)=-t+st
!WRITE(34,*) "deriv(s1) =",deriv(1,1)
!WRITE(34,*) "deriv(s2) =",deriv(1,2)
!WRITE(34,*) "deriv(s3) =",deriv(1,3)
!WRITE(34,*) "deriv(s4) =",deriv(1,4)
!WRITE(34,*) "deriv(s5) =",deriv(1,5)
!WRITE(34,*) "deriv(s6) =",deriv(1,6)
!WRITE(34,*) "deriv(s7) =",deriv(1,7)
!WRITE(34,*) "deriv(s8) =",deriv(1,8)
!WRITE(34,*) "deriv(t1) =",deriv(2,1)
!WRITE(34,*) "deriv(t2) =",deriv(2,2)
!WRITE(34,*) "deriv(t3) =",deriv(2,3)
!WRITE(34,*) "deriv(t4) =",deriv(2,4)
!WRITE(34,*) "deriv(t5) =",deriv(2,5)
!WRITE(34,*) "deriv(t6) =",deriv(2,6)
!WRITE(34,*) "deriv(t7) =",deriv(2,7)
!WRITE(34,*) "deriv(t8) =",deriv(2,8)
            return
30          continue
            ss=s*s
            st=s*t
            tt=t*t
            s1=s+1.0
            t1=t+1.0
            s2=s*2.0
            t2=t*2.0
            s9=s-1.0
            t9=t-1.0
!
!*** shape functions for 9 noded element
```

```
!
          shap(1)=0.25*s9*st*t9
          shap(2)=0.5*(1.0-ss)*t*t9
          shap(3)=0.25*s1*st*t9
          shap(4)=0.5*s*s1*(1.0-tt)
          shap(5)=0.25*s1*st*t1
          shap(6)=0.5*(1.0-ss)*t*t1
          shap(7)=0.25*s9*st*t1
          shap(8)=0.5*s*s9*(1.0-tt)
          shap(9)=(1.0-ss)*(1.0-tt)
!
!*** shape function derivatives
!
          deriv(1,1)=0.25*t*t9*(-1.0+s2)
          deriv(1,2)=-st*t9
          deriv(1,3)=0.25*(1.0+s2)*t*t9
          deriv(1,4)=0.5*(1.0+s2)*(1.0-tt)
          deriv(1,5)=0.25*(1.0+s2)*t*t1
          deriv(1,6)=-st*t1
          deriv(1,7)=0.25*(-1.0+s2)*t*t1
          deriv(1,8)=0.5*(-1.0+s2)*(1.0-tt)
          deriv(1,9)=-s2*(1.0-tt)
          deriv(2,1)=0.25*(-1.0+t2)*s*s9
          deriv(2,2)=0.5*(1.0-ss)*(-1.0+t2)
          deriv(2,3)=0.25*s*s1*(-1.0+t2)
          deriv(2,4)=-st*s1
          deriv(2,5)=0.25*s*s1*(1.0+t2)
          deriv(2,6)=0.5*(1.0-ss)*(1.0+t2)
          deriv(2,7)=0.25*s*s9*(1.0+t2)
          deriv(2,8)=-st*s9
          deriv(2,9)=-t2*(1.0-ss)

20        continue
          return
          end
```

SUBROUTINE **STIFFP**

```
Subroutine stiffp(coord,epstn,iincs,lnods,matno,mevab,mmats, &
                                mpoin,mtotv,nelem,nevab,ngaus,nnode,nstre, &
                                nstr1,posgp,props,weigp,melem,mtotg, &
                                strsg,ntype,ncrit)
!
implicit none
!
!**********************************************************************
!
!***** This subroutine evaluates the stiffness matrix for each element in turn
!                    Owen and Hinton pg244 for detailed explanation
!
! The element stiffnesses are always calculated for the first iteration of the
! first load increment and elastic behaviour at this step is assumed.
! Every other time this subroutine is accessed, the stifnesses are to be recalculated
```

```
! to account for any plastic deformation of the material and so the Dep matrix
! must be employed
!
!*********************************************************************
!
        integer :: melem,mpoin,mtotg,mmats,kgaus,ielem,nelem,lprop,inode,nnode,lnode
        integer :: iposn,idime,ievab,nevab,jevab,kgasp,igaus,ngaus,jgaus,ntype
        integer :: iincs,istr1,nstr1,ncrit,istre,nstre,jstre,mevab,mtotv
        integer :: iii,jjj,ie,i,j,k,ii1,ii2,ki1,ki2,l,jj1,jj2,lj1,lj2
        double precision :: twopi,thick,exisp,etasp,djacb,dvolu,sint3,steff,theta,varj2,yield
        double precision :: abeta
        double precision :: bmatx(4,18),cartd(2,9),coord(mpoin,2),dbmat(4,18), &
                deriv(2,9),devia(4),dmatx(4,4), &
                elcod(2,9),epstn(mtotg),estif(18,18), &
                posgp(4),props(mmats,7),shap(9), &
                weigp(4),stres(4),strsg(4,mtotg), &
                dvect(4),avect(4),gpcod(2,9)
        integer :: lnods(melem,9),matno(melem)
!
        twopi=6.283185308
        rewind 1 !each element stiffness matrix will be stored in this file
! set to zero the counter which indicates the overall Gauss point location
! So KGAUSS ranges from 1 to NGAUS*NGAUS*NELEM (NELEM is the tot. no. of elements)
        kgaus=0
!
!*** loop over each element
!
        do 70 ielem=1,nelem
                lprop=matno(ielem) !identify the mat. prop, type of the current elem.
!
!*** evaluate the coordinates of the element nodal points
!
                do 10 inode=1,nnode
                        lnode=iabs(lnods(ielem,inode))
                        iposn=(lnode-1)*2
                do 10 idime=1,2
                iposn=iposn+1
                10 elcod(idime,inode)=coord(lnode,idime)
                thick=props(lprop,3) !identify the element thickness
!
!*** initialize to zero the element stiffness matrix
!
                do 20 ievab=1,nevab
                do 20 jevab=1,nevab
                20 estif(ievab,jevab)=0.0
! set to zero the element Gauss point counter - so KGASP ranges from 1
! to NGAUSS*NGAUSS
                kgasp=0
!
!*** enter loops for area numerical integration
!
                do 50 igaus=1,ngaus
                        exisp=posgp(igaus) !Gauss point position in parent element
                do 50 jgaus=1,ngaus
                        etasp=posgp(jgaus)
                        kgasp=kgasp+1 !inc. the element Gauss point counter
```

```
                        kgaus=kgaus+1 !inc. counter which indicates the overall Gauss point location

!
!*** evaluate the D matrix
!
                    call modps(dmatx,lprop,mmats,ntype,props)
!
!*** evaluate the shape functions, elemental, volume, etc
!
                    call sfr2(deriv,etasp,exisp,nnode,shap)
                    !'shap' gives the shape function matrix
                    !'deriv' gives the B-matrix
                    call jacob2(cartd,deriv,djacb,elcod,gpcod,ielem,kgasp, &
                            nnode,shap)
                    !'djacb' gives the value of the Jacobi determinant
                            dvolu=djacb*weigp(igaus)*weigp(jgaus)
                            if (ntype.eq.3) dvolu=dvolu*twopi*gpcod(1,kgasp)
                            if (thick.ne.0.0) dvolu=dvolu*thick
!
!*** evaluate the B and DB matrices
!
                    call bmatps(bmatx,cartd,nnode,shap,gpcod,ntype,kgasp)
                    if (iincs.eq.1) goto 80 !for the first time avoid the replacement of D
                    ! by Dep as defined in (7.47)
                    if (epstn(kgaus).eq.0.0) goto 80 !also for gauss points at which there is
                    ! no plastic strain avoid the replacement of D by Dep
                            do 90 istr1=1,nstr1 !store the total current stresses in the array STRES
                            90 stres(istr1)=strsg(istr1,kgaus)
                    !Call the following 3 subroutines to evaluate stuff which are in turn
                    ! needed to evaluate Dep - see owen & hinton pg247
                    call invar(devia,lprop,mmats,ncrit,props,sint3,steff,stres, &
                            theta,varj2,yield,ntype)
                    call yieldf(avect,devia,lprop,mmats,ncrit,nstr1, &
                            props,sint3,steff,theta,varj2)
                    call flowpl(avect,abeta,dvect,ntype,props,lprop,nstr1,mmats)
                    !Evaluate Dep according to (7.47)
                    do 100 istre=1,nstre
                    do 100 jstre=1,nstre
                    100 dmatx(istre,jstre)=dmatx(istre,jstre)-abeta*dvect(istre)* &
                                    dvect(jstre)
                    80 continue
                    ! Evaluate D.B
                    call dbe(bmatx,dbmat,dmatx,mevab,nevab,nstre,nstr1)
!
!*** calculate the element stiffness (upper triangle part)
!           at current Gauss point of current element
!
                            do 30 ievab=1,nevab
                            do 30 jevab=ievab,nevab
                            do 30 istre=1,nstre
                            30 estif(ievab,jevab)=estif(ievab,jevab)+bmatx(istre,ievab)* &
                                    dbmat(istre,jevab)*dvolu
                    50 continue

!
!*** construct the lower triangle of the  stiffness matrix by symmetry
```

```
!** note that this is the element stiffness matrix which needs to be assembled
!  into the global stiffness matrix so take heed of this note if you change
! the solver
                    do 60 ievab=1,nevab
                    do 60 jevab=1,nevab
                    60 estif(jevab,ievab)=estif(ievab,jevab)

!*** store the  element stiffness matrix,stress matrix and sampling point
!           coordinates for each element on disk file 1 named 'scratch'
!
                    write(1) estif
!                   write(34,*) 'element stiffness',ielem
!                   write(34,*) estif
!                   write(3) !perhaps missing - see O.hinton small pg116
          70 continue
return
end
```

SUBROUTINE **WORK**

```
subroutine work(plwork,elwork,coord,props,posgp,weigp,epstn,effst,matno, &
                                        lnods,nelem,nnode,ngaus,ntype,mpoin,mtotg,mmats,ndofn, &
                                        melem,iincs)
!
!**********************************************************************
!
!***** This subroutine calculates the elastic & plastic work so far
!
!**********************************************************************
!
implicit none
!
integer :: kgaus,kgasp,ielem,nelem,lprop,lnode,inode,nnode,nposn,ndofn,idofn
integer :: igaus,jgaus,ngaus,mpoin,mmats,mtotg,melem,ntype,iincs
double precision :: coord(mpoin,2),props(mmats,7),elcod(2,9),posgp(4),weigp(4),workratio
double precision :: epstn(mtotg),effst(mtotg),deriv(2,9),shap(9),cartd(2,9),gpcod(2,9)
integer :: matno(melem),lnods(melem,9)
double precision :: plwork,elwork,hards,emodulus,thick,uniax,exisp,etasp,djacb,dvolu,twopi
!
!           Reset work values. N.B. Each time this subroutine is called the whole of the
!           total work done is calculated so there is a need to reset the work values to zero
          plwork=0
          elwork=0
          twopi=6.283185308
          kgaus=0 !zero the Gauss point counter over all elements
          do 203 ielem=1,nelem !loop over each element
                    lprop=matno(ielem) !identify the element material property number
                    uniax=props(lprop,5) !identify the initial uniaxial yield stress,
                    hards=props(lprop,6) !the linear strain hardening parameter H',
                    thick=props(lprop,3) !identify the element thickness
                    Emodulus=props(lprop,1) !identify the elastic Young's modulus
!
!following is for testing purposes
```

```
!write(60,*) 'uniax=', uniax
!write(60,*) 'hards=', hards
!write(60,*) 'thick=', thick
!write(60,*) 'Emodulus=', Emodulus
!
!store the element nodal coords. in array ELCOD
                    do 30 inode=1,nnode
                              lnode=iabs(lnods(ielem,inode))
                              nposn=(lnode-1)*ndofn
                    do 30 idofn=1,ndofn
                              nposn=nposn+1
                    30        elcod(idofn,inode)=coord(lnode,idofn)
!
!following is for testing purposes
!write(60,*) 'nodal coords='
!write(60,*)  elcod
!
                     kgasp=0 !zero the local Gauss point counter
                     do 40 igaus=1,ngaus !enter the loops for numerical integration and evaluate
                     do 40 jgaus=1,ngaus !the local coords.(ksi,neta) at the sampling point
                              exisp=posgp(igaus)
                              etasp=posgp(jgaus)
                              kgaus=kgaus+1 !increment the local
                              kgasp=kgasp+1 !and global gauss point counters
                              call sfr2(deriv,etasp,exisp,nnode,shap) !evaluate the shape functs.
                              ! & their derivatives, evaluate the Gauss point coords.
GPCOD(IDIME,KGASP),
          !the det. of the jacobian matrix and the cartesian derivatives of the shape functions
                              call jacob2(cartd,deriv,djacb,elcod,gpcod,ielem,kgasp,nnode,shap)
                              dvolu=djacb*weigp(igaus)*weigp(jgaus)
!
!write(60,*) "detj =", djacb
!write(60,*) "weigp"
!write(60,*) weigp
!
                    !calculate the elemental volume for numerical integration - for plane stress/strain
                    !the default value of the thickness is 1.0
                              if (ntype.eq.3) dvolu=dvolu*twopi*gpcod(1,kgasp)
                              if (thick.ne.0.0) dvolu=dvolu*thick
!write(60,*) "effst(",kgaus,")=", effst(kgaus)
!write(60,*) "epstn(",kgaus,")=", epstn(kgaus)
!write(60,*) "dvolu=", dvolu
                              plwork=plwork+(0.5*(effst(kgaus)+uniax)*epstn(kgaus)*dvolu)
!wrong me thinks               plwork=plwork+((effst(kgaus))*epstn(kgaus)*dvolu)
                     40       elwork=elwork+((effst(kgaus)*effst(kgaus)*0.5/emodulus)*dvolu)
          203 continue
!Write to text file load,elwork,plwork,plwork/elwork at end of increment
workratio=plwork/elwork
write(60,*) "LOAD INCREMENT",iincs
write(60,*) "---------------------------------------"
write(60,*) "elwork       =",elwork
write(60,*) "plwork       =",plwork
write(60,*) "plwork/elwork =",workratio
write(60,*) " "
```

```
!write(60,930)
!                   930 format ("increment",5x,"elwork",5x,"plwork",5x,"plwork/elwork")
!write(60,931) iincs,elwork,plwork,workratio
!                   931 format (5i,e14.6,e14.6,e14.6)
!
!
return
end
```

SUBROUTINE **YIELDF**

```
Subroutine yieldf(avect,devia,lprop,mmats,ncrit,nstr1, &
                                props,sint3,steff,theta,varj2)
!
!**********************************************************************
!
!***** This subroutine evaluates the flow vector 'a' defined in (7.74)
!
!**********************************************************************
!
implicit none
!
        integer :: lprop,mmats,istr1,nstr1,ncrit
        double precision :: theta,tanth,tant3,sinth,costh,cost3,steff,frict,root3,varj2
        double precision :: cons1,abthe,cons2,cons3,plumi,snphi,sint3
        double precision :: avect(4),devia(4),props(mmats,7), &
                                veca1(4),veca2(4),veca3(4)
!
        if (steff.eq.0.0) return
        frict=props(lprop,7)
        tanth=tan(theta)
        tant3=tan(3.0*theta)
        sinth=sin(theta)
        costh=cos(theta)
        cost3=cos(3.0*theta)
        root3=1.73205080757
!
!*** calculate vector A1
!
        veca1(1)=1.0
        veca1(2)=1.0
        veca1(3)=0.0
        veca1(4)=1.0
!
!*** calculate vector A2
!
        do 10 istr1=1,nstr1
        10 veca2(istr1)=devia(istr1)/(2.0*steff)
        veca2(3)=devia(3)/steff
!
!*** calculate vector A3
!
        veca3(1)=devia(2)*devia(4)+varj2/3.0
```

```
            veca3(2)=devia(1)*devia(4)+varj2/3.0
            veca3(3)=-2.0*devia(3)*devia(4)
            veca3(4)=devia(1)*devia(2)-devia(3)*devia(3)+varj2/3.0
            goto (1,2,3,4) ncrit
!
!*** tresca
!
            1 cons1=0.0
            abthe=abs(theta*57.29577951308)
            if (abthe.lt.29.0) goto 20
            cons2=root3
            cons3=0.0
            goto 40
            20 cons2=2.0*(costh+sinth*tant3)
            cons3=root3*sinth/(varj2*cost3)
            goto 40
!
!*** von mises
!
            2 cons1=0.0
            cons2=root3
            cons3=0.0
            goto 40
!
!*** mohr-coloumb
!
            3 cons1=sin(frict*0.017453292)/3.0
            abthe=abs(theta*57.29577951308)
            if (abthe.lt.29.0) goto 30
            cons3=0.0
            plumi=1.0
            if (theta.gt.0.0) plumi=-1.0
            cons2=0.5*(root3+plumi*cons1*root3)
            goto 40
            30 cons2=costh*((1.0+tanth*tant3)+cons1*(tant3-tanth)*root3)
            cons3=(root3*sinth+3.0*cons1*costh)/(2.0*varj2*cost3)
            goto 40
!
!*** drucker-prager
!
            4 snphi=sin(frict*0.017453292)
            cons1=2.0*snphi/(root3*(3.0-snphi))
            cons2=1.0
            cons3=0.0
            40 continue
            do 50 istr1=1,nstr1
            50 avect(istr1)=cons1*veca1(istr1)+cons2*veca2(istr1)+cons3*veca3(istr1)
return
end
```

SUBROUTINE **ZERO**

```
Subroutine zero(eload,melem,mevab,mpoin,mtotg,mtotv,ndofn,nelem, &
```

```
                    nevab,ngaus,nstr1,ntotg,epstn,effst, &
                    ntotv,nvfix,strsg,tdisp,tfact, &
                    tload,treac,mvfix)
!**********************************************************************
!
!***** This subroutine initialises various arrays to zero
!
!**********************************************************************
!
implicit none
!
          integer :: melem,mevab,mtotg,mtotv,mvfix,ielem,nelem,ievab,nevab,itotv
          integer :: ntotv,ivfix,nvfix,idofn,ndofn,itotg,ntotg,istr1,nstr1,mpoin,ngaus
          double precision :: tfact
          double precision :: eload(melem,mevab),strsg(4,mtotg),tdisp(mtotv), &
                                    tload(melem,mevab),treac(mvfix,2),epstn(mtotg),effst(mtotg)
          tfact=0.0
          do 30 ielem=1,nelem
          do 30 ievab=1,nevab
                    eload(ielem,ievab)=0.0
                    30 tload(ielem,ievab)=0.0
          do 40 itotv=1,ntotv
                    40 tdisp(itotv)=0.0
          do 50 ivfix=1,nvfix
          do 50 idofn=1,ndofn
                    50 treac(ivfix,idofn)=0.0
          do 60 itotg=1,ntotg
                    epstn(itotg)=0.0
                    effst(itotg)=0.0
          do 60 istr1=1,nstr1
                    60 strsg(istr1,itotg)=0.0 !strsg are the gauss point stresses
write(34,*) 'aw hii'
return
end
```