```
************************************************************************
*******
*
* Gasperini P. and Vannucci G., FPSPACK: a package of simple Fortran
subroutines
* to manage earthquake focal mechanism data
*
************************************************************************
*******
*     BASIC ROUTINES
************************************************************************
*******
      subroutine pl2nd(strike,dip,rake,anx,any,anz,dx,dy,dz,ierr)
c
c     compute Cartesian components of outward normal and slip
c     vectors from strike, dip and rake
c
c     usage:
c     call pl2nd(strike,dip,rake,anx,any,anz,dx,dy,dz,ierr)
c
c     arguments:
c     strike         strike angle in degrees (INPUT)
c     dip            dip angle in degrees (INPUT)
c     rake           rake angle in degrees (INPUT)
c     anx,any,anz    components of fault plane outward normal versor in
the
c                    Aki-Richards Cartesian coordinate system (OUTPUT)
c     dx,dy,dz       components of slip versor in the Aki-Richards
c                    Cartesian coordinate system (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input STRIKE angle out of range
c     2              input DIP angle out of range
c     4              input RAKE angle out of range
c     3              1+2
c     5              1+4
c     7              1+2+4
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real anx,any,anz,dx,dy,dz,strike,dip,rake,wstrik,wdip,wrake
      integer ierr
c
      call fpsset
c
      anx=c0
      any=c0
      anz=c0
      dx=c0
      dy=c0
```

```
      dz=c0
      ierr=0
      if(strike.lt.amistr.or.strike.gt.amastr) then
         write(io,'(1x,a,g10.4,a)') 'PL2ND: input STRIKE angle ',strike,
     1   ' out of range'
         ierr=1
      endif
      if(dip.lt.amidip.or.dip.gt.amadip) then
         if(dip.lt.amadip.and.dip.gt.-ovrtol) then
            dip=amidip
         else if(dip.gt.amidip.and.dip-amadip.lt.ovrtol) then
            dip=amadip
         else
            write(io,'(1x,a,g10.4,a)') 'PL2ND: input DIP angle ',dip,
     1      ' out of range'
            ierr=ierr+2
         endif
      endif
      if(rake.lt.amirak.or.rake.gt.amarak) then
         write(io,'(1x,a,g10.4,a)') 'PL2ND: input RAKE angle ',rake,
     1   ' out of range'
         ierr=ierr+4
      endif
      if(ierr.ne.0) return
      wstrik=strike*dtor
      wdip=dip*dtor
      wrake=rake*dtor
c
      anx=-sin(wdip)*sin(wstrik)
      any=sin(wdip)*cos(wstrik)
      anz=-cos(wdip)
      dx=cos(wrake)*cos(wstrik)+cos(wdip)*sin(wrake)*sin(wstrik)
      dy=cos(wrake)*sin(wstrik)-cos(wdip)*sin(wrake)*cos(wstrik)
      dz=-sin(wdip)*sin(wrake)
      return
      end
************************************************************************
*******
      subroutine nd2pl(wanx,wany,wanz,wdx,wdy,wdz,phi,delta,alam,dipdir
     1,ierr)
c
c     compute strike, dip, rake and dip directions from Cartesian
c     components of the outward normal and slip vectors
c
c     usage:
c     call nd2pl(anx,any,anz,dx,dy,dz,strike,dip,rake,dipdir,ierr)
c
c     arguments:
c     anx,any,anz    components of fault plane outward normal vector in
the
c                    Aki-Richards Cartesian coordinate system (INPUT)
c     dx,dy,dz       components of slip vector in the Aki-Richards
c                    Cartesian coordinate system (INPUT)
c     strike         strike angle in degrees (OUTPUT)
c     dip            dip angle in degrees (OUTPUT)
c     rake           rake angle in degrees (OUTPUT)
c     dipdir         dip direction angle in degrees (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
```

```
c     1              input vectors not perpendicular among each other
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real wanx,wany,wanz,wdx,wdy,wdz,ang,anorm,anx,any,anz,dnorm
     1,dx,dy,dz,wdelta,wphi,walam,phi,delta,alam,dipdir
      integer ierr
c
      call fpsset
c
      ierr=0
      call angle(wanx,wany,wanz,wdx,wdy,wdz,ang)
      if(abs(ang-c90).gt.orttol) then
         write(io,'(1x,a,g15.7,a)') 'ND2PL: input vectors not '
     1   //'perpendicular, angle=',ang
         ierr=1
      endif
      call norm(wanx,wany,wanz,anorm,anx,any,anz)
      call norm(wdx,wdy,wdz,dnorm,dx,dy,dz)
      if(anz.gt.c0) then
         call invert(anx,any,anz)
         call invert(dx,dy,dz)
      endif
c
      if(anz.eq.-c1) then
         wdelta=c0
         wphi=c0
         walam=atan2(-dy,dx)
      else
         wdelta=acos(-anz)
         wphi=atan2(-anx,any)
         walam=atan2(-dz/sin(wdelta),dx*cos(wphi)+dy*sin(wphi))
      endif
      phi=wphi/dtor
      delta=wdelta/dtor
      alam=walam/dtor
      phi=mod(phi+c360,c360)
      dipdir=phi+c90
      dipdir=mod(dipdir+c360,c360)
      return
      end
***********************************************************************
*******
      subroutine ax2ca(trend,plunge,ax,ay,az,ierr)
c
c     compute cartesian components from trend and plunge
c
c     usage:
c     call ax2ca(trend,plunge,ax,ay,az,ierr)
c
c     arguments:
```

```
c     trend          clockwise angle from North in degrees (INPUT)
c     plunge         inclination angle in degrees (INPUT)
c     ax,ay,az       components of the axis direction downward versor in
the
c                    Aki-Richards Cartesian coordinate system (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input TREND angle out of range
c     2              input PLUNGE angle out of range
c     3              1+2
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real ax,ay,az,trend,plunge
      integer ierr
c
      call fpsset
c
      ax=c0
      ay=c0
      az=c0
      ierr=0
      if(trend.lt.amitre.or.trend.gt.amatre) then
         write(io,'(1x,a,g10.4,a)') 'AX2CA: input TREND angle ',trend,
     1   ' out of range'
         ierr=1
      endif
      if(plunge.lt.amiplu.or.plunge.gt.amaplu) then
         if(plunge.lt.amaplu.and.plunge.gt.-ovrtol) then
            plunge=amiplu
         else if(plunge.gt.amiplu.and.plunge-amaplu.lt.ovrtol) then
            plunge=amaplu
         else
            write(io,'(1x,a,g10.4,a)') 'AX2CA: input PLUNGE angle ',
     1      plunge,' out of range'
            ierr=ierr+2
         endif
      endif
      if(ierr.ne.0) return
      ax=cos(plunge*dtor)*cos(trend*dtor)
      ay=cos(plunge*dtor)*sin(trend*dtor)
      az=sin(plunge*dtor)
      return
      end
**********************************************************************
*******
      subroutine ca2ax(wax,way,waz,trend,plunge,ierr)
c
c     compute trend and plunge from Cartesian components
c
```

```
c     usage:
c     call ca2ax(ax,ay,az,trend,plunge,ierr)
c
c     arguments:
c     ax,ay,az       components of axis direction vector in the Aki-
Richards
c                    Cartesian coordinate system (INPUT)
c     trend          clockwise angle from North in degrees (OUTPUT)
c     plunge         inclination angle in degrees (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real wax,way,waz,wnorm,ax,ay,az,trend,plunge
      integer ierr
c
      call fpsset
c
      ierr=0
      call norm(wax,way,waz,wnorm,ax,ay,az)
      if(az.lt.c0) call invert(ax,ay,az)
      if(ay.ne.c0.or.ax.ne.c0) then
         trend=atan2(ay,ax)/dtor
      else
         trend=c0
      endif
      trend=mod(trend+c360,c360)
      plunge=asin(az)/dtor
      return
      end
**********************************************************************
*******
      subroutine pt2nd(wpx,wpy,wpz,wtx,wty,wtz,anx,any,anz,dx,dy,dz
     1,ierr)
c
c     compute Cartesian component of P and T versors
c     from outward normal and slip vectors
c
c     usage:
c     call pt2nd(px,py,pz,tx,ty,tz,anx,any,anz,dx,dy,dz,ierr)
c
c     arguments:
c     px,py,pz       components of P (maximum dilatation) axis vector
c                    in the Aki-Richards Cartesian coordinate system
(INPUT)
c     tx,ty,tz       components of T (maximum tension) axis vector
c                    in the Aki-Richards Cartesian coordinate system
(INPUT)
```

```
c     anx,any,anz    components of fault plane outward normal versor in
the
c                    Aki-Richards Cartesian coordinate system (OUTPUT)
c     dx,dy,dz       components of slip versor in the Aki-Richards
c                    Cartesian coordinate system (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input vectors not perpendicular among each other
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real anx,any,anz,dx,dy,dz,wpx,wpy,wpz,wtx,wty,wtz,ang,pnorm
     1,px,py,pz,tnorm,tx,ty,tz,amn
      integer ierr
c
      call fpsset
c
      anx=c0
      any=c0
      anz=c0
      dx=c0
      dy=c0
      dz=c0
      ierr=0
      call angle(wpx,wpy,wpz,wtx,wty,wtz,ang)
      if(abs(ang-c90).gt.orttol) then
         write(io,'(1x,a,g15.7,a)') 'PT2ND: input vectors not '
     1   //'perpendicular, angle=',ang
         ierr=1
      endif
      call norm(wpx,wpy,wpz,pnorm,px,py,pz)
      if(pz.lt.c0) call invert(px,py,pz)
      call norm(wtx,wty,wtz,tnorm,tx,ty,tz)
      if(tz.lt.c0) call invert(tx,ty,tz)
      anx=tx+px
      any=ty+py
      anz=tz+pz
      call norm(anx,any,anz,amn,anx,any,anz)
c
      dx=tx-px
      dy=ty-py
      dz=tz-pz
      call norm(dx,dy,dz,amn,dx,dy,dz)
      if(anz.gt.c0) then
         call invert(anx,any,anz)
         call invert(dx,dy,dz)
      endif
      return
      end
```

```
*********************************************************************
*******
      subroutine nd2pt(wanx,wany,wanz,wdx,wdy,wdz,px,py,pz,tx,ty,tz,bx
     1,by,bz,ierr)
c
c     compute Cartesian component of P, T and B axes from outward normal
c     and slip vectors
c
c     usage:
c     call nd2pt(anx,any,anz,dx,dy,dz,px,py,pz,tx,ty,tz,bx,by,bz,ierr)
c
c     arguments:
c     anx,any,anz    components of fault plane outward normal vector in
the
c                    Aki-Richards Cartesian coordinate system (INPUT)
c     dx,dy,dz       components of slip vector in the Aki-Richards
c                    Cartesian coordinate system (INPUT)
c     px,py,pz       components of downward P (maximum dilatation) axis
versor
c                    in the Aki-Richards Cartesian coordinate system
(OUTPUT)
c     tx,ty,tz       components of downward T (maximum tension) axis
versor
c                    in the Aki-Richards Cartesian coordinate system
(OUTPUT)
c     bx,by,bz       components of downward B (neutral) axis versor in
the
c                    Aki-Richards Cartesian coordinate system (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input vectors not perpendicular among each other
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real wanx,wany,wanz,amn,anx,any,anz,wdx,wdy,wdz,amd,dx,dy,dz
     1,ang,px,py,pz,tx,ty,tz,bx,by,bz,amp
      integer ierr
c
      call fpsset
c
      ierr=0
      call norm(wanx,wany,wanz,amn,anx,any,anz)
      call norm(wdx,wdy,wdz,amd,dx,dy,dz)
      call angle(anx,any,anz,dx,dy,dz,ang)
      if(abs(ang-c90).gt.orttol) then
         write(io,'(1x,a,g15.7,a)') 'ND2PT: input vectors not '
     1   //'perpendicular, angle=',ang
         ierr=1
      endif
      px=anx-dx
```

```
      py=any-dy
      pz=anz-dz
      call norm(px,py,pz,amp,px,py,pz)
      if(pz.lt.c0) call invert(px,py,pz)
      tx=anx+dx
      ty=any+dy
      tz=anz+dz
      call norm(tx,ty,tz,amp,tx,ty,tz)
      if(tz.lt.c0) call invert(tx,ty,tz)
      call vecpro(px,py,pz,tx,ty,tz,bx,by,bz)
      if(bz.lt.c0) call invert(bx,by,bz)
      return
      end
*********************************************************************
*******
      subroutine ar2pt(am,am0,am1,e,am0b,px,py,pz,tx,ty,tz,bx,by,bz
     1,eta,ierr)
c
c     compute Cartesian components of deformation axes (P, T and B)
c     from moment tensor (Aki & Richards convention)
c
c     usage:
c     call ar2pt(am,am0,am1,e,am0b,px,py,pz,tx,ty,tz,bx,by,bz,eta,ierr)
c
c     arguments:
c     am              seismic moment tensor (3x3 matrix) (INPUT)
c     am0             scalar seismic moment of principal double couple
c                     (OUTPUT)
c     am1             scalar seismic moment of secondary double couple
c                     (OUTPUT)
c     e               isotropic component (trace of the input tensor)
c                     (OUTPUT)
c     am0b            scalar seismic moment of best double couple (OUTPUT)
c     px,py,pz        components of downward P (maximum dilatation) axis
versor
c                     in the Aki-Richards Cartesian coordinate system
(OUTPUT)
c     tx,ty,tz        components of downward T (maximum tension) axis
versor
c                     in the Aki-Richards Cartesian coordinate system
(OUTPUT)
c     bx,by,bz        components of downward B (neutral) axis versor in
the
c                     Aki-Richards Cartesian coordinate system (OUTPUT)
c     eta             percentage of CLVD remainder (OUTPUT)
c     ierr            error indicator (OUTPUT)
c
c     errors:
c     1               input tensor not symmetrical: am(1,2).ne.am(2,1)
c     2               input tensor not symmetrical: am(1,3).ne.am(3,1)
c     3               input tensor not symmetrical: am(2,3).ne.am(3,2)
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
```

```
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,val,vec,am0,am1,e,px,py,pz,tx,ty,tz,bx,by,bz,am0b,eta,dum
      integer ierr,i,j,k
      dimension am(3,3),val(3),vec(3,3)
c
      call fpsset
c
      am0=c0
      am1=c0
      e=c0
      am0b=c0
      px=c0
      py=c0
      pz=c0
      tx=c0
      ty=c0
      tz=c0
      bx=c0
      by=c0
      bz=c0
      ierr=0
      if(abs(am(1,2)-am(2,1)).gt.tentol) then
         write(io,'(1x,a,g10.4,a,g10.4)') 'AR2PT: input tensor not'
     1   //' symmetrical, m(1,2)=',am(1,2),' m(2,1)=',am(2,1)
         ierr=1
      endif
      if(abs(am(1,3)-am(3,1)).gt.tentol) then
         write(io,'(1x,a,g10.4,a,g10.4)') 'AR2PT: input tensor not'
     1   //' symmetrical, m(1,3)=',am(1,3),' m(3,1)=',am(3,1)
         ierr=ierr+2
      endif
      if(abs(am(3,2)-am(2,3)).gt.tentol) then
         write(io,'(1x,a,g10.4,a,g10.4)') 'AR2PT: input tensor not'
     1   //' symmetrical, m(2,3)=',am(2,3),' m(3,2)=',am(3,2)
         ierr=ierr+4
      endif
      if(ierr.ne.0) return
      call avec(am,val,vec)
      e=(val(1)+val(2)+val(3))/c3
c
c     compute deviatoric eigevalues
c
      do i=1,3
         val(i)=val(i)-e
      enddo
c
c     sort deviatoric eigenvalues (with isotropic component removed)
c     and eigenvectors by inverse order of eigenvalue modulus magnitude
c
1     do 2 i=1,2
         do 3 j=i+1,3
            if(abs(val(i)).lt.abs(val(j))) then
               dum=val(i)
               val(i)=val(j)
               val(j)=dum
               do 4 k=1,3
                  dum=vec(k,i)
                  vec(k,i)=vec(k,j)
```

```
                  vec(k,j)=dum
4              continue
            endif
3        continue
2     continue
      am0=val(1)
      eta=-val(3)/(c2*am0)
      am1=abs(val(3))
      am0b=(abs(val(1))+abs(val(2)))/c2
      if(am0.lt.c0) then
         am0=-am0
         tx=vec(1,2)
         ty=vec(2,2)
         tz=vec(3,2)
         px=vec(1,1)
         py=vec(2,1)
         pz=vec(3,1)
         bx=vec(1,3)
         by=vec(2,3)
         bz=vec(3,3)
      else
         tx=vec(1,1)
         ty=vec(2,1)
         tz=vec(3,1)
         px=vec(1,2)
         py=vec(2,2)
         pz=vec(3,2)
         bx=vec(1,3)
         by=vec(2,3)
         bz=vec(3,3)
      endif
      return
      end
*************************************************************************
*******
      subroutine nd2ar(anx,any,anz,dx,dy,dz,am0,am,ierr)
c
c     compute moment tensor Cartesian components (Aki & Richards
convention)
c     from outward normal and slip vectors Cartesian components
c
c     usage:
c     call nd2ar(anx,any,anz,dx,dy,dz,am0,am,ierr)
c
c     arguments:
c     anx,any,anz    components of fault plane outward normal vector in
the
c                    Aki-Richards Cartesian coordinate system (INPUT)
c     dx,dy,dz       components of slip vector in the Aki-Richards
c                    Cartesian coordinate system (INPUT)
c     am0            scalar seismic moment  (INPPUT)
c     am             seismic moment tensor (3x3 matrix) (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input vectors not perpendicular among each other
c
c      implicit none
c-----------------------------------------------------------------------
-------
```

```
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real anx,any,anz,dx,dy,dz,am0,am,aam0,ang,anorm,wanx,wany,wanz
     1,wdx,wdy,wdz
      integer ierr,i,j
      dimension am(3,3)
c
      call fpsset
c
      do 1 i=1,3
         do  2 j=1,3
            am(i,j)=c0
2        continue
1     continue
      if(am0.eq.c0) then
         aam0=c1
      else
         aam0=am0
      endif
      ierr=0
      call angle(anx,any,anz,dx,dy,dz,ang)
      if(abs(ang-c90).gt.orttol) then
         write(io,'(1x,a,g15.7,a)') 'ND2AR: input vectors not '
     1   //'perpendicular, angle=',ang
         ierr=1
      endif
      call norm(anx,any,anz,anorm,wanx,wany,wanz)
      call norm(dx,dy,dz,anorm,wdx,wdy,wdz)
      am(1,1)=aam0*c2*wdx*wanx
      am(1,2)=aam0*(wdx*wany+wdy*wanx)
      am(2,1)=am(1,2)
      am(1,3)=aam0*(wdx*wanz+wdz*wanx)
      am(3,1)=am(1,3)
      am(2,2)=aam0*c2*wdy*wany
      am(2,3)=aam0*(wdy*wanz+wdz*wany)
      am(3,2)=am(2,3)
      am(3,3)=aam0*c2*wdz*wanz
      return
      end
***********************************************************************
*******
      subroutine ar2ha(am,amo,ierr)
c
c     transforms moment tensor component from Aki Richards to Harvard CMT
c     reference systems and viceversa
c
c     usage:
c     call ar2ha(am,amo,ierr)
c
c     arguments:
c     am             input seismic moment tensor (3x3 matrix) (INPUT)
c     amo            output seismic moment tensor (3x3 matrix) (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
```

```
c     errors:
c     1              input tensor not symmetrical: am(1,2).ne.am(2,1)
c     2              input tensor not symmetrical: am(1,3).ne.am(3,1)
c     3              input tensor not symmetrical: am(2,3).ne.am(3,2)
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,amo
      integer ierr,i,j
      dimension am(3,3),amo(3,3)
c
      call fpsset
c
      ierr=0
      if(abs(am(1,2)-am(2,1)).gt.tentol) then
         write(io,'(1x,a,g10.4,a,g10.4)') 'AR2HA: input tensor not'
     1   //' symmetrical, m(1,2)=',am(1,2),' m(2,1)=',am(2,1)
         ierr=1
      endif
      if(abs(am(1,3)-am(3,1)).gt.tentol) then
         write(io,'(1x,a,g10.4,a,g10.4)') 'AR2HA: input tensor not'
     1   //' symmetrical, m(1,3)=',am(1,3),' m(3,1)=',am(3,1)
         ierr=ierr+2
      endif
      if(abs(am(3,2)-am(2,3)).gt.tentol) then
         write(io,'(1x,a,g10.4,a,g10.4)') 'AR2HA: input tensor not'
     1   //' symmetrical, m(2,3)=',am(2,3),' m(3,2)=',am(3,2)
         ierr=ierr+4
      endif
      if(ierr.ne.0) then
         do 1 i=1,3
            do 2 j=1,3
               amo(i,j)=c0
2           continue
1        continue
         return
      endif
      amo(1,1)=am(1,1)
      amo(1,2)=-am(1,2)
      amo(1,3)=am(1,3)
      amo(2,1)=-am(2,1)
      amo(2,2)=am(2,2)
      amo(2,3)=-am(2,3)
      amo(3,1)=am(3,1)
      amo(3,2)=-am(3,2)
      amo(3,3)=am(3,3)
      return
      end
*********************************************************************
*******
*     COMPOSITE ROUTINES
```

```
*********************************************************************
*******
      subroutine nd2ha(anx,any,anz,dx,dy,dz,am0,am,ierr)
c
c     compute moment tensor Cartesian components (Harvard CMT convention)
c     from outward normal and slip vectors Cartesian components
c
c     usage:
c     call nd2ha(anx,any,anz,dx,dy,dz,am0,am,ierr)
c
c     arguments:
c     anx,any,anz    components of fault plane outward normal vector in
the
c                    Aki-Richards Cartesian coordinate system (INPUT)
c     dx,dy,dz       components of slip vector in the Aki-Richards
c                    Cartesian coordinate system (INPUT)
c     am0            scalar seismic moment  (INPPUT)
c     am             seismic moment tensor (3x3 matrix) (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input vectors not perpendicular among each other
c     2              internal error
c
c      implicit none
c---------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c---------------------------------------------------------------------
-------
      real am,anx,any,anz,dx,dy,dz,am0
      integer ierr,i,j
      dimension am(3,3)
c
      call fpsset
c
      do 1 i=1,3
         do 2 j=1,3
            am(i,j)=c0
2        continue
1     continue
      ierr=0
      call nd2ar(anx,any,anz,dx,dy,dz,am0,am,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'ND2HA: ierr=',ierr
         return
      endif
      call ar2ha(am,am,ierr)
      if(ierr.ne.0) then
         ierr=2
         write(io,'(1x,a,i3)') 'ND2HA: ierr=',ierr
      endif
      return
      end
```

```
*********************************************************************
*******
      subroutine pl2pl(strika,dipa,rakea,strikb,dipb,rakeb,
     1dipdib,ierr)
c
c     compute strike, dip and rake of a nodal plane
c     from strike, dip and rake of the other one
c
c
c     usage:
c     call pl2pl(strika,dipa,rakea,strikb,dipb,rakeb,dipdib,ierr)
c
c     arguments:
c     strika         strike angle in degrees of the first nodal plane
(INPUT)
c     dipa           dip angle in degrees of the first nodal plane
(INPUT)
c     rakea          rake angle in degrees of the first nodal plane
(INPUT)
c     strikb         strike angle in degrees of the second nodal plane
(OUTPUT)
c     dipb           dip angle in degrees of the second nodal plane
(OUTPUT)
c     rakeb          rake angle in degrees of the second nodal plane
(OUTPUT)
c     dipdib         dip direction in degrees of the second nodal plane
(OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input STRIKE angle out of range
c     2              input DIP angle out of range
c     4              input RAKE angle out of range
c     3              1+2
c     5              1+4
c     7              1+2+4
c     8              internal error
c
c      implicit none
c---------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c---------------------------------------------------------------------
-------
      real strika,dipa,rakea,anx,any,anz,dx,dy,dz,strikb,dipb,rakeb,
     1dipdib
      integer ierr
c
      call fpsset
c
      call pl2nd(strika,dipa,rakea,anx,any,anz,dx,dy,dz,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'PL2PL: ierr=',ierr
         return
      endif
```

```
      call nd2pl(dx,dy,dz,anx,any,anz,strikb,dipb,rakeb,dipdib,ierr)
      if(ierr.ne.0) then
         ierr=8
         write(io,'(1x,a,i3)') 'PL2PL: ierr=',ierr
      endif
      return
      end
*********************************************************************
*******
      subroutine pl2pt(strike,dip,rake,trendp,plungp,trendt,plungt,
     1trendb,plungb,ierr)
c
c     compute trend and plunge of P, T and B axes
c     from strike, dip and rake of a nodal plane
c
c
c     usage:
c     call
pl2pt(strike,dip,rake,trendp,plungp,trendt,plungt,trendb,plungb,ierr)
c
c     arguments:
c     strike         strike angle in degrees of the first nodal plane
(INPUT)
c     dip            dip angle in degrees of the first nodal plane
(INPUT)
c     rake           rake angle in degrees of the first nodal plane
(INPUT)
c     trendp         trend of P axis (OUTPUT)
c     plungp         plunge or P axis (OUTPUT)
c     trendt         trend of T axis (OUTPUT)
c     plungt         plunge or T axis (OUTPUT)
c     trendb         trend of B axis (OUTPUT)
c     plungb         plunge or B axis (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input STRIKE angle out of range
c     2              input DIP angle out of range
c     4              input RAKE angle out of range
c     3              1+2
c     5              1+4
c     7              1+2+4
c     8,9,10,11      internal error
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real strike,dip,rake,anx,any,anz,dx,dy,dz,px,py,pz,tx,ty,tz
     1,bx,by,bz,trendp,plungp,trendt,plungt,trendb,plungb
      integer ierr
c
      call fpsset
```

```
c
      call pl2nd(strike,dip,rake,anx,any,anz,dx,dy,dz,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'PL2PT: ierr=',ierr
         return
      endif
      call nd2pt(dx,dy,dz,anx,any,anz,px,py,pz,tx,ty,tz,bx,by,bz,ierr)
      if(ierr.ne.0) then
         ierr=8
         write(io,'(1x,a,i3)') 'PL2PT: ierr=',ierr
      endif
      call ca2ax(px,py,pz,trendp,plungp,ierr)
      if(ierr.ne.0) then
         ierr=9
         write(io,'(1x,a,i3)') 'PL2PT: ierr=',ierr
      endif
      call ca2ax(tx,ty,tz,trendt,plungt,ierr)
      if(ierr.ne.0) then
         ierr=10
         write(io,'(1x,a,i3)') 'PL2PT: ierr=',ierr
      endif
      call ca2ax(bx,by,bz,trendb,plungb,ierr)
      if(ierr.ne.0) then
         ierr=11
         write(io,'(1x,a,i3)') 'PL2PT: ierr=',ierr
      endif
      return
      end
************************************************************************
*******
      subroutine pt2pl(trendp,plungp,trendt,plungt,strika,dipa,rakea
     1,dipdia,strikb,dipb,rakeb,dipdib,ierr)
c
c     compute strike dip and rake (and dip direction) of two nodal planes
c     from trend and plung of P and T axes
c
c     usage:
c     call pt2pl(trendp,plungp,trendt,plungt,strika,dipa,rakea
c    1,dipdia,strikb,dipb,rakeb,dipdib,ierr)
c
c     arguments:
c     trendp         trend of P axis in degrees (INPUT)
c     plungp         plunge of P axis in degrees (INPUT)
c     trendt         trend of P axis in degrees (INPUT)
c     plungt         plunge of P axis in degrees (INPUT)
c     strika         strike angle of first nodal plane in degrees
(OUTPUT)
c     dipa           dip angle of first nodal plane in degrees (OUTPUT)
c     rakea          rake angle of first nodal plane in degrees (OUTPUT)
c     dipdia         dip direction angle of first nodal plane in degrees
(OUTPUT)
c     strikb         strike angle of second nodal plane in degrees
(OUTPUT)
c     dipb           dip angle of second nodal plane in degrees (OUTPUT)
c     rakeb          rake angle of second nodal plane in degrees (OUTPUT)
c     dipdib         dip direction angle of second nodal plane in degrees
(OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
```

```
c     1              input TREND angle of P axis out of range
c     2              input PLUNGE angle P axis out of range
c     3              1+2
c     4              input TREND angle of P axis out of range
c     5              input PLUNGE angle P axis out of range
c     6              4+5
c     8,9,10         internal errors
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real trendp,plungp,trendt,plungt,px,py,pz,tx,ty,tz
     1,anx,any,anz,dx,dy,dz,strika,dipa,rakea,dipdia,strikb,dipb
     1,rakeb,dipdib
      integer ierr
c
      call fpsset
c
      call ax2ca(trendp,plungp,px,py,pz,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'PT2PL: ierr=',ierr
         return
      endif
      call ax2ca(trendt,plungt,tx,ty,tz,ierr)
      if(ierr.ne.0) then
         ierr=ierr+3
         write(io,'(1x,a,i3)') 'PT2PL: ierr=',ierr
         return
      endif
      call pt2nd(px,py,pz,tx,ty,tz,anx,any,anz,dx,dy,dz,ierr)
      if(ierr.ne.0) then
         ierr=8
         write(io,'(1x,a,i3)') 'PT2PL: ierr=',ierr
         return
      endif
      call nd2pl(anx,any,anz,dx,dy,dz,strika,dipa,rakea,dipdia,ierr)
      if(ierr.ne.0) then
         ierr=9
         write(io,'(1x,a,i3)') 'PT2PL: ierr=',ierr
         return
      endif
      call nd2pl(dx,dy,dz,anx,any,anz,strikb,dipb,rakeb,dipdib,ierr)
      if(ierr.ne.0) then
         ierr=10
         write(io,'(1x,a,i3)') 'PT2PL: ierr=',ierr
         return
      endif
      return
      end
c
*********************************************************************
*******
```

```
      subroutine ar2plp(am,am0,am1,e,am0b,phia,deltaa,alama,slipa,
     1phib,deltab,alamb,slipb,trendp,plungp,trendt,plungt,trendb,
     2plungb,eta,ierr)
c
c     compute planes and axes of principal double couple
c     from moment tensor (Aki & Richards convention)
c
c     usage:
c     call ar2plp(am,am0,am1,e,am0b,phia,deltaa,alama,slipa,
c    1phib,deltab,alamb,slipb,trendp,plungp,trendt,plungt,trendb,
c    2plungb,eta,ierr)
c
c     arguments:
c     am              seismic moment tensor (3x3 matrix) (INPUT)
c     am0             scalar seismic moment of principal double couple
c                     (OUTPUT)
c     am1             scalar seismic moment of secondary double couple
c                     (OUTPUT)
c     e               isotropic component (trace of the input tensor)
c                     (OUTPUT)
c     am0b            scalar seismic moment of best double couple (OUTPUT)
c     phia            strike angle in degrees of the first nodal plane
(OUTPUT)
c     deltaa          dip angle in degrees of the first nodal plane
(OUTPUT)
c     alama           rake angle in degrees of the first nodal plane
(OUTPUT)
c     slipa           dip direction in degrees of the first nodal plane
(OUTPUT)
c     phib            strike angle in degrees of the second nodal plane
(OUTPUT)
c     deltab          dip angle in degrees of the second nodal plane
(OUTPUT)
c     alamb           rake angle in degrees of the second nodal plane
(OUTPUT)
c     slipb           dip direction in degrees of the second nodal plane
(OUTPUT)
c     trendp          trend angle in degrees of the P axis (OUTPUT)
c     plungp          plunge angle in degrees of the P axis (OUTPUT)
c     trendt          trend angle in degrees of the T axis (OUTPUT)
c     plungt          plunge angle in degrees of the T axis (OUTPUT)
c     trendb          trend angle in degrees of the B axis (OUTPUT)
c     plungb          plunge angle in degrees of the B axis (OUTPUT)
c     eta             percentage of CLVD remainder (OUTPUT)
c     ierr            error indicator (OUTPUT)
c
c     errors:
c     1               input tensor not symmetrical: am(1,2).ne.am(2,1)
c     2               input tensor not symmetrical: am(1,3).ne.am(3,1)
c     3               input tensor not symmetrical: am(2,3).ne.am(3,2)
c     5,6,7,8,9,10    internal errors
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
```

```
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,am0,am1,e,am0b,phia,deltaa,alama,slipa,phib,deltab
     1,alamb,slipb,trendp,plungp,trendt,plungt,trendb,plungb
     2,px,py,pz,tx,ty,tz,anx,any,anz,dx,dy,dz,bx,by,bz,eta
      integer ierr
      dimension am(3,3)
c
      call fpsset
c
      am0=c0
      am1=c0
      e=c0
      am0b=c0
      phia=c0
      deltaa=c0
      alama=c0
      slipa=c0
      phib=c0
      deltab=c0
      alamb=c0
      slipb=c0
      trendp=c0
      plungp=c0
      trendt=c0
      plungt=c0
      trendb=c0
      plungb=c0
      ierr=0
      call ar2pt(am,am0,am1,e,am0b,px,py,pz,tx,ty,tz,bx,by,bz,eta,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'AR2PLP: ierr=',ierr
         return
      endif
      call ca2ax(px,py,pz,trendp,plungp,ierr)
      if(ierr.ne.0) then
         ierr=5
         write(io,'(1x,a,i3)') 'AR2PLP: ierr=',ierr
         return
      endif
      call ca2ax(tx,ty,tz,trendt,plungt,ierr)
      if(ierr.ne.0) then
         ierr=6
         write(io,'(1x,a,i3)') 'AR2PLP: ierr=',ierr
         return
      endif
      call ca2ax(bx,by,bz,trendb,plungb,ierr)
      if(ierr.ne.0) then
         ierr=7
         write(io,'(1x,a,i3)') 'AR2PLP: ierr=',ierr
         return
      endif
      call pt2nd(px,py,pz,tx,ty,tz,anx,any,anz,dx,dy,dz,ierr)
      if(ierr.ne.0) then
         ierr=8
         write(io,'(1x,a,i3)') 'AR2PLP: ierr=',ierr
         return
      endif
      call nd2pl(anx,any,anz,dx,dy,dz,phia,deltaa,alama,slipa,ierr)
```

```
      if(ierr.ne.0) then
         ierr=9
         write(io,'(1x,a,i3)') 'AR2PLP: ierr=',ierr
         return
      endif
      call nd2pl(dx,dy,dz,anx,any,anz,phib,deltab,alamb,slipb,ierr)
      if(ierr.ne.0) then
         ierr=10
         write(io,'(1x,a,i3)') 'AR2PLP: ierr=',ierr
         return
      endif
      return
      end
************************************************************************
*******
      subroutine ha2plp(am,am0,am1,e,am0b,strika,dipa,rakea,slipa,
     1strikb,dipb,rakeb,slipb,trendp,plungp,trendt,plungt,trendb,
     2plungb,eta,ierr)
c
c     compute planes and axes of principal double couple
c     from moment tensor (Harvard CMT convention)
c
c     usage:
c      subroutine ha2plp(am,am0,am1,e,am0b,strika,dipa,rakea,slipa,
c     1strikb,dipb,rakeb,slipb,trendp,plungp,trendt,plungt,trendb,
c     2plungb,eta,ierr)
c
c     arguments:
c     am              seismic moment tensor (3x3 matrix) (INPUT)
c     am0             scalar seismic moment of principal double couple
c                     (OUTPUT)
c     am1             scalar seismic moment of secondary double couple
c                     (OUTPUT)
c     e               isotropic component (trace of the input tensor)
c                     (OUTPUT)
c     am0b            scalar seismic moment of best double couple (OUTPUT)
c     strika          strike angle in degrees of the first nodal plane
(OUTPUT)
c     dipa            dip angle in degrees of the first nodal plane
(OUTPUT)
c     rakea           rake angle in degrees of the first nodal plane
(OUTPUT)
c     slipa           dip direction in degrees of the first nodal plane
(OUTPUT)
c     strikb          strike angle in degrees of the second nodal plane
(OUTPUT)
c     dipb            dip angle in degrees of the second nodal plane
(OUTPUT)
c     rakeb           rake angle in degrees of the second nodal plane
(OUTPUT)
c     slipb           dip direction in degrees of the second nodal plane
(OUTPUT)
c     trendp          trend angle in degrees of the P axis (OUTPUT)
c     plungp          plunge angle in degrees of the P axis (OUTPUT)
c     trendt          trend angle in degrees of the T axis (OUTPUT)
c     plungt          plunge angle in degrees of the T axis (OUTPUT)
c     trendb          trend angle in degrees of the B axis (OUTPUT)
c     plungb          plunge angle in degrees of the B axis (OUTPUT)
c     eta             percentage of CLVD remainder (OUTPUT)
c     ierr            error indicator (OUTPUT)
```

```
c
c     errors:
c     1              input tensor not symmetrical: am(1,2).ne.am(2,1)
c     2              input tensor not symmetrical: am(1,3).ne.am(3,1)
c     3              input tensor not symmetrical: am(2,3).ne.am(3,2)
c     5,6,7,8,9,10   internal errors
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,ama,am0,am1,e,am0b,strika,dipa,rakea,slipa,strikb,dipb
     1,rakeb,slipb,trendp,plungp,trendt,plungt,trendb,plungb,eta
      integer ierr
      dimension am(3,3),ama(3,3)
c
      call fpsset
c
      am0=c0
      am1=c0
      e=c0
      am0b=c0
      strika=c0
      dipa=c0
      rakea=c0
      slipa=c0
      strikb=c0
      dipb=c0
      rakeb=c0
      slipb=c0
      trendp=c0
      plungp=c0
      trendt=c0
      plungt=c0
      trendb=c0
      plungb=c0
      ierr=0
      call ar2ha(am,ama,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'HA2PLP: ierr=',ierr
         return
      endif
      call ar2plp(ama,am0,am1,e,am0b,strika,dipa,rakea,slipa,
     1strikb,dipb,rakeb,slipb,trendp,plungp,trendt,plungt,trendb,
     2plungb,eta,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'HA2PLP: ierr=',ierr
      endif
      return
      end
**********************************************************************
*******
      subroutine pl2ar(strike,dip,rake,am0,am,ierr)
```

```
c
c     compute moment tensor Cartesian components (Aki & Richards
convention)
c     from strike, dip and rake
c
c     usage:
c     call pl2ar(strike,dip,rake,am0,am,ierr)
c
c     arguments:
c     strike         strike angle in degrees  (INPUT)
c     dip            dip angle in degrees (INPUT)
c     rake           rake angle in degrees (INPUT)
c     am0            scalar seismic moment  (INPUT)
c     am             seismic moment tensor (3x3 matrix) (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input STRIKE angle out of range
c     2              input DIP angle out of range
c     4              input RAKE angle out of range
c     3              1+2
c     5              1+4
c     7              1+2+4
c     8              internal error
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,anx,any,anz,dx,dy,dz,am0,strike,dip,rake
      integer ierr,i,j
      dimension am(3,3)
c
      call fpsset
c
      do 1 i=1,3
         do 2 j=1,3
            am(i,j)=c0
2        continue
1     continue
      ierr=0
      call pl2nd(strike,dip,rake,anx,any,anz,dx,dy,dz,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'PL2AR: ierr=',ierr
         return
      endif
      call nd2ar(anx,any,anz,dx,dy,dz,am0,am,ierr)
      if(ierr.ne.0) then
         ierr=8
         write(io,'(1x,a,i3)') 'PL2AR: ierr=',ierr
      endif
      return
      end
```

```
*********************************************************************
*******
      subroutine pl2ha(strike,dip,rake,am0,am,ierr)
c
c     compute moment tensor Cartesian components (Harvard CMT convention)
c     from strike, dip and rake
c
c     usage:
c     call pl2ha(strike,dip,rake,am0,am,ierr)
c
c     arguments:
c     strike         strike angle in degrees  (INPUT)
c     dip            dip angle in degrees (INPUT)
c     rake           rake angle in degrees (INPUT)
c     am0            scalar seismic moment  (INPUT)
c     am             seismic moment tensor (3x3 matrix) (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input STRIKE angle out of range
c     2              input DIP angle out of range
c     4              input RAKE angle out of range
c     3              1+2
c     5              1+4
c     7              1+2+4
c     8,9            internal errors
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real strike,dip,rake,am0,am
      integer ierr,i,j
      dimension am(3,3)
c
      call fpsset
c
      do 1 i=1,3
         do 2 j=1,3
            am(i,j)=c0
2        continue
1     continue
      ierr=0
      call pl2ar(strike,dip,rake,am0,am,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'PL2HA: ierr=',ierr
         return
      endif
      call ar2ha(am,am,ierr)
      if(ierr.ne.0) then
         ierr=9
         write(io,'(1x,a,i3)') 'PL2HA: ierr=',ierr
      endif
```

```
      return
      end
*********************************************************************
*******
      subroutine pt2ar(trendp,plungp,trendt,plungt,am0,am,ierr)
c
c     compute moment tensor Cartesian components (Aki & Richards
convention)
c     from P and T axes
c
c     usage:
c     call pt2ar(trendp,plungp,trendt,plungt,am0,am)
c
c     arguments:
c     trendp         trend angle of P axis in degrees (INPUT)
c     plungp         plunge angle of P axis in degrees (INPUT)
c     trendt         trend angle of T axis in degrees (INPUT)
c     plungt         plunge angle of T axis in degrees (INPUT)
c     am0            scalar seismic moment  (INPUT)
c     am             seismic moment tensor (3x3 matrix) (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input TREND angle of P axis out of range
c     2              input PLUNGE angle P axis out of range
c     3              1+2
c     4              input TREND angle of P axis out of range
c     5              input PLUNGE angle P axis out of range
c     6              4+5
c     8,9            internal errors
c
c      implicit none
c---------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c---------------------------------------------------------------------
-------
      real trendp,plungp,trendt,plungt,am0,am,px,py,pz,tx,ty,tz
     1,anx,any,anz,dx,dy,dz
      integer ierr,i,j
      dimension am(3,3)
c
      call fpsset
c
      do 1 i=1,3
         do 2 j=1,3
            am(i,j)=c0
2        continue
1     continue
      ierr=0
      call ax2ca(trendp,plungp,px,py,pz,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'PT2AR: ierr=',ierr
         return
      endif
```

```
      call ax2ca(trendt,plungt,tx,ty,tz,ierr)
      if(ierr.ne.0) then
         ierr=ierr+3
         write(io,'(1x,a,i3)') 'PT2AR: ierr=',ierr
         return
      endif
      call pt2nd(px,py,pz,tx,ty,tz,anx,any,anz,dx,dy,dz,ierr)
      if(ierr.ne.0) then
         ierr=8
         write(io,'(1x,a,i3)') 'PT2AR: ierr=',ierr
         return
      endif
      call nd2ar(anx,any,anz,dx,dy,dz,am0,am,ierr)
      if(ierr.ne.0) then
         ierr=9
         write(io,'(1x,a,i3)') 'PT2AR: ierr=',ierr
      endif
      return
      end
*********************************************************************
*******
      subroutine pt2ha(trendp,plungp,trendt,plungt,am0,am,ierr)
c
c     compute moment tensor Cartesian components (Harvard CMT convention)
c     from P and T axes
c
c     usage:
c     call pt2ha(trendp,plungp,trendt,plungt,am0,am)
c
c     arguments:
c     trendp         trend angle of P axis in degrees (INPUT)
c     plungp         plunge angle of P axis in degrees (INPUT)
c     trendt         trend angle of T axis in degrees (INPUT)
c     plungt         plunge angle of T axis in degrees (INPUT)
c     am0            scalar seismic moment  (INPUT)
c     am             seismic moment tensor (3x3 matrix) (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input TREND angle of P axis out of range
c     2              input PLUNGE angle P axis out of range
c     3              1+2
c     4              input TREND angle of P axis out of range
c     5              input PLUNGE angle P axis out of range
c     6              4+5
c     8,9,10         internal errors
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real trendp,plungp,trendt,plungt,am0,am
      integer ierr,i,j
```

```
      dimension am(3,3)
c
      call fpsset
c
      do 1 i=1,3
         do 2 j=1,3
            am(i,j)=c0
2        continue
1     continue
      ierr=0
      call pt2ar(trendp,plungp,trendt,plungt,am0,am,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'PT2HA: ierr=',ierr
         return
      endif
      call ar2ha(am,am,ierr)
      if(ierr.ne.0) then
         ierr=10
         write(io,'(1x,a,i3)') 'PT2HA: ierr=',ierr
      endif
      return
      end
************************************************************************
*******
*     UTILITY ROUTINES
************************************************************************
*******
      subroutine avec(am,eval,evec)
c
c     compute eigenvalues and eigenvectors
c
c     usage:
c     utility routine for internal use only
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,eval,evec,dum
      integer i,j,k
      dimension am(3,3),eval(3),evec(3,3)
c
      call fpsset
c
      CALL EVCSF (3, AM, 3, EVAL, EVEC, 3)
1     do 2 i=1,2
         do 3 j=i+1,3
            if(abs(eval(i)).lt.abs(eval(j))) then
               dum=eval(i)
               eval(i)=eval(j)
               eval(j)=dum
               do 4 k=1,3
                  dum=evec(k,i)
```

```
                  evec(k,i)=evec(k,j)
                  evec(k,j)=dum
4              continue
            endif
3        continue
2     continue
      return
      end
*********************************************************************
*******
      subroutine angle(wax,way,waz,wbx,wby,wbz,ang)
c
c     compute the angle (in degrees) between two vectors
c
c     usage:
c     call angle(wax,way,waz,wbx,wby,wbz,ang)
c
c     arguments:
c     wax,way,waz    Cartesian component of first vector (INPUT)
c     wbx,wby,wbz    Cartesian component of second vector (INPUT)
c     ang            angle between the two vectors in degrees (OUTPUT)
c
c      implicit none
c--------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c--------------------------------------------------------------------
-------
      real wax,way,waz,wbx,wby,wbz,ax,ay,az,bx,by,bz,ang
     1,anorm,bnorm,prod
c
      call fpsset
c
      call norm(wax,way,waz,anorm,ax,ay,az)
      call norm(wbx,wby,wbz,bnorm,bx,by,bz)
      prod=ax*bx+ay*by+az*bz
      ang=acos(max(-c1,min(c1,prod)))/dtor
      return
      end
*********************************************************************
*******
      subroutine angles(strika,dipa,rakea,strikb,dipb,rakeb,
     1anglep,angled,ierr)
c
c     compute the angle (in degrees) between nodal planes and slip
vectors
c
c     usage:
c     call
angles(strikea,dipa,rakea,strikeb,dipb,rakeb,anglep,angled,ierr)
c
c     arguments:
c     strika         strike angle in degrees of the first nodal plane
(INPUT)
```

```
c     dipa           dip angle in degrees of the first nodal plane
(INPUT)
c     rakea          rake angle in degrees of the first nodal plane
(INPUT)
c     strikb         strike angle in degrees of the second nodal plane
(INPUT)
c     dipb           dip angle in degrees of the second nodal plane
(INPUT)
c     rakeb          rake angle in degrees of the second nodal plane
(INPUT)
c     anglep         angle in degrees between planes
c     angled         angle in degrees between slip directions
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input STRIKE angle of first plane out of range
c     2              input DIP angle of first plane out of range
c     4              input RAKE angle of first plane out of range
c     3              1+2
c     5              1+4
c     7              1+2+4
c     8              input STRIKE angle of first plane out of range
c     9              input DIP angle of first plane out of range
c     11             input RAKE angle of first plane out of range
c     10             8+9
c     13             8+11
c     15             8+9+11
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real strika,dipa,rakea,strikb,dipb,rakeb,anglep,angled
     1,anax,anay,anaz,anbx,anby,anbz,dax,day,daz,dbx,dby,dbz
      integer ierr
c
      call fpsset
c
      call pl2nd(strika,dipa,rakea,anax,anay,anaz,dax,day,daz,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'ANGLES: ierr=',ierr
         return
      endif
      call pl2nd(strikb,dipb,rakeb,anbx,anby,anbz,dbx,dby,dbz,ierr)
      if(ierr.ne.0) then
         ierr=ierr+8
         write(io,'(1x,a,i3)') 'ANGLES: ierr=',ierr
         return
      endif
      call angle(anax,anay,anaz,anbx,anby,anbz,anglep)
      call angle(dax,day,daz,dbx,dby,dbz,angled)
      return
      end
```

```
*********************************************************************
*******
      subroutine anglea(trenda,plunga,trendb,plungb,ang,ierr)
c
c     usage:
c     call anglea(trenda,plunga,trendb,plungb,ang,ierr)
c
c     arguments:
c     trenda         clockwise angle from North in degrees of first axis
(INPUT)
c     plunga         inclination angle in degrees of first axis (INPUT)
c     trendb         clockwise angle from North in degrees of second axis
(INPUT)
c     plungb         inclination angle in degrees of second axis (INPUT)
c     ang            angle in degrees between the axes (OUTPUT)
c     ierr           error indicator (OUTPUT)
c
c     errors:
c     1              input TREND angle of first axis out of range
c     2              input PLUNGE angle of first axis out of range
c     3              1+2
c     5              input TREND angle of first axis out of range
c     6              input PLUNGE angle of first axis out of range
c     7              5+6
c
c      implicit none
c-------------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-------------------------------------------------------------------------
-------
      real trenda,plunga,ax,ay,az,trendb,plungb,bx,by,bz,ang
      integer ierr
c
      call fpsset
c
      call ax2ca(trenda,plunga,ax,ay,az,ierr)
      if(ierr.ne.0) then
         write(io,'(1x,a,i3)') 'ANGLEA: ierr=',ierr
         return
      endif
      call ax2ca(trendb,plungb,bx,by,bz,ierr)
      if(ierr.ne.0) then
         ierr=ierr+4
         write(io,'(1x,a,i3)') 'ANGLEA: ierr=',ierr
         return
      endif
      call angle(ax,ay,az,bx,by,bz,ang)
      return
      end
*********************************************************************
*******
      subroutine norm(wax,way,waz,anorm,ax,ay,az)
c
c     compute euclidean norm and versor components
```

```
c
c     usage:
c     call norm(wax,way,waz,anorm,ax,ay,az)
c
c     arguments:
c     wax,way,waz    Cartesian component of input vector (INPUT)
c     anorm          Euclidean norm of input vector (OUTPUT)
c     ax,ay,az       normalized Cartesian component of the vector
(OUTPUT)
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real wax,way,waz,anorm,ax,ay,az
c
      call fpsset
c
      anorm=sqrt(wax*wax+way*way+waz*waz)
      if(anorm.eq.c0) return
      ax=wax/anorm
      ay=way/anorm
      az=waz/anorm
      return
      end
***********************************************************************
*******
      subroutine vecpro(px,py,pz,tx,ty,tz,bx,by,bz)
c
c     compute vector products of two vectors
c
c     usage:
c     call vecpro(px,py,pz,tx,ty,tz,bx,by,bz)
c
c     arguments:
c
c     px,py,pz       Cartesian component of first vector (INPUT)
c     tx,ty,tz       Cartesian component of second vector (INPUT)
c     bx,by,bz       Cartesian component of vector product (OUTUT)
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real px,py,pz,tx,ty,tz,bx,by,bz
c
```

```
      call fpsset
c
      bx=py*tz-pz*ty
      by=pz*tx-px*tz
      bz=px*ty-py*tx
      return
      end
*********************************************************************
*******
      subroutine invert(ax,ay,az)
c
c     invert vector
c
c     usage:
c     utility routine for internal use only
c
c      implicit none
c-------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-------------------------------------------------------------------
-------
      real ax,ay,az
c
      call fpsset
c
      ax=-ax
      ay=-ay
      az=-az
      return
      end
*********************************************************************
*******
      subroutine hatens(amrr,amss,amee,amrs,amre,amse,am)
c
c     build the Harvard CMT tensor from independent components
c
c     usage:
c     call hatens(amrr,amss,amee,amrs,amre,amse,am)
c
c     arguments:
c     amss           Harvard CMT South-South component (INPUT)
c     amse           Harvard CMT South-East component (INPUT)
c     amrs           Harvard CMT Radial-South component (INPUT)
c     amee           Harvard CMT East-East component (INPUT)
c     amre           Harvard CMT Radial-East component (INPUT)
c     amrr           Harvard CMT Radial-Radial component (INPUT)
c     am             seismic moment tensor (3x3 matrix) (OUTPUT)
c
c      implicit none
c-------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
```

```
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,amss,amse,amrs,amee,amre,amrr
      dimension am(3,3)
c
      call fpsset
c
      am(1,1)=amss
      am(1,2)=amse
      am(1,3)=amrs
      am(2,1)=amse
      am(2,2)=amee
      am(2,3)=amre
      am(3,1)=amrs
      am(3,2)=amre
      am(3,3)=amrr
      return
      end
***********************************************************************
*******
      subroutine tensha(am,amrr,amss,amee,amrs,amre,amse)
c
c     give independent components from Harvard CMT tensor
c
c     usage:
c     call tensha(am,amrr,amss,amee,amrs,amre,amse)
c
c     arguments:
c     am             seismic moment tensor (3x3 matrix) (INPUT)
c     amss           Harvard CMT South-South component (OUTPUT)
c     amse           Harvard CMT South-East component (OUTPUT)
c     amrs           Harvard CMT Radial-South component (OUTPUT)
c     amee           Harvard CMT East-East component (OUTPUT)
c     amre           Harvard CMT Radial-East component (OUTPUT)
c     amrr           Harvard CMT Radial-Radial component (OUTPUT)
c
c      implicit none
c-----------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-----------------------------------------------------------------------
-------
      real am,amss,amse,amrs,amee,amre,amrr
      dimension am(3,3)
c
      call fpsset
c
      amrr=am(3,3)
      amss=am(1,1)
      amee=am(2,2)
      amrs=am(3,1)
      amre=am(3,2)
```

```
      amse=am(1,2)
      return
      end
*********************************************************************
*******
      subroutine fpsset
c
c     define constants (i.e. input ranges and tolerances) used throughout
the
c     package. It is called by every subroutines to setup constants
c
c     usage:
c     call fpsset
c
c     constants in fpscom common block:
c
c     amistr         strike lower limit
c     amastr         strike upper limit
c     amidip         dip lower limit
c     amadip         dip upper limit
c     amirak         rake lower limit
c     amarak         rake upper limit
c     amitre         trend lower limit
c     amatre         trend upper limit
c     amiplu         plunge lower limit
c     amaplu         plunge upper limit
c     orttol         orthogonality tolerance
c     ovrtol         dip overtaking tolerance
c     tentol         moment tensor symmetry tolerance
c     dtor      degree to radians
c     c360      360.
c     c90            90.
c     c0             0.
c     c1             1.
c     c2             2.
c     c3             3.
c     io             error messages file unit
c
c      implicit none
c-------------------------------------------------------------------------
-------
      integer io
      real amistr,amastr,amidip,amadip,amirak,amarak,amitre,amatre
     1,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2,c3
      common /fpscom/amistr,amastr,amidip,amadip,amirak,amarak,amitre
     1,amatre,amiplu,amaplu,orttol,ovrtol,tentol,dtor,c360,c90,c0,c1,c2
     2,c3,io
c-------------------------------------------------------------------------
-------
      integer ifl
      save ifl
      data ifl/0/
      if(ifl.eq.0) then
         amistr=-360.
         amastr=360.
         amidip=0.
         amadip=90.
         amirak=-360.
         amarak=360.
         amitre=-360.
```

```
   amatre=360.
   amiplu=0.
   amaplu=90.
   orttol=2.
   ovrtol=0.001
   tentol=0.0001
   dtor=0.017453292519943296
   c360=360.
   c90=90.
   c0=0.
   c1=1.
   c2=2.
   c3=3.
   io=6
   ifl=1
endif
return
end
```