```
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      ELEMENT6.FOR    (ErikSoft   5 June 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Makes output files containing Keplerian orbital elements from data created
c by Mercury6 and higher.
c
c The user specifies the names of the required objects in the file elements.in
c See subroutine M_FORMAT for the identities of each element in the EL array
c e.g. el(1)=a, el(2)=e etc.
c
c------------------------------------------------------------------------------
c
      implicit none
      include 'mercury.inc'
c
      integer itmp,i,j,k,l,iback(NMAX),precision,lenin
      integer nmaster,nopen,nwait,nbig,nsml,nbod,nsub,lim(2,100)
      integer year,month,timestyle,line_num,lenhead,lmem(NMESS)
      integer nchar,algor,centre,allflag,firstflag,ninfile,nel,iel(22)
      integer nbod1,nbig1,unit(NMAX),code(NMAX),master_unit(NMAX)
      real*8 time,teval,t0,t1,tprevious,rmax,rcen,rfac,rhocgs,temp
      real*8 mcen,jcen(3),el(22,NMAX),s(3),is(NMAX),ns(NMAX),a(NMAX)
      real*8 mio_c2re, mio_c2fl,fr,theta,phi,fv,vtheta,vphi,gm
      real*8 x(3,NMAX),v(3,NMAX),xh(3,NMAX),vh(3,NMAX),m(NMAX)
      logical test
      character*250 string,fout,header,infile(50)
      character*80 mem(NMESS),cc,c(NMAX)
      character*8 master_id(NMAX),id(NMAX)
      character*5 fin
      character*1 check,style,type,c1
      character*2 c2
c
c------------------------------------------------------------------------------
c
      allflag = 0
      tprevious = 0.d0
      rhocgs = AU * AU * AU * K2 / MSUN
c
c Read in output messages
      inquire (file='message.in', exist=test)
      if (.not.test) then
        write (*,'(/,2a)') ' ERROR: This file is needed to continue: ',
     %    ' message.in'
        stop
      end if
      open (14, file='message.in', status='old')
  10  continue
        read (14,'(i3,1x,i2,1x,a80)',end=20) j,lmem(j),mem(j)
      goto 10
  20  close (14)
c
c Open file containing parameters for this programme
      inquire (file='element.in', exist=test)
      if (test) then
        open (10, file='element.in', status='old')
      else
        call mio_err (6,mem(81),lmem(81),mem(88),lmem(88),' ',1,
     %    'element.in',9)
      end if
c
c Read number of input files
  30  read (10,'(a250)') string
      if (string(1:1).eq.')') goto 30
      call mio_spl (250,string,nsub,lim)
      read (string(lim(1,nsub):lim(2,nsub)),*) ninfile
c
c Make sure all the input files exist
```

```
      do j = 1, ninfile
   40   read (10,'(a250)') string
        if (string(1:1).eq.')') goto 40
        call mio_spl (250,string,nsub,lim)
        infile(j)(1:(lim(2,1)-lim(1,1)+1)) = string(lim(1,1):lim(2,1))
        inquire (file=infile(j), exist=test)
        if (.not.test) call mio_err (6,mem(81),lmem(81),mem(88),
     %    lmem(88),' ',1,infile(j),80)
      end do
c
c What type elements does the user want?
      centre = 0
   45 read (10,'(a250)') string
      if (string(1:1).eq.')') goto 45
      call mio_spl (250,string,nsub,lim)
      c2 = string(lim(1,nsub):(lim(1,nsub)+1))
      if (c2.eq.'ce'.or.c2.eq.'CE'.or.c2.eq.'Ce') then
        centre = 0
      else if (c2.eq.'ba'.or.c2.eq.'BA'.or.c2.eq.'Ba') then
        centre = 1
      else if (c2.eq.'ja'.or.c2.eq.'JA'.or.c2.eq.'Ja') then
        centre = 2
      else
        call mio_err (6,mem(81),lmem(81),mem(107),lmem(107),' ',1,
     %    '       Check element.in',23)
      end if
c
c Read parameters used by this programme
      timestyle = 1
      do j = 1, 4
   50   read (10,'(a250)') string
        if (string(1:1).eq.')') goto 50
        call mio_spl (250,string,nsub,lim)
        c1 = string(lim(1,nsub):lim(2,nsub))
        if (j.eq.1) read (string(lim(1,nsub):lim(2,nsub)),*) teval
        teval = abs(teval) * .999d0
        if (j.eq.2.and.(c1.eq.'d'.or.c1.eq.'D')) timestyle = 0
        if (j.eq.3.and.(c1.eq.'y'.or.c1.eq.'Y')) timestyle = timestyle+2
        if (j.eq.4) call m_format (string,timestyle,nel,iel,fout,header,
     %    lenhead)
      end do
c
c Read in the names of the objects for which orbital elements are required
      nopen = 0
      nwait = 0
      nmaster = 0
   60 continue
        read (10,'(a250)',end=70) string
        call mio_spl (250,string,nsub,lim)
        if (string(1:1).eq.')'.or.lim(1,1).eq.-1) goto 60
c
c Either open an aei file for this object or put it on the waiting list
        nmaster = nmaster + 1
        itmp = min(7,lim(2,1)-lim(1,1))
        master_id(nmaster)='        '
        master_id(nmaster)(1:itmp+1) = string(lim(1,1):lim(1,1)+itmp)
        if (nopen.lt.NFILES) then
          nopen = nopen + 1
          master_unit(nmaster) = 10 + nopen
          call mio_aei (master_id(nmaster),'.aei',master_unit(nmaster),
     %      header,lenhead,mem,lmem)
        else
          nwait = nwait + 1
          master_unit(nmaster) = -2
        end if
      goto 60
c
   70 continue
c If no objects are listed in ELEMENT.IN assume that all objects are required
      if (nopen.eq.0) allflag = 1
      close (10)
c
```

```
c-------------------------------------------------------------------------------
c
c  LOOP  OVER  EACH  INPUT  FILE  CONTAINING  INTEGRATION  DATA
c
  90  continue
      firstflag = 0
      do i = 1, ninfile
        line_num = 0
        open (10, file=infile(i), status='old')
c
c Loop over each time slice
 100    continue
        line_num = line_num + 1
        read (10,'(3a1)',end=900,err=666) check,style,type
        line_num = line_num - 1
        backspace 10
c
c Check if this is an old style input file
        if (ichar(check).eq.12.and.(style.eq.'0'.or.style.eq.'1'.or.
     %    style.eq.'2'.or.style.eq.'3'.or.style.eq.'4')) then
          write (*,'(/,2a)') ' ERROR: This is an old style data file',
     %      '         Try running m_elem5.for instead.'
          stop
        end if
        if (ichar(check).ne.12) goto 666
c
c-------------------------------------------------------------------------------
c
c  IF  SPECIAL  INPUT,  READ  TIME,  PARAMETERS,  NAMES,  MASSES  ETC.
c
        if (type.eq.'a') then
          line_num = line_num + 1
          read (10,'(3x,i2,a62,i1)') algor,cc(1:62),precision
c
c Decompress the time, number of objects, central mass and J components etc.
          time = mio_c2fl (cc(1:8))
          nbig = int(.5d0 + mio_c2re(cc(9:16), 0.d0, 11239424.d0, 3))
          nsml = int(.5d0 + mio_c2re(cc(12:19),0.d0, 11239424.d0, 3))
          mcen = mio_c2fl (cc(15:22))
          jcen(1) = mio_c2fl (cc(23:30))
          jcen(2) = mio_c2fl (cc(31:38))
          jcen(3) = mio_c2fl (cc(39:46))
          rcen = mio_c2fl (cc(47:54))
          rmax = mio_c2fl (cc(55:62))
          rfac = log10 (rmax / rcen)
c
c Read in strings containing compressed data for each object
          do j = 1, nbig + nsml
            line_num = line_num + 1
            read (10,'(a)',err=666) c(j)(1:51)
          end do
c
c Create input format list
          if (precision.eq.1) nchar = 2
          if (precision.eq.2) nchar = 4
          if (precision.eq.3) nchar = 7
          lenin = 3  +  6 * nchar
          fin(1:5) = '(a00)'
          write (fin(3:4),'(i2)') lenin
c
c For each object decompress its name, code number, mass, spin and density
          do j = 1, nbig + nsml
            k = int(.5d0 + mio_c2re(c(j)(1:8),0.d0,11239424.d0,3))
            id(k) = c(j)(4:11)
            el(18,k) = mio_c2fl (c(j)(12:19))
            s(1) = mio_c2fl (c(j)(20:27))
            s(2) = mio_c2fl (c(j)(28:35))
            s(3) = mio_c2fl (c(j)(36:43))
            el(21,k) = mio_c2fl (c(j)(44:51))
c
c Calculate spin rate and longitude & inclination of spin vector
            temp = sqrt(s(1)*s(1) + s(2)*s(2) + s(3)*s(3))
```

```
            if (temp.gt.0) then
              call mce_spin (1.d0,el(18,k)*K2,temp*K2,el(21,k)*
     %            rhocgs,el(20,k))
              temp = s(3) / temp
              if (abs(temp).lt.1) then
                is(k) = acos (temp)
                ns(k) = atan2 (s(1), -s(2))
              else
                if (temp.gt.0) is(k) = 0.d0
                if (temp.lt.0) is(k) = PI
                ns(k) = 0.d0
              end if
            else
              el(20,k) = 0.d0
              is(k) = 0.d0
              ns(k) = 0.d0
            end if
c
c Find the object on the master list
            unit(k) = 0
            do l = 1, nmaster
              if (id(k).eq.master_id(l)) unit(k) = master_unit(l)
            end do
c
c If object is not on the master list, add it to the list now
            if (unit(k).eq.0) then
              nmaster = nmaster + 1
              master_id(nmaster) = id(k)
c
c Either open an aei file for this object or put it on the waiting list
              if (allflag.eq.1) then
                if (nopen.lt.NFILES) then
                  nopen = nopen + 1
                  master_unit(nmaster) = 10 + nopen
                  call mio_aei (master_id(nmaster),'.aei',
     %              master_unit(nmaster),header,lenhead,mem,lmem)
                else
                  nwait = nwait + 1
                  master_unit(nmaster) = -2
                end if
              else
                master_unit(nmaster) = -1
              end if
              unit(k) = master_unit(nmaster)
            end if
          end do
c
c-----------------------------------------------------------------------
c
c  IF  NORMAL  INPUT,  READ  COMPRESSED  ORBITAL  VARIABLES  FOR  ALL  OBJECTS
c
        else if (type.eq.'b') then
          line_num = line_num + 1
          read (10,'(3x,a14)',err=666) cc(1:14)
c
c Decompress the time and the number of objects
          time = mio_c2fl (cc(1:8))
          nbig = int(.5d0 + mio_c2re(cc(9:16),  0.d0, 11239424.d0, 3))
          nsml = int(.5d0 + mio_c2re(cc(12:19), 0.d0, 11239424.d0, 3))
          nbod = nbig + nsml
          if (firstflag.eq.0) t0 = time
c
c Read in strings containing compressed data for each object
          do j = 1, nbod
            line_num = line_num + 1
            read (10,fin,err=666) c(j)(1:lenin)
          end do
c
c Look for objects for which orbital elements are required
          m(1) = mcen * K2
          do j = 1, nbod
            code(j) = int(.5d0 + mio_c2re(c(j)(1:8), 0.d0,
```

```
     %        11239424.d0, 3))
              if (code(j).gt.NMAX) then
                write (*,'(/,2a)') mem(81)(1:lmem(81)),
     %            mem(90)(1:lmem(90))
                stop
              end if
c
c Decompress orbital variables for each object
              l = j + 1
              m(l) = el(18,code(j)) * K2
              fr     = mio_c2re (c(j)(4:11), 0.d0, rfac,  nchar)
              theta  = mio_c2re (c(j)(4+  nchar:11+  nchar), 0.d0, PI,
     %                 nchar)
              phi    = mio_c2re (c(j)(4+2*nchar:11+2*nchar), 0.d0, TWOPI,
     %                 nchar)
              fv     = mio_c2re (c(j)(4+3*nchar:11+3*nchar), 0.d0, 1.d0,
     %                 nchar)
              vtheta = mio_c2re (c(j)(4+4*nchar:11+4*nchar), 0.d0, PI,
     %                 nchar)
              vphi   = mio_c2re (c(j)(4+5*nchar:11+5*nchar), 0.d0, TWOPI,
     %                 nchar)
              call mco_ov2x (rcen,rmax,m(1),m(l),fr,theta,phi,fv,
     %          vtheta,vphi,x(1,l),x(2,l),x(3,l),v(1,l),v(2,l),v(3,l))
              el(16,code(j)) = sqrt(x(1,l)*x(1,l) + x(2,l)*x(2,l)
     %                      + x(3,l)*x(3,l))
            end do
c
c Convert to barycentric, Jacobi or close-binary coordinates if desired
            nbod1 = nbod + 1
            nbig1 = nbig + 1
            call mco_iden (jcen,nbod1,nbig1,temp,m,x,v,xh,vh)
            if (centre.eq.1) call mco_h2b (jcen,nbod1,nbig1,temp,m,xh,vh,
     %        x,v)
            if (centre.eq.2) call mco_h2j (jcen,nbod1,nbig1,temp,m,xh,vh,
     %        x,v)
            if (centre.eq.0.and.algor.eq.11) call mco_h2cb (jcen,nbod1,
     %        nbig1,temp,m,xh,vh,x,v)
c
c Put Cartesian coordinates into element arrays
            do j = 1, nbod
              k = code(j)
              l = j + 1
              el(10,k) = x(1,l)
              el(11,k) = x(2,l)
              el(12,k) = x(3,l)
              el(13,k) = v(1,l)
              el(14,k) = v(2,l)
              el(15,k) = v(3,l)
c
c Convert to Keplerian orbital elements
              gm = (mcen + el(18,k)) * K2
              call mco_x2el (gm,el(10,k),el(11,k),el(12,k),el(13,k),
     %          el(14,k),el(15,k),el(8,k),el(2,k),el(3,k),el(7,k),
     %          el(5,k),el(6,k))
              el(1,k) = el(8,k) / (1.d0 - el(2,k))
              el(9,k) = el(1,k) * (1.d0 + el(2,k))
              el(4,k) = mod(el(7,k) - el(5,k) + TWOPI, TWOPI)
c Calculate true anomaly
              if (el(2,k).eq.0) then
                el(17,k) = el(6,k)
              else
                temp = (el(8,k)*(1.d0 + el(2,k))/el(16,k) - 1.d0) /el(2,k)
                temp = sign (min(abs(temp), 1.d0), temp)
                el(17,k) = acos(temp)
                if (sin(el(6,k)).lt.0) el(17,k) = TWOPI - el(17,k)
              end if
c Calculate obliquity
              el(19,k) = acos (cos(el(3,k))*cos(is(k))
     %          + sin(el(3,k))*sin(is(k))*cos(ns(k) - el(5,k)))
c
c Convert angular elements from radians to degrees
              do l = 3, 7
```

```
              el(l,k) = mod(el(l,k) / DR, 360.d0)
            end do
            el(17,k) = el(17,k) / DR
            el(19,k) = el(19,k) / DR
          end do
c
c Convert time to desired format
          if (timestyle.eq.0) t1 = time
          if (timestyle.eq.1) call mio_jd_y (time,year,month,t1)
          if (timestyle.eq.2) t1 = time - t0
          if (timestyle.eq.3) t1 = (time - t0) / 365.25d0
c
c If output is required at this epoch, write elements to appropriate files
          if (firstflag.eq.0.or.abs(time-tprevious).ge.teval) then
            firstflag = 1
            tprevious = time
c
c Write required elements to the appropriate aei file
            do j = 1, nbod
              k = code(j)
              if (unit(k).ge.10) then
                if (timestyle.eq.1) then
                  write (unit(k),fout) year,month,t1,(el(iel(l),k),l=1,
     %               nel)
                else
                  write (unit(k),fout) t1,(el(iel(l),k),l=1,nel)
                end if
              end if
            end do
          end if
c
c-------------------------------------------------------------------------
c
c  IF  TYPE  IS  NOT  'a'  OR  'b',  THE  INPUT  FILE  IS  CORRUPTED
c
        else
          goto 666
        end if
c
c Move on to the next time slice
        goto 100
c
c If input file is corrupted, try to continue from next uncorrupted time slice
 666    continue
        write (*,'(2a,/,a,i10)') mem(121)(1:lmem(121)),
     %    infile(i)(1:60),mem(104)(1:lmem(104)),line_num
        c1 = ' '
        do while (ichar(c1).ne.12)
          line_num = line_num + 1
          read (10,'(a1)',end=900) c1
        end do
        line_num = line_num - 1
        backspace 10
c
c Move on to the next file containing integration data
 900    continue
        close (10)
      end do
c
c Close aei files
      do j = 1, nopen
        close (10+j)
      end do
      nopen = 0
c
c If some objects remain on waiting list, read through input files again
      if (nwait.gt.0) then
        do j = 1, nmaster
          if (master_unit(j).ge.10) master_unit(j) = -1
          if (master_unit(j).eq.-2.and.nopen.lt.NFILES) then
            nopen = nopen + 1
            nwait = nwait - 1
```

```
            master_unit(j) = 10 + nopen
            call mio_aei (master_id(j),'.aei',master_unit(j),header,
     %        lenhead,mem,lmem)
          end if
        end do
        goto 90
      end if
c
c-----------------------------------------------------------------------
c
c  CREATE  A  SUMMARY  OF  FINAL  MASSES  AND  ELEMENTS
c
      open (10, file='element.out', status='unknown')
      rewind 10
c
      if (timestyle.eq.0.or.timestyle.eq.2) then
        write (10,'(/,a,f18.5,/)') ' Time (days): ',t1
      else if (timestyle.eq.1) then
        write (10,'(/,a,i10,1x,i2,1x,f8.5,/)') ' Date: ',year,month,t1
      else if (timestyle.eq.3) then
        write (10,'(/,a,f18.7,/)') ' Time (years): ',t1
      end if
      write (10,'(2a,/)') '              a        e       i      mass',
     %  '    Rot/day  Obl'
c
c Sort surviving objects in order of increasing semi-major axis
      do j = 1, nbod
        k = code(j)
        a(j) = el(1,k)
      end do
      call mxx_sort (nbod,a,iback)
c
c Write values of a, e, i and m for surviving objects in an output file
      do j = 1, nbod
        k = code(iback(j))
        write (10,213) id(k),el(1,k),el(2,k),el(3,k),el(18,k),el(20,k),
     %      el(19,k)
      end do
c
c-----------------------------------------------------------------------
c
c Format statements
 213  format (1x,a8,1x,f8.4,1x,f7.5,1x,f7.3,1p,e11.4,0p,1x,f6.3,1x,f6.2)
c
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_OV2X.FOR    (ErikSoft   28 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts output variables for an object to coordinates and velocities.
c The output variables are:
c  r = the radial distance
c  theta = polar angle
c  phi = azimuthal angle
c  fv = 1 / [1 + 2(ke/be)^2], where be and ke are the object's binding and
c                             kinetic energies. (Note that 0 < fv < 1).
c  vtheta = polar angle of velocity vector
c  vphi = azimuthal angle of the velocity vector
c
c-----------------------------------------------------------------------
c
      subroutine mco_ov2x (rcen,rmax,mcen,m,fr,theta,phi,fv,vtheta,
     %  vphi,x,y,z,u,v,w)
c
      implicit none
      include 'mercury.inc'
c
```

```
c Input/Output
      real*8 rcen,rmax,mcen,m,x,y,z,u,v,w,fr,theta,phi,fv,vtheta,vphi
c
c Local
      real*8 r,v1,temp
c
c-----------------------------------------------------------------------------
c
        r = rcen * 10.d0**fr
        temp = sqrt(.5d0*(1.d0/fv - 1.d0))
        v1 = sqrt(2.d0 * temp * (mcen + m) / r)
c
        x = r * sin(theta) * cos(phi)
        y = r * sin(theta) * sin(phi)
        z = r * cos(theta)
        u = v1 * sin(vtheta) * cos(vphi)
        v = v1 * sin(vtheta) * sin(vphi)
        w = v1 * cos(vtheta)
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCE_SPIN.FOR    (ErikSoft  2 December 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates the spin rate (in rotations per day) for a fluid body given
c its mass, spin angular momentum and density. The routine assumes the
c body is a MacClaurin ellipsoid, whose axis ratio is defined by the
c quantity SS = SQRT(A^2/C^2 - 1), where A and C are the
c major and minor axes.
c
c-----------------------------------------------------------------------------
c
      subroutine mce_spin (g,mass,spin,rho,rote)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      real*8 g,mass,spin,rho,rote
c
c Local
      integer k
      real*8 ss,s2,f,df,z,dz,tmp0,tmp1,t23
c
c-----------------------------------------------------------------------------
c
      t23 = 2.d0 / 3.d0
      tmp1 = spin * spin / (2.d0 * PI * rho * g)
     %     * ( 250.d0*PI*PI*rho*rho / (9.d0*mass**5) )**t23
c
c Calculate SS using Newton's method
      ss = 1.d0
      do k = 1, 20
        s2 = ss * ss
        tmp0 = (1.d0 + s2)**t23
        call m_sfunc (ss,z,dz)
        f = z * tmp0  -  tmp1
        df = tmp0 * ( dz  +  4.d0 * ss * z / (3.d0*(1.d0 + s2)) )
        ss = ss - f/df
      end do
c
      rote = sqrt(TWOPI * g * rho * z) / TWOPI
c
c-----------------------------------------------------------------------------
```

```
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_EL2X.FOR    (ErikSoft  7 July 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates Cartesian coordinates and velocities given Keplerian orbital
c elements (for elliptical, parabolic or hyperbolic orbits).
c
c Based on a routine from Levison and Duncan's SWIFT integrator.
c
c  mu = grav const * (central + secondary mass)
c  q = perihelion distance
c  e = eccentricity
c  i = inclination                 )
c  p = longitude of perihelion !!! )   in
c  n = longitude of ascending node ) radians
c  l = mean anomaly                )
c
c  x,y,z = Cartesian positions  ( units the same as a )
c  u,v,w =     "     velocities ( units the same as sqrt(mu/a) )
c
c------------------------------------------------------------------------------
c
      subroutine mco_el2x (mu,q,e,i,p,n,l,x,y,z,u,v,w)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      real*8 mu,q,e,i,p,n,l,x,y,z,u,v,w
c
c Local
      real*8 g,a,ci,si,cn,sn,cg,sg,ce,se,romes,temp
      real*8 z1,z2,z3,z4,d11,d12,d13,d21,d22,d23
      real*8 mco_kep, orbel_fhybrid, orbel_zget
c
c------------------------------------------------------------------------------
c
c Change from longitude of perihelion to argument of perihelion
      g = p - n
c
c Rotation factors
      call mco_sine (i,si,ci)
      call mco_sine (g,sg,cg)
      call mco_sine (n,sn,cn)
      z1 = cg * cn
      z2 = cg * sn
      z3 = sg * cn
      z4 = sg * sn
      d11 =  z1 - z4*ci
      d12 =  z2 + z3*ci
      d13 = sg * si
      d21 = -z3 - z2*ci
      d22 = -z4 + z1*ci
      d23 = cg * si
c
c Semi-major axis
      a = q / (1.d0 - e)
c
c Ellipse
      if (e.lt.1.d0) then
        romes = sqrt(1.d0 - e*e)
        temp = mco_kep (e,l)
        call mco_sine (temp,se,ce)
        z1 = a * (ce - e)
```

```
        z2 = a * romes * se
        temp = sqrt(mu/a) / (1.d0 - e*ce)
        z3 = -se * temp
        z4 = romes * ce * temp
      else
c Parabola
        if (e.eq.1.d0) then
          ce = orbel_zget(l)
          z1 = q * (1.d0 - ce*ce)
          z2 = 2.d0 * q * ce
          z4 = sqrt(2.d0*mu/q) / (1.d0 + ce*ce)
          z3 = -ce * z4
        else
c Hyperbola
          romes = sqrt(e*e - 1.d0)
          temp = orbel_fhybrid(e,l)
          call mco_sinh (temp,se,ce)
          z1 = a * (ce - e)
          z2 = -a * romes * se
          temp = sqrt(mu/abs(a)) / (e*ce - 1.d0)
          z3 = -se * temp
          z4 = romes * ce * temp
        end if
      endif
c
      x = d11*z1 + d21*z2
      y = d12*z1 + d22*z2
      z = d13*z1 + d23*z2
      u = d11*z3 + d21*z4
      v = d12*z3 + d22*z4
      w = d13*z3 + d23*z4
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_KEP.FOR    (ErikSoft  7 July 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Solves Kepler's equation for eccentricities less than one.
c Algorithm from A. Nijenhuis (1991) Cel. Mech. Dyn. Astron. 51, 319-330.
c
c  e = eccentricity
c  l = mean anomaly      (radians)
c  u = eccentric anomaly (   "   )
c
c-----------------------------------------------------------------------------
c
      function mco_kep (e,oldl)
      implicit none
c
c Input/Outout
      real*8 oldl,e,mco_kep
c
c Local
      real*8 l,pi,twopi,piby2,u1,u2,ome,sign
      real*8 x,x2,sn,dsn,z1,z2,z3,f0,f1,f2,f3
      real*8 p,q,p2,ss,cc
      logical flag,big,bigg
c
c-----------------------------------------------------------------------------
c
      pi = 3.141592653589793d0
      twopi = 2.d0 * pi
      piby2 = .5d0 * pi
c
```

```
c Reduce mean anomaly to lie in the range 0 < l < pi
      if (oldl.ge.0) then
        l = mod(oldl, twopi)
      else
        l = mod(oldl, twopi) + twopi
      end if
      sign = 1.d0
      if (l.gt.pi) then
        l = twopi - l
        sign = -1.d0
      end if
c
      ome = 1.d0 - e
c
      if (l.ge..45d0.or.e.lt..55d0) then
c
c Regions A,B or C in Nijenhuis
c ----------------------------
c
c Rough starting value for eccentric anomaly
        if (l.lt.ome) then
          u1 = ome
        else
          if (l.gt.(pi-1.d0-e)) then
            u1 = (l+e*pi)/(1.d0+e)
          else
            u1 = l + e
          end if
        end if
c
c Improved value using Halley's method
        flag = u1.gt.piby2
        if (flag) then
          x = pi - u1
        else
          x = u1
        end if
        x2 = x*x
        sn = x*(1.d0 + x2*(-.16605 + x2*.00761) )
        dsn = 1.d0 + x2*(-.49815 + x2*.03805)
        if (flag) dsn = -dsn
        f2 = e*sn
        f0 = u1 - f2 - l
        f1 = 1.d0 - e*dsn
        u2 = u1 - f0/(f1 - .5d0*f0*f2/f1)
      else
c
c Region D in Nijenhuis
c ---------------------
c
c Rough starting value for eccentric anomaly
        z1 = 4.d0*e + .5d0
        p = ome / z1
        q = .5d0 * l / z1
        p2 = p*p
        z2 = exp( log( dsqrt( p2*p + q*q ) + q )/1.5 )
        u1 = 2.d0*q / ( z2 + p + p2/z2 )
c
c Improved value using Newton's method
        z2 = u1*u1
        z3 = z2*z2
        u2 = u1 - .075d0*u1*z3 / (ome + z1*z2 + .375d0*z3)
        u2 = l + e*u2*( 3.d0 - 4.d0*u2*u2 )
      end if
c
c Accurate value using 3rd-order version of Newton's method
c N.B. Keep cos(u2) rather than sqrt( 1-sin^2(u2) ) to maintain accuracy!
c
c First get accurate values for u2 - sin(u2) and 1 - cos(u2)
      bigg = (u2.gt.piby2)
      if (bigg) then
        z3 = pi - u2
```

```
      else
        z3 = u2
      end if
c
      big = (z3.gt.(.5d0*piby2))
      if (big) then
        x = piby2 - z3
      else
        x = z3
      end if
c
      x2 = x*x
      ss = 1.d0
      cc = 1.d0
c
      ss = x*x2/6.*(1. - x2/20.*(1. - x2/42.*(1. - x2/72.*(1. -
     %   x2/110.*(1. - x2/156.*(1. - x2/210.*(1. - x2/272.)))))))
      cc =   x2/2.*(1. - x2/12.*(1. - x2/30.*(1. - x2/56.*(1. -
     %   x2/ 90.*(1. - x2/132.*(1. - x2/182.*(1. - x2/240.*(1. -
     %   x2/306.))))))))
c
      if (big) then
        z1 = cc + z3 - 1.d0
        z2 = ss + z3 + 1.d0 - piby2
      else
        z1 = ss
        z2 = cc
      end if
c
      if (bigg) then
        z1 = 2.d0*u2 + z1 - pi
        z2 = 2.d0 - z2
      end if
c
      f0 = l - u2*ome - e*z1
      f1 = ome + e*z2
      f2 = .5d0*e*(u2-z1)
      f3 = e/6.d0*(1.d0-z2)
      z1 = f0/f1
      z2 = f0/(f2*z1+f1)
      mco_kep = sign*( u2 + f0/((f3*z1+f2)*z2+f1) )
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_SINE.FOR    (ErikSoft  17 April 1997)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates sin and cos of an angle X (in radians).
c
c-----------------------------------------------------------------------
c
      subroutine mco_sine (x,sx,cx)
c
      implicit none
c
c Input/Output
      real*8 x,sx,cx
c
c Local
      real*8 pi,twopi
c
c-----------------------------------------------------------------------
c
      pi = 3.141592653589793d0
```

```
      twopi = 2.d0 * pi
c
      if (x.gt.0) then
        x = mod(x,twopi)
      else
        x = mod(x,twopi) + twopi
      end if
c
      cx = cos(x)
c
      if (x.gt.pi) then
        sx = -sqrt(1.d0 - cx*cx)
      else
        sx =  sqrt(1.d0 - cx*cx)
      end if
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_SINH.FOR    (ErikSoft  12 June 1998)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Calculates sinh and cosh of an angle X (in radians)
c
c-----------------------------------------------------------------------------
c
      subroutine mco_sinh (x,sx,cx)
c
      implicit none
c
c Input/Output
      real*8 x,sx,cx
c
c-----------------------------------------------------------------------------
c
      sx = sinh(x)
      cx = sqrt (1.d0 + sx*sx)
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_AEI.FOR    (ErikSoft   31 January 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Creates a filename and opens a file to store aei information for an object.
c The filename is based on the name of the object.
c
c-----------------------------------------------------------------------------
c
      subroutine mio_aei (id,extn,unitnum,header,lenhead,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer unitnum,lenhead,lmem(NMESS)
      character*4 extn
      character*8 id
      character*250 header
      character*80 mem(NMESS)
```

```
c
c Local
      integer j,k,itmp,nsub,lim(2,4)
      logical test
      character*1 bad(5)
      character*250 filename
c
c-----------------------------------------------------------------------------
c
      data bad/ '*', '/', '.', ':', '&'/
c
c Create a filename based on the object's name
      call mio_spl (8,id,nsub,lim)
      itmp = min(7,lim(2,1)-lim(1,1))
      filename(1:itmp+1) = id(1:itmp+1)
      filename(itmp+2:itmp+5) = extn
      do j = itmp + 6, 250
        filename(j:j) = ' '
      end do
c
c Check for inappropriate characters in the filename
      do j = 1, itmp + 1
        do k = 1, 5
          if (filename(j:j).eq.bad(k)) filename(j:j) = '_'
        end do
      end do
c
c If the file exists already, give a warning and don't overwrite it
      inquire (file=filename, exist=test)
      if (test) then
        write (*,'(/,3a)') mem(121)(1:lmem(121)),mem(87)(1:lmem(87)),
     %    filename(1:80)
        unitnum = -1
      else
        open (unitnum, file=filename, status='new')
        write (unitnum, '(/,30x,a8,//,a)') id,header(1:lenhead)
      end if
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_C2FL.FOR    (ErikSoft   5 June 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c CHARACTER*8 ASCII string into a REAL*8 variable.
c
c N.B. X will lie in the range -1.e112 < X < 1.e112
c ===
c
c-----------------------------------------------------------------------------
c
      function mio_c2fl (c)
c
      implicit none
c
c Input/Output
      real*8 mio_c2fl
      character*8 c
c
c Local
      real*8 x,mio_c2re
      integer ex
c
c-----------------------------------------------------------------------------
c
      x = mio_c2re (c(1:8), 0.d0, 1.d0, 7)
      x = x * 2.d0 - 1.d0
```

```
      ex = mod(ichar(c(8:8)) + 256, 256) - 32 - 112
      mio_c2fl = x * (10.d0**dble(ex))
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_C2RE.FOR    (ErikSoft   5 June 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts an ASCII string into a REAL*8 variable X, where XMIN <= X < XMAX,
c using the new format compression:
c
c X is assumed to be made up of NCHAR base-224 digits, each one represented
c by a character in the ASCII string. Each digit is given by the ASCII
c number of the character minus 32.
c The first 32 ASCII characters (CTRL characters) are avoided, because they
c cause problems when using some operating systems.
c
c-----------------------------------------------------------------------
c
      function mio_c2re (c,xmin,xmax,nchar)
c
      implicit none
c
c Input/output
      integer nchar
      real*8 xmin,xmax,mio_c2re
      character*8 c
c
c Local
      integer j
      real*8 y
c
c-----------------------------------------------------------------------
c
      y = 0
      do j = nchar, 1, -1
        y = (y + dble(mod(ichar(c(j:j)) + 256, 256) - 32)) / 224.d0
      end do
c
      mio_c2re = xmin  +  y * (xmax - xmin)
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_ERR.FOR    (ErikSoft  6 December 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Writes out an error message and terminates Mercury.
c
c-----------------------------------------------------------------------
c
      subroutine mio_err (unit,s1,ls1,s2,ls2,s3,ls3,s4,ls4)
c
      implicit none
c
c Input/Output
      integer unit,ls1,ls2,ls3,ls4
```

```
      character*80 s1,s2,s3,s4
c
c-----------------------------------------------------------------------------
c
      write (*,'(a)') ' ERROR: Programme terminated.'
      write (unit,'(/,3a,/,2a)') s1(1:ls1),s2(1:ls2),s3(1:ls3),
     %  ' ',s4(1:ls4)
      stop
c
c-----------------------------------------------------------------------------
c
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_H2B.FOR    (ErikSoft   2 November 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts coordinates with respect to the central body to barycentric
c coordinates.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_h2b (jcen,nbod,nbig,h,m,xh,vh,x,v)
c
      implicit none
c
c Input/Output
      integer nbod,nbig
      real*8 jcen(3),h,m(nbod),xh(3,nbod),vh(3,nbod),x(3,nbod),v(3,nbod)
c
c Local
      integer j
      real*8 mtot,temp
c
c-----------------------------------------------------------------------------
c
      mtot = 0.d0
      x(1,1) = 0.d0
      x(2,1) = 0.d0
      x(3,1) = 0.d0
      v(1,1) = 0.d0
      v(2,1) = 0.d0
      v(3,1) = 0.d0
c
c Calculate coordinates and velocities of the central body
      do j = 2, nbod
        mtot = mtot  +  m(j)
        x(1,1) = x(1,1)  +  m(j) * xh(1,j)
        x(2,1) = x(2,1)  +  m(j) * xh(2,j)
        x(3,1) = x(3,1)  +  m(j) * xh(3,j)
        v(1,1) = v(1,1)  +  m(j) * vh(1,j)
        v(2,1) = v(2,1)  +  m(j) * vh(2,j)
        v(3,1) = v(3,1)  +  m(j) * vh(3,j)
      enddo
c
      temp = -1.d0 / (mtot + m(1))
      x(1,1) = temp * x(1,1)
      x(2,1) = temp * x(2,1)
      x(3,1) = temp * x(3,1)
      v(1,1) = temp * v(1,1)
      v(2,1) = temp * v(2,1)
      v(3,1) = temp * v(3,1)
c
c Calculate the barycentric coordinates and velocities
      do j = 2, nbod
        x(1,j) = xh(1,j) + x(1,1)
        x(2,j) = xh(2,j) + x(2,1)
        x(3,j) = xh(3,j) + x(3,1)
```

```
        v(1,j) = vh(1,j) + v(1,1)
        v(2,j) = vh(2,j) + v(2,1)
        v(3,j) = vh(3,j) + v(3,1)
      enddo
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_H2CB.FOR    (ErikSoft   2 November 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Convert coordinates with respect to the central body to close-binary
c coordinates.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_h2cb (jcen,nbod,nbig,h,m,xh,vh,x,v)
c
      implicit none
c
c Input/Output
      integer nbod,nbig
      real*8 jcen(3),h,m(nbod),xh(3,nbod),vh(3,nbod),x(3,nbod),v(3,nbod)
c
c Local
      integer j
      real*8 msum,mvsum(3),temp,mbin,mbin_1,mtot_1
c
c-----------------------------------------------------------------------------
c
      msum = 0.d0
      mvsum(1) = 0.d0
      mvsum(2) = 0.d0
      mvsum(3) = 0.d0
      mbin = m(1) + m(2)
      mbin_1 = 1.d0 / mbin
c
      x(1,2) = xh(1,2)
      x(2,2) = xh(2,2)
      x(3,2) = xh(3,2)
      temp = m(1) * mbin_1
      v(1,2) = temp * vh(1,2)
      v(2,2) = temp * vh(2,2)
      v(3,2) = temp * vh(3,2)
c
      do j = 3, nbod
        msum = msum + m(j)
        mvsum(1) = mvsum(1)  +  m(j) * vh(1,j)
        mvsum(2) = mvsum(2)  +  m(j) * vh(2,j)
        mvsum(3) = mvsum(3)  +  m(j) * vh(3,j)
      end do
      mtot_1 = 1.d0 / (msum + mbin)
      mvsum(1) = mtot_1 * (mvsum(1) + m(2)*vh(1,2))
      mvsum(2) = mtot_1 * (mvsum(2) + m(2)*vh(2,2))
      mvsum(3) = mtot_1 * (mvsum(3) + m(2)*vh(3,2))
c
      temp = m(2) * mbin_1
      do j = 3, nbod
        x(1,j) = xh(1,j)  -  temp * xh(1,2)
        x(2,j) = xh(2,j)  -  temp * xh(2,2)
        x(3,j) = xh(3,j)  -  temp * xh(3,2)
        v(1,j) = vh(1,j)  -  mvsum(1)
        v(2,j) = vh(2,j)  -  mvsum(2)
        v(3,j) = vh(3,j)  -  mvsum(3)
      end do
```

```
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_H2J.FOR    (ErikSoft   2 November 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts coordinates with respect to the central body to Jacobi coordinates.
c Note that the Jacobi coordinates of all small bodies are assumed to be the
c same as their coordinates with respect to the central body.
c
c-----------------------------------------------------------------------
c
      subroutine mco_h2j (jcen,nbod,nbig,h,m,xh,vh,x,v)
c
      implicit none
c
c Input/Output
      integer nbod,nbig
      real*8 jcen(3),h,m(nbig),xh(3,nbig),vh(3,nbig),x(3,nbig),v(3,nbig)
c
c Local
      integer j
      real*8 mtot, mx, my, mz, mu, mv, mw, temp
c
c-----------------------------------------------------------------------c
      mtot = m(2)
      x(1,2) = xh(1,2)
      x(2,2) = xh(2,2)
      x(3,2) = xh(3,2)
      v(1,2) = vh(1,2)
      v(2,2) = vh(2,2)
      v(3,2) = vh(3,2)
      mx = m(2) * xh(1,2)
      my = m(2) * xh(2,2)
      mz = m(2) * xh(3,2)
      mu = m(2) * vh(1,2)
      mv = m(2) * vh(2,2)
      mw = m(2) * vh(3,2)
c
      do j = 3, nbig - 1
        temp = 1.d0 / (mtot + m(1))
        mtot = mtot + m(j)
        x(1,j) = xh(1,j)  -  temp * mx
        x(2,j) = xh(2,j)  -  temp * my
        x(3,j) = xh(3,j)  -  temp * mz
        v(1,j) = vh(1,j)  -  temp * mu
        v(2,j) = vh(2,j)  -  temp * mv
        v(3,j) = vh(3,j)  -  temp * mw
        mx = mx  +  m(j) * xh(1,j)
        my = my  +  m(j) * xh(2,j)
        mz = mz  +  m(j) * xh(3,j)
        mu = mu  +  m(j) * vh(1,j)
        mv = mv  +  m(j) * vh(2,j)
        mw = mw  +  m(j) * vh(3,j)
      enddo
c
      if (nbig.gt.2) then
        temp = 1.d0 / (mtot + m(1))
        x(1,nbig) = xh(1,nbig)  -  temp * mx
        x(2,nbig) = xh(2,nbig)  -  temp * my
        x(3,nbig) = xh(3,nbig)  -  temp * mz
        v(1,nbig) = vh(1,nbig)  -  temp * mu
        v(2,nbig) = vh(2,nbig)  -  temp * mv
        v(3,nbig) = vh(3,nbig)  -  temp * mw
```

```
      end if
c
      do j = nbig + 1, nbod
        x(1,j) = xh(1,j)
        x(2,j) = xh(2,j)
        x(3,j) = xh(3,j)
        v(1,j) = vh(1,j)
        v(2,j) = vh(2,j)
        v(3,j) = vh(3,j)
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_IDEN.FOR    (ErikSoft   2 November 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Makes a new copy of a set of coordinates.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_iden (jcen,nbod,nbig,h,m,xh,vh,x,v)
c
      implicit none
c
c Input/Output
      integer nbod,nbig
      real*8 jcen(3),h,m(nbod),x(3,nbod),v(3,nbod),xh(3,nbod),vh(3,nbod)
c
c Local
      integer j
c
c-----------------------------------------------------------------------------
c
      do j = 1, nbod
        x(1,j) = xh(1,j)
        x(2,j) = xh(2,j)
        x(3,j) = xh(3,j)
        v(1,j) = vh(1,j)
        v(2,j) = vh(2,j)
        v(3,j) = vh(3,j)
      enddo
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_X2EL.FOR    (ErikSoft  20 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates Keplerian orbital elements given relative coordinates and
c velocities, and GM = G times the sum of the masses.
c
c The elements are: q = perihelion distance
c                   e = eccentricity
c                   i = inclination
c                   p = longitude of perihelion (NOT argument of perihelion!!)
c                   n = longitude of ascending node
c                   l = mean anomaly (or mean longitude if e < 1.e-8)
```

```
c
c-----------------------------------------------------------------------------
c
      subroutine mco_x2el (gm,x,y,z,u,v,w,q,e,i,p,n,l)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      real*8 gm,q,e,i,p,n,l,x,y,z,u,v,w
c
c Local
      real*8 hx,hy,hz,h2,h,v2,r,rv,s,true
      real*8 ci,to,temp,tmp2,bige,f,cf,ce
c
c-----------------------------------------------------------------------------
c
      hx = y * w  -  z * v
      hy = z * u  -  x * w
      hz = x * v  -  y * u
      h2 = hx*hx + hy*hy + hz*hz
      v2 = u * u  +  v * v  +  w * w
      rv = x * u  +  y * v  +  z * w
      r = sqrt(x*x + y*y + z*z)
      h = sqrt(h2)
      s = h2 / gm
c
c Inclination and node
      ci = hz / h
      if (abs(ci).lt.1) then
        i = acos (ci)
        n = atan2 (hx,-hy)
        if (n.lt.0) n = n + TWOPI
      else
        if (ci.gt.0) i = 0.d0
        if (ci.lt.0) i = PI
        n = 0.d0
      end if
c
c Eccentricity and perihelion distance
      temp = 1.d0  +  s * (v2 / gm  -  2.d0 / r)
      if (temp.le.0) then
        e = 0.d0
      else
        e = sqrt (temp)
      end if
      q = s / (1.d0 + e)
c
c True longitude
      if (hy.ne.0) then
        to = -hx/hy
        temp = (1.d0 - ci) * to
        tmp2 = to * to
        true = atan2((y*(1.d0+tmp2*ci)-x*temp),(x*(tmp2+ci)-y*temp))
      else
        true = atan2(y * ci, x)
      end if
      if (ci.lt.0) true = true + PI
c
      if (e.lt.3.d-8) then
        p = 0.d0
        l = true
      else
        ce = (v2*r - gm) / (e*gm)
c
c Mean anomaly for ellipse
        if (e.lt.1) then
          if (abs(ce).gt.1) ce = sign(1.d0,ce)
          bige = acos(ce)
          if (rv.lt.0) bige = TWOPI - bige
          l = bige - e*sin(bige)
        else
```

```
c
c Mean anomaly for hyperbola
          if (ce.lt.1) ce = 1.d0
          bige = log( ce + sqrt(ce*ce-1.d0) )
          if (rv.lt.0) bige = - bige
          l = e*sinh(bige) - bige
        end if
c
c Longitude of perihelion
        cf = (s - r) / (e*r)
        if (abs(cf).gt.1) cf = sign(1.d0,cf)
        f = acos(cf)
        if (rv.lt.0) f = TWOPI - f
        p = true - f
        p = mod (p + TWOPI + TWOPI, TWOPI)
      end if
c
      if (l.lt.0.and.e.lt.1) l = l + TWOPI
      if (l.gt.TWOPI.and.e.lt.1) l = mod (l, TWOPI)
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_JD_Y.FOR    (ErikSoft  2 June 1998)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts from Julian day number to Julian/Gregorian Calendar dates, assuming
c the dates are those used by the English calendar.
c
c Algorithm taken from 'Practical Astronomy with your calculator' (1988)
c by Peter Duffett-Smith, 3rd edition, C.U.P.
c
c Algorithm for negative Julian day numbers (Julian calendar assumed) by
c J. E. Chambers.
c
c N.B. The output date is with respect to the Julian Calendar on or before
c ===  4th October 1582, and with respect to the Gregorian Calendar on or
c      after 15th October 1582.
c
c
c-----------------------------------------------------------------------
c
      subroutine mio_jd_y (jd0,year,month,day)
c
      implicit none
c
c Input/Output
      real*8 jd0,day
      integer year,month
c
c Local
      integer i,a,b,c,d,e,g
      real*8 jd,f,temp,x,y,z
c
c-----------------------------------------------------------------------
c
      if (jd0.le.0) goto 50
c
      jd = jd0 + 0.5d0
      i = sign( dint(dabs(jd)), jd )
      f = jd - 1.d0*i
c
c If on or after 15th October 1582
      if (i.gt.2299160) then
        temp = (1.d0*i-1867216.25d0) / 36524.25d0
```

```
        a = sign( dint(dabs(temp)), temp )
        temp = .25d0 * a
        b = i + 1 + a - sign( dint(dabs(temp)), temp )
      else
        b = i
      end if
c
      c = b + 1524
      temp = (1.d0*c - 122.1d0) / 365.25d0
      d = sign( dint(dabs(temp)), temp )
      temp = 365.25d0 * d
      e = sign( dint(dabs(temp)), temp )
      temp = (c-e) / 30.6001d0
      g = sign( dint(dabs(temp)), temp )
c
      temp = 30.6001d0 * g
      day = 1.d0*(c-e) + f - 1.d0*sign( dint(dabs(temp)), temp )
c
      if (g.le.13) month = g - 1
      if (g.gt.13) month = g - 13
c
      if (month.gt.2) year = d - 4716
      if (month.le.2) year = d - 4715
c
      if (day.gt.32) then
        day = day - 32
        month = month + 1
      end if
c
      if (month.gt.12) then
        month = month - 12
        year = year + 1
      end if
      return
c
  50  continue
c
c Algorithm for negative Julian day numbers (Duffett-Smith won't work)
      x = jd0 - 2232101.5
      f = x - dint(x)
      if (f.lt.0) f = f + 1.d0
      y = dint(mod(x,1461.d0) + 1461.d0)
      z = dint(mod(y,365.25d0))
      month = int((z + 0.5d0) / 30.61d0)
      day = dint(z + 1.5d0 - 30.61d0*dble(month)) + f
      month = mod(month + 2, 12) + 1
c
      year = 1399 + int (x / 365.25d0)
      if (x.lt.0) year = year - 1
      if (month.lt.3) year = year + 1
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_SPL.FOR    (ErikSoft  14 November 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Given a character string STRING, of length LEN bytes, the routine finds
c the beginnings and ends of NSUB substrings present in the original, and
c delimited by spaces. The positions of the extremes of each substring are
c returned in the array DELIMIT.
c Substrings are those which are separated by spaces or the = symbol.
c
c-----------------------------------------------------------------------
c
```

```
      subroutine mio_spl (len,string,nsub,delimit)
c
      implicit none
c
c Input/Output
      integer len,nsub,delimit(2,100)
      character*1 string(len)
c
c Local
      integer j,k
      character*1 c
c
c-----------------------------------------------------------------------
c
      nsub = 0
      j = 0
      c = ' '
      delimit(1,1) = -1
c
c Find the start of string
   10 j = j + 1
      if (j.gt.len) goto 99
      c = string(j)
      if (c.eq.' '.or.c.eq.'=') goto 10
c
c Find the end of string
      k = j
   20 k = k + 1
      if (k.gt.len) goto 30
      c = string(k)
      if (c.ne.' '.and.c.ne.'=') goto 20
c
c Store details for this string
   30 nsub = nsub + 1
      delimit(1,nsub) = j
      delimit(2,nsub) = k - 1
c
      if (k.lt.len) then
        j = k
        goto 10
      end if
c
   99 continue
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MXX_SORT.FOR    (ErikSoft 24 May 1997)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Sorts an array X, of size N, using Shell's method. Also returns an array
c INDEX that gives the original index of every item in the sorted array X.
c
c N.B. The maximum array size is 29523.
c ===
c
c-----------------------------------------------------------------------
c
      subroutine mxx_sort (n,x,index)
c
      implicit none
c
c Input/Output
      integer n,index(n)
      real*8 x(n)
```

```
c
c Local
      integer i,j,k,l,m,inc,incarr(9),iy
      real*8 y
      data incarr/1,4,13,40,121,364,1093,3280,9841/
c
c-----------------------------------------------------------------------------
c
      do i = 1, n
        index(i) = i
      end do
c
      m = 0
  10  m = m + 1
      if (incarr(m).lt.n) goto 10
      m = m - 1
c
      do i = m, 1, -1
        inc = incarr(i)
        do j = 1, inc
          do k = inc, n - j, inc
            y = x(j+k)
            iy = index(j+k)
            do l = j + k - inc, j, -inc
              if (x(l).le.y) goto 20
              x(l+inc) = x(l)
              index(l+inc) = index(l)
            end do
  20        x(l+inc) = y
            index(l+inc) = iy
          end do
        end do
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      M_SFUNC.FOR     (ErikSoft  14 November 1998)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Calculates Z = [ (3 + S^2)arctan(S) - 3S ] / S^3 and its derivative DZ,
c for S > 0.
c
c-----------------------------------------------------------------------------
c
      subroutine m_sfunc (s,z,dz)
c
      implicit none
c
c Input/Output
      real*8 s, z, dz
c
c Local
      real*8 s2,s4,s6,s8,a
c
c-----------------------------------------------------------------------------
c
      s2 = s * s
c
      if (s.gt.1.d-2) then
        a  = atan(s)
        z  = ((3.d0 + s2)*a - 3.d0*s) / (s * s2)
        dz = (2.d0*s*a - 3.d0 + (3.d0+s2)/(1.d0+s2)) / (s * s2)
     %     - 3.d0 * z / s
      else
        s4 = s2 * s2
        s6 = s2 * s4
```

```
         s8 = s4 * s4
         z  = - .1616161616161616d0*s8
     %        + .1904761904761905d0*s6
     %        - .2285714285714286d0*s4
     %        + .2666666666666667d0*s2
         dz = s * (- 1.292929292929293d0*s6
     %             + 1.142857142857143d0*s4
     %             - 0.914285714285714d0*s2
     %             + 0.5333333333333333d0)
       end if
c
c-----------------------------------------------------------------------------
c
       return
       end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      M_FORMAT.FOR    (ErikSoft   31 January 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Makes an output format list and file header for the orbital-element files
c created by M_ELEM3.FOR
c Also identifies which orbital elements will be output for each object.
c
c-----------------------------------------------------------------------------
c
       subroutine m_format (string,timestyle,nel,iel,fout,header,lenhead)
c
       implicit none
       include 'mercury.inc'
c
c Input/Output
       integer timestyle,nel,iel(22),lenhead
       character*250 string,header,fout
c
c Local
       integer i,j,pos,nsub,lim(2,20),formflag,lenfout,f1,f2,itmp
       character*1 elcode(22)
       character*4 elhead(22)
c
c-----------------------------------------------------------------------------
c
       data elcode/ 'a','e','i','g','n','l','p','q','b','x','y','z',
     %  'u','v','w','r','f','m','o','s','d','c'/
       data elhead/ '  a ','  e ','  i ','peri','node','  M ','long',
     %  '  q ','  Q ','  x ','  y ','  z ',' vx ',' vy ',' vz ','  r ',
     %  '  f ','mass','oblq','spin','dens','comp'/
c
c Initialize header to a blank string
       do i = 1, 250
         header(i:i) = ' '
       end do
c
c Create part of the format list and header for the required time style
       if (timestyle.eq.0.or.timestyle.eq.2) then
         fout(1:9) = '(1x,f18.5'
         lenfout = 9
         header(1:19) = '    Time (days)    '
         lenhead = 19
       else if (timestyle.eq.1) then
         fout(1:21) = '(1x,i10,1x,i2,1x,f8.5'
         lenfout = 21
         header(1:23) = '    Year/Month/Day     '
         lenhead = 23
       else if (timestyle.eq.3) then
         fout(1:9) = '(1x,f18.7'
         lenfout = 9
         header(1:19) = '    Time (years)   '
```

```
        lenhead = 19
      end if
c
c Identify the required elements
      call mio_spl (250,string,nsub,lim)
      do i = 1, nsub
        do j = 1, 22
          if (string(lim(1,i):lim(1,i)).eq.elcode(j)) iel(i) = j
        end do
      end do
      nel = nsub
c
c For each element, see whether normal or exponential notation is required
      do i = 1, nsub
        formflag = 0
        do j = lim(1,i)+1, lim(2,i)
          if (formflag.eq.0) pos = j
          if (string(j:j).eq.'.') formflag = 1
          if (string(j:j).eq.'e') formflag = 2
        end do
c
c Create the rest of the format list and header
        if (formflag.eq.1) then
          read (string(lim(1,i)+1:pos-1),*) f1
          read (string(pos+1:lim(2,i)),*) f2
          write (fout(lenfout+1:lenfout+10),'(a10)') ',1x,f  .  '
          write (fout(lenfout+6:lenfout+7),'(i2)') f1
          write (fout(lenfout+9:lenfout+10),'(i2)') f2
          lenfout = lenfout + 10
        else if (formflag.eq.2) then
          read (string(lim(1,i)+1:pos-1),*) f1
          write (fout(lenfout+1:lenfout+16),'(a16)') ',1x,1p,e  .  ,0p'
          write (fout(lenfout+9:lenfout+10),'(i2)') f1
          write (fout(lenfout+12:lenfout+13),'(i2)') f1 - 7
          lenfout = lenfout + 16
        end if
        itmp = (f1 - 4) / 2
        header(lenhead+itmp+2:lenhead+itmp+5) = elhead(iel(i))
        lenhead = lenhead + f1 + 1
      end do
c
      lenfout = lenfout + 1
      fout(lenfout:lenfout) = ')'
c
c-----------------------------------------------------------------------------
c
      return
      end
c
************************************************************************
c                     ORBEL_FHYBRID.F
************************************************************************
*     PURPOSE:  Solves Kepler's eqn. for hyperbola using hybrid approach.
*
*             Input:
*                           e ==> eccentricity anomaly. (real scalar)
*                           n ==> hyperbola mean anomaly. (real scalar)
*             Returns:
*               orbel_fhybrid ==>  eccentric anomaly. (real scalar)
*
*     ALGORITHM: For abs(N) < 0.636*ecc -0.6 , use FLON
*                For larger N, uses FGET
*     REMARKS:
*     AUTHOR: M. Duncan
*     DATE WRITTEN: May 26,1992.
*     REVISIONS:
*     REVISIONS: 2/26/93 hfl
************************************************************************

        real*8 function orbel_fhybrid(e,n)

      include 'swift.inc'
```

```
c...  Inputs Only:
        real*8 e,n

c...  Internals:
        real*8 abn
        real*8 orbel_flon,orbel_fget

c----
c...  Executable code

        abn = n
        if(n.lt.0.d0) abn = -abn

        if(abn .lt. 0.636d0*e -0.6d0) then
          orbel_fhybrid = orbel_flon(e,n)
        else
          orbel_fhybrid = orbel_fget(e,n)
        endif

        return
        end  ! orbel_fhybrid
c-----------------------------------------------------------------------
c
*************************************************************************
c                    ORBEL_FGET.F
*************************************************************************
*     PURPOSE:  Solves Kepler's eqn. for hyperbola using hybrid approach.
*
*             Input:
*                           e ==> eccentricity anomaly. (real scalar)
*                        capn ==> hyperbola mean anomaly. (real scalar)
*             Returns:
*                  orbel_fget ==>  eccentric anomaly. (real scalar)
*
*     ALGORITHM: Based on pp. 70-72 of Fitzpatrick's book "Principles of
*           Cel. Mech. ".  Quartic convergence from Danby's book.
*     REMARKS:
*     AUTHOR: M. Duncan
*     DATE WRITTEN: May 11, 1992.
*     REVISIONS: 2/26/93 hfl
c     Modified by JEC
*************************************************************************

        real*8 function orbel_fget(e,capn)

      include 'swift.inc'

c...  Inputs Only:
        real*8 e,capn

c...  Internals:
        integer i,IMAX
        real*8 tmp,x,shx,chx
        real*8 esh,ech,f,fp,fpp,fppp,dx
        PARAMETER (IMAX = 10)

c----
c...  Executable code

c Function to solve "Kepler's eqn" for F (here called
c x) for given e and CAPN.

c  begin with a guess proposed by Danby
        if( capn .lt. 0.d0) then
           tmp = -2.d0*capn/e + 1.8d0
           x = -log(tmp)
        else
           tmp = +2.d0*capn/e + 1.8d0
           x = log( tmp)
        endif
```

```
        orbel_fget = x

        do i = 1,IMAX
          call mco_sinh (x,shx,chx)
          esh = e*shx
          ech = e*chx
          f = esh - x - capn
c         write(6,*) 'i,x,f : ',i,x,f
          fp = ech - 1.d0
          fpp = esh
          fppp = ech
          dx = -f/fp
          dx = -f/(fp + dx*fpp/2.d0)
          dx = -f/(fp + dx*fpp/2.d0 + dx*dx*fppp/6.d0)
          orbel_fget = x + dx
c   If we have converged here there's no point in going on
          if(abs(dx) .le. TINY) RETURN
          x = orbel_fget
        enddo

        write(6,*) 'FGET : RETURNING WITHOUT COMPLETE CONVERGENCE'
        return
        end   ! orbel_fget
c-----------------------------------------------------------------------
c
*************************************************************************
c                    ORBEL_FLON.F
*************************************************************************
*     PURPOSE:  Solves Kepler's eqn. for hyperbola using hybrid approach.
*
*             Input:
*                         e ==> eccentricity anomaly. (real scalar)
*                      capn ==> hyperbola mean anomaly. (real scalar)
*             Returns:
*                  orbel_flon ==>  eccentric anomaly. (real scalar)
*
*     ALGORITHM: Uses power series for N in terms of F and Newton,s method
*     REMARKS: ONLY GOOD FOR LOW VALUES OF N (N < 0.636*e -0.6)
*     AUTHOR: M. Duncan
*     DATE WRITTEN: May 26, 1992.
*     REVISIONS:
*************************************************************************

        real*8 function orbel_flon(e,capn)

      include 'swift.inc'

c...  Inputs Only:
        real*8 e,capn

c...  Internals:
        integer iflag,i,IMAX
        real*8 a,b,sq,biga,bigb
        real*8 x,x2
        real*8 f,fp,dx
        real*8 diff
        real*8 a0,a1,a3,a5,a7,a9,a11
        real*8 b1,b3,b5,b7,b9,b11
        PARAMETER (IMAX = 10)
        PARAMETER (a11 = 156.d0,a9 = 17160.d0,a7 = 1235520.d0)
        PARAMETER (a5 = 51891840.d0,a3 = 1037836800.d0)
        PARAMETER (b11 = 11.d0*a11,b9 = 9.d0*a9,b7 = 7.d0*a7)
        PARAMETER (b5 = 5.d0*a5, b3 = 3.d0*a3)

c----
c...  Executable code


c Function to solve "Kepler's eqn" for F (here called
c x) for given e and CAPN. Only good for smallish CAPN

        iflag = 0
```

```
        if( capn .lt. 0.d0) then
           iflag = 1
           capn = -capn
        endif

        a1 = 6227020800.d0 * (1.d0 - 1.d0/e)
        a0 = -6227020800.d0*capn/e
        b1 = a1

c  Set iflag nonzero if capn < 0., in which case solve for -capn
c  and change the sign of the final answer for F.
c  Begin with a reasonable guess based on solving the cubic for small F


        a = 6.d0*(e-1.d0)/e
        b = -6.d0*capn/e
        sq = sqrt(0.25*b*b +a*a*a/27.d0)
        biga = (-0.5*b + sq)**0.3333333333333333d0
        bigb = -(+0.5*b + sq)**0.3333333333333333d0
        x = biga + bigb
c       write(6,*) 'cubic = ',x**3 +a*x +b
        orbel_flon = x
c If capn is tiny (or zero) no need to go further than cubic even for
c e =1.
        if( capn .lt. TINY) go to 100

        do i = 1,IMAX
          x2 = x*x
          f = a0 +x*(a1+x2*(a3+x2*(a5+x2*(a7+x2*(a9+x2*(a11+x2))))))
          fp = b1 +x2*(b3+x2*(b5+x2*(b7+x2*(b9+x2*(b11 + 13.d0*x2)))))
          dx = -f/fp
c         write(6,*) 'i,dx,x,f : '
c         write(6,432) i,dx,x,f
432       format(1x,i3,3(2x,1p1e22.15))
          orbel_flon = x + dx
c   If we have converged here there's no point in going on
          if(abs(dx) .le. TINY) go to 100
          x = orbel_flon
        enddo

c Abnormal return here - we've gone thru the loop
c IMAX times without convergence
        if(iflag .eq. 1) then
           orbel_flon = -orbel_flon
           capn = -capn
        endif
        write(6,*) 'FLON : RETURNING WITHOUT COMPLETE CONVERGENCE'
          diff = e*sinh(orbel_flon) - orbel_flon - capn
          write(6,*) 'N, F, ecc*sinh(F) - F - N : '
          write(6,*) capn,orbel_flon,diff
        return

c  Normal return here, but check if capn was originally negative
100     if(iflag .eq. 1) then
           orbel_flon = -orbel_flon
           capn = -capn
        endif

        return
        end     ! orbel_flon
c-----------------------------------------------------------------------
c
************************************************************************
c                    ORBEL_ZGET.F
************************************************************************
*     PURPOSE:  Solves the equivalent of Kepler's eqn. for a parabola
*          given Q (Fitz. notation.)
*
*             Input:
*                          q ==>  parabola mean anomaly. (real scalar)
*             Returns:
*                  orbel_zget ==>  eccentric anomaly. (real scalar)
```

```
*
*     ALGORITHM: p. 70-72 of Fitzpatrick's book "Princ. of Cel. Mech."
*     REMARKS: For a parabola we can solve analytically.
*     AUTHOR: M. Duncan
*     DATE WRITTEN: May 11, 1992.
*     REVISIONS: May 27 - corrected it for negative Q and use power
*             series for small Q.
**********************************************************************

        real*8 function orbel_zget(q)

      include 'swift.inc'

c...  Inputs Only:
        real*8 q

c...  Internals:
        integer iflag
        real*8 x,tmp

c----
c...  Executable code

        iflag = 0
        if(q.lt.0.d0) then
          iflag = 1
          q = -q
        endif

        if (q.lt.1.d-3) then
           orbel_zget = q*(1.d0 - (q*q/3.d0)*(1.d0 -q*q))
        else
           x = 0.5d0*(3.d0*q + sqrt(9.d0*(q**2) +4.d0))
           tmp = x**(1.d0/3.d0)
           orbel_zget = tmp - 1.d0/tmp
        endif

        if(iflag .eq.1) then
           orbel_zget = -orbel_zget
           q = -q
        endif

        return
        end    ! orbel_zget
c----------------------------------------------------------------------
```