

```

c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      CLOSE6.FOR      (ErikSoft   5 June 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Makes output files containing details of close encounters that occurred
c during an integration using Mercury6 or higher.
c
c The user specifies the names of the required objects in the file close.in
c
c-----
c
c      implicit none
c      include 'mercury.inc'
c
c      integer itmp,i,j,k,l,iclo,jclo,precision,lenin
c      integer nmaster,nopen,nwait,nbig,nsml,nsup,lim(2,100)
c      integer year,month,timestyle,line_num,lenhead,lmem(NMESS)
c      integer nchar,algor,allflag,firstflag,ninfile
c      integer unit(NMAX),master_unit(NMAX)
c      real*8 time,t0,t1,rmax,rcen,rfac,dclo,mcen,jcen(3)
c      real*8 mio_c2re, mio_c2fl,fr,theta,phi,fv,vtheta,vphi,gm
c      real*8 x1(3),x2(3),v1(3),v2(3),m(NMAX)
c      real*8 a1,a2,e1,e2,i1,i2,p1,p2,n1,n2,l1,l2,q1,q2
c      logical test
c      character*250 string,fout,header,infile(50)
c      character*80 mem(NMESS),cc,c(NMAX)
c      character*8 master_id(NMAX),id(NMAX)
c      character*5 fin
c      character*1 check,style,type,c1
c
c-----
c
c      allflag = 0
c
c Read in output messages
c      inquire (file='message.in', exist=test)
c      if (.not.test) then
c          write (*,'(/,2a)') ' ERROR: This file is needed to continue: ',
%      ' message.in'
c          stop
c      end if
c      open (14, file='message.in', status='old')
10  continue
c      read (14,'(i3,1x,i2,1x,a80)',end=20) j,lmem(j),mem(j)
c      goto 10
20  close (14)
c
c Open file containing parameters for this programme
c      inquire (file='close.in', exist=test)
c      if (test) then
c          open (10, file='close.in', status='old')
c      else
c          call mio_err (6,mem(81),lmem(81),mem(88),lmem(88),' ',1,
%      'close.in',9)
c      end if
c
c Read number of input files
c      30 read (10,'(a250)') string
c          if (string(1:1).eq.'') goto 30
c          call mio_spl (250,string,nsup,lim)
c          read (string(lim(1,nsup):lim(2,nsup)),*) ninfile
c
c Make sure all the input files exist
c      do j = 1, ninfile
c          40 read (10,'(a250)') string
c              if (string(1:1).eq.'') goto 40
c              call mio_spl (250,string,nsup,lim)
c              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 Read parameters used by this programme
        timestyle = 1
        do j = 1, 2
50      read (10,'(a250)') string
          if (string(1:1).eq.'') goto 50
          call mio_spl (250,string,nsub,lim)
          cl = string(lim(1,nsub):lim(2,nsub))
          if (j.eq.1.and.(cl.eq.'d'.or.cl.eq.'D')) timestyle = 0
          if (j.eq.2.and.(cl.eq.'y'.or.cl.eq.'Y')) timestyle = timestyle+2
        end do
c
c Read in the names of the objects for which orbital elements are required
        nopen = 0
        nwait = 0
        nmaster = 0
        call m_formce (timestyle,fout,header,lenhead)
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),'.clo',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 CLOSE.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',
%       '
          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))
      if (firstflag.eq.0) then
        t0 = time
        firstflag = 1
      end if
      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)) * K2
      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)
        m(k) = mio_c2fl (c(j)(12:19)) * K2
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),'.clo',
                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

```

```

        end do
C
C-----
C
C IF NORMAL INPUT, READ COMPRESSED DATA ON THE CLOSE ENCOUNTER
C
        else if (type.eq.'b') then
            line_num = line_num + 1
            read (10,'(3x,a70)',err=666) cc(1:70)
C
C Decompress time, distance and orbital variables for each object
        time = mio_c2fl (cc(1:8))
        iclo = int(.5d0 + mio_c2re(cc(9:16), 0.d0, 11239424.d0, 3))
        jclo = int(.5d0 + mio_c2re(cc(12:19), 0.d0, 11239424.d0, 3))
        if (iclo.gt.NMAX.or.jclo.gt.NMAX) then
            write (*,'(/,2a)') mem(81)(1:lmem(81)),
%             mem(90)(1:lmem(90))
            stop
        end if
        dclo = mio_c2fl (cc(15:22))
        fr = mio_c2re (cc(23:30), 0.d0, rfac, 4)
        theta = mio_c2re (cc(27:34), 0.d0, PI, 4)
        phi = mio_c2re (cc(31:38), 0.d0, TWOPI, 4)
        fv = mio_c2re (cc(35:42), 0.d0, 1.d0, 4)
        vtheta = mio_c2re (cc(39:46), 0.d0, PI, 4)
        vphi = mio_c2re (cc(43:50), 0.d0, TWOPI, 4)
        call mco_ov2x (rcen,rmax,mcen,m(iclo),fr,theta,phi,fv,
%             vtheta,vphi,x1(1),x1(2),x1(3),v1(1),v1(2),v1(3))
C
        fr = mio_c2re (cc(47:54), 0.d0, rfac, 4)
        theta = mio_c2re (cc(51:58), 0.d0, PI, 4)
        phi = mio_c2re (cc(55:62), 0.d0, TWOPI, 4)
        fv = mio_c2re (cc(59:66), 0.d0, 1.d0, 4)
        vtheta = mio_c2re (cc(63:70), 0.d0, PI, 4)
        vphi = mio_c2re (cc(67:74), 0.d0, TWOPI, 4)
        call mco_ov2x (rcen,rmax,mcen,m(jclo),fr,theta,phi,fv,
%             vtheta,vphi,x2(1),x2(2),x2(3),v2(1),v2(2),v2(3))
C
C Convert to Keplerian elements
        gm = mcen + m(iclo)
        call mco_x2el (gm,x1(1),x1(2),x1(3),v1(1),v1(2),v1(3),
%             q1,e1,i1,p1,n1,l1)
        a1 = q1 / (1.d0 - e1)
        gm = mcen + m(jclo)
        call mco_x2el (gm,x2(1),x2(2),x2(3),v2(1),v2(2),v2(3),
%             q2,e2,i2,p2,n2,l2)
        a2 = q2 / (1.d0 - e2)
        i1 = i1 / DR
        i2 = i2 / DR
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 Write encounter details to appropriate files
        if (timestyle.eq.1) then
            if (unit(iclo).ge.10) write (unit(iclo),fout) year,month,
%             t1,id(jclo),dclo,a1,e1,i1,a2,e2,i2
C
            if (unit(jclo).ge.10) write (unit(jclo),fout) year,month,
%             t1,id(iclo),dclo,a2,e2,i2,a1,e1,i1
        else
            if (unit(iclo).ge.10) write (unit(iclo),fout) t1,id(jclo),
%             dclo,a1,e1,i1,a2,e2,i2
            if (unit(jclo).ge.10) write (unit(jclo),fout) t1,id(iclo),
%             dclo,a2,e2,i2,a1,e1,i1
        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 close encounter data
900  continue
      close (10)
      end do
c
c Close clo 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),'clo',master_unit(j),header,
%              lenhead,mem,lmem)
          end if
        end do
        goto 90
      end if
c
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      M_FORMCE.FOR      (ErikSoft  30 November 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c
c-----
c
      subroutine m_formce (timestyle,fout,header,lenhead)
c
      implicit none
c
c Input/Output
      integer timestyle,lenhead
      character*250 fout,header
c
c-----
c
      if (timestyle.eq.0.or.timestyle.eq.2) then
        header(1:19) = '      Time (days)      '

```

```

      header(20:58) = '   Object   dmin (AU)      a1      e1      '
      header(59:90) = '      i1      a2      e2      i2'
      lenhead = 90
      fout = '(1x,f18.5,1x,a8,1x,f10.8,2(1x,f9.4,1x,f8.6,1x,f7.3))'
    else
      if (timestyle.eq.1) then
        header(1:23) = '          Year/Month/Day      '
        header(24:62) = '   Object   dmin (AU)      a1      e1      '
        header(63:94) = '      i1      a2      e2      i2'
        lenhead = 94
        fout(1:37) = '(1x,i10,1x,i2,1x,f8.5,1x,a8,1x,f10.8,'
        fout(38:64) = '2(1x,f9.4,1x,f8.6,1x,f7.3))'
      else
        header(1:19) = '      Time (years)      '
        header(20:58) = '   Object   dmin (AU)      a1      e1      '
        header(59:90) = '      i1      a2      e2      i2'
        fout = '(1x,f18.7,1x,a8,1x,f10.8,2(1x,f9.4,1x,f8.6,1x,f7.3))'
        lenhead = 90
      end if
    end if
  end if

c
c-----
c
      return
    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 Input/Output
      real*8 rcen,rmax,mcen,m,x,y,z,u,v,w,fr,theta,phi,fv,vtheta,vphi

c Local
      real*8 r,v1,temp

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
c      return
c      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
c      subroutine mco_el2x (mu,q,e,i,p,n,l,x,y,z,u,v,w)
c
c      implicit none
c      include 'mercury.inc'
c
c Input/Output
c      real*8 mu,q,e,i,p,n,l,x,y,z,u,v,w
c
c Local
c      real*8 g,a,ci,si,cn,sn,cg,sg,se,se,romes,temp
c      real*8 z1,z2,z3,z4,d11,d12,d13,d21,d22,d23
c      real*8 mco_kep, orbel_fhybrid, orbel_zget
c
c-----
c
c Change from longitude of perihelion to argument of perihelion
c      g = p - n
c
c Rotation factors
c      call mco_sine (i,si,ci)
c      call mco_sine (g,sg,cg)
c      call mco_sine (n,sn,cn)
c      z1 = cg * cn
c      z2 = cg * sn
c      z3 = sg * cn
c      z4 = sg * sn
c      d11 = z1 - z4*ci
c      d12 = z2 + z3*ci
c      d13 = sg * si
c      d21 = -z3 - z2*ci
c      d22 = -z4 + z1*ci
c      d23 = cg * si
c
c Semi-major axis
c      a = q / (1.d0 - e)
c
c Ellipse
c      if (e.lt.1.d0) then
c          romes = sqrt(1.d0 - e*e)
c          temp = mco_kep (e,l)
c          call mco_sine (temp,se,ce)
c          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(1)
        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,1)
        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 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 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.pi/2
  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 = 1 + 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.pi/2)
  if (bigg) then
    z3 = pi - u2

```

```

        else
            z3 = u2
        end if
C
big = (z3.gt(.5d0*piy2))
if (big) then
    x = piy2 - 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 - piy2
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 = 1 - 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,nsup,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,nsup,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 Input/output
      integer nchar
      real*8 xmin,xmax,mio_c2re
      character*8 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

      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,ls1,ls1,s2,ls2,s3,ls3,s4,ls4)

c
      implicit none

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
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
c      return
c      end

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
c      implicit none
c
c Input/Output
c      integer nbod,nbig
c      real*8 jcen(3),h,m(nbig),xh(3,nbig),vh(3,nbig),x(3,nbig),v(3,nbig)
c
c Local
c      integer j
c      real*8 mtot, mx, my, mz, mu, mv, mw, temp
c-----c
c      mtot = m(2)
c      x(1,2) = xh(1,2)
c      x(2,2) = xh(2,2)
c      x(3,2) = xh(3,2)
c      v(1,2) = vh(1,2)
c      v(2,2) = vh(2,2)
c      v(3,2) = vh(3,2)
c      mx = m(2) * xh(1,2)
c      my = m(2) * xh(2,2)
c      mz = m(2) * xh(3,2)
c      mu = m(2) * vh(1,2)
c      mv = m(2) * vh(2,2)
c      mw = m(2) * vh(3,2)
c
c
c      do j = 3, nbig - 1
c          temp = 1.d0 / (mtot + m(1))
c          mtot = mtot + m(j)
c          x(1,j) = xh(1,j) - temp * mx
c          x(2,j) = xh(2,j) - temp * my
c          x(3,j) = xh(3,j) - temp * mz
c          v(1,j) = vh(1,j) - temp * mu
c          v(2,j) = vh(2,j) - temp * mv
c          v(3,j) = vh(3,j) - temp * mw
c          mx = mx + m(j) * xh(1,j)
c          my = my + m(j) * xh(2,j)
c          mz = mz + m(j) * xh(3,j)
c          mu = mu + m(j) * vh(1,j)
c          mv = mv + m(j) * vh(2,j)
c          mw = mw + m(j) * vh(3,j)
c      enddo
c
c      if (nbig.gt.2) then
c          temp = 1.d0 / (mtot + m(1))
c          x(1,nbig) = xh(1,nbig) - temp * mx
c          x(2,nbig) = xh(2,nbig) - temp * my
c          x(3,nbig) = xh(3,nbig) - temp * mz
c          v(1,nbig) = vh(1,nbig) - temp * mu
c          v(2,nbig) = vh(2,nbig) - temp * mv
c          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
      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

```

```

        subroutine mio_spl (len,string,nsub,delimit)
c
c      implicit none
c
c Input/Output
c      integer len,nsub,delimit(2,100)
c      character*1 string(len)
c
c Local
c      integer j,k
c      character*1 c
c
c-----
c
c      nsub = 0
c      j = 0
c      c = ' '
c      delimit(1,1) = -1
c
c Find the start of string
c      10 j = j + 1
c         if (j.gt.len) goto 99
c         c = string(j)
c         if (c.eq.' ' .or. c.eq.'=') goto 10
c
c Find the end of string
c      k = j
c      20 k = k + 1
c         if (k.gt.len) goto 30
c         c = string(k)
c         if (c.ne.' ' .and. c.ne.'=') goto 20
c
c Store details for this string
c      30 nsub = nsub + 1
c         delimit(1,nsub) = j
c         delimit(2,nsub) = k - 1
c
c         if (k.lt.len) then
c             j = k
c             goto 10
c         end if
c
c      99 continue
c
c-----
c
c      return
c      end
c
c *****
c      ORBEL_FHYBRID.F
c *****
c      PURPOSE:  Solves Kepler's eqn. for hyperbola using hybrid approach.
c
c      Input:
c
c              e ==> eccentricity anomaly. (real scalar)
c              n ==> hyperbola mean anomaly. (real scalar)
c
c      Returns:
c
c      orbelfhybrid ==>  eccentric anomaly. (real scalar)
c
c      ALGORITHM: For abs(N) < 0.636*ecc -0.6 , use FLON
c                  For larger N, uses FGET
c
c      REMARKS:
c      AUTHOR: M. Duncan
c      DATE WRITTEN: May 26,1992.
c      REVISIONS:
c      REVISIONS: 2/26/93 hfl
c *****
c
c      real*8 function orbelfhybrid(e,n)
c
c      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
*   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,chk
    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 *****
C         ORBEL_FLON.F
C *****
C         PURPOSE:  Solves Kepler's eqn. for hyperbola using hybrid approach.
C
C         Input:
C
C                 e ==> eccentricity anomaly. (real scalar)
C                 capn ==> hyperbola mean anomaly. (real scalar)
C
C         Returns:
C
C                 orbel_flon ==> eccentric anomaly. (real scalar)
C
C         ALGORITHM: Uses power series for N in terms of F and Newton's method
C         REMARKS: ONLY GOOD FOR LOW VALUES OF N (N < 0.636*e -0.6)
C         AUTHOR: M. Duncan
C         DATE WRITTEN: May 26, 1992.
C         REVISIONS:
C *****

        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 *****
C ORBEL_ZGET.F
C *****
C PURPOSE:  Solves the equivalent of Kepler's eqn. for a parabola
C           given Q (Fitz. notation.)
C
C           Input:
C
C               q ==> parabola mean anomaly. (real scalar)
C
C           Returns:
C
C               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-----

```