```
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MERCURY6_1.FOR    (ErikSoft   3 May 2002)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Mercury is a general-purpose N-body integration package for problems in
c celestial mechanics.
c
c------------------------------------------------------------------------------
c This package contains some subroutines taken from the Swift integration
c package by H.F.Levison and M.J.Duncan (1994) Icarus, vol 108, pp18.
c Routines taken from Swift have names beginning with `drift' or `orbel'.
c
c The standard symplectic (MVS) algorithm is described in J.Widsom and
c M.Holman (1991) Astronomical Journal, vol 102, pp1528.
c
c The hybrid symplectic algorithm is described in J.E.Chambers (1999)
c Monthly Notices of the RAS, vol 304, pp793.
c
c RADAU is described in E.Everhart (1985) in ``The Dynamics of Comets:
c Their Origin and Evolution'' p185-202, eds. A.Carusi & G.B.Valsecchi,
c pub. Reidel.
c
c The Bulirsch-Stoer algorithms are described in W.H.Press et al. (1992)
c ``Numerical Recipes in Fortran'', pub. Cambridge.
c------------------------------------------------------------------------------
c
c Variables:
c ---------
c  M       = mass (in solar masses)
c  XH      = coordinates (x,y,z) with respect to the central body (AU)
c  VH      = velocities (vx,vy,vz) with respect to the central body (AU/day)
c  S       = spin angular momentum (solar masses AU^2/day)
c  RHO     = physical density (g/cm^3)
c  RCEH    = close-encounter limit (Hill radii)
c  STAT    = status (0 => alive, <>0 => to be removed)
c  ID      = name of the object (8 characters)
c  CE      = close encounter status
c  NGF     = (1-3) cometary non-gravitational (jet) force parameters
c   "      =  (4)  beta parameter for radiation pressure and P-R drag
c  EPOCH  = epoch of orbit (days)
c  NBOD  = current number of bodies (INCLUDING the central object)
c  NBIG  =    "       "     " big bodies (ones that perturb everything else)
c  TIME  = current epoch (days)
c  TOUT  = time of next output evaluation
c  TDUMP = time of next data dump
c  TFUN  = time of next periodic effect (e.g. next check for ejections)
c  H     = current integration timestep (days)
c  EN(1) = initial energy of the system
c  " (2) = current    "    "  "    "
c  " (3) = energy change due to collisions, ejections etc.
c  AM(1,2,3) = as above but for angular momentum
c
c Integration Parameters :
c ----------------------
c  ALGOR = 1  ->  Mixed-variable symplectic
c          2  ->  Bulirsch-Stoer integrator
c          3  ->         "           "      (conservative systems only)
c          4  ->  RA15 `radau' integrator
c          10 ->  Hybrid MVS/BS (democratic-heliocentric coords)
c          11 ->  Close-binary hybrid (close-binary coords)
c          12 ->  Wide-binary hybrid (wide-binary coords)
c
c TSTART = epoch of first required output (days)
c TSTOP  =   "      final required output ( "  )
c DTOUT  = data output interval           ( "  )
c DTDUMP = data-dump interval             ( "  )
c DTFUN  = interval for other periodic effects (e.g. check for ejections)
c  H0    = initial integration timestep (days)
```

```
c  TOL    = Integrator tolerance parameter (approx. error per timestep)
c  RMAX   = heliocentric distance at which objects are considered ejected (AU)
c  RCEN   = radius of central body (AU)
c  JCEN(1,2,3) = J2,J4,J6 for central body (units of RCEN^i for Ji)
c
c Options:
c  OPT(1) = close-encounter option (0=stop after an encounter, 1=continue)
c  OPT(2) = collision option (0=no collisions, 1=merge, 2=merge+fragment)
c  OPT(3) = time style (0=days 1=Greg.date 2/3=days/years w/respect to start)
c  OPT(4) = o/p precision (1,2,3 = 4,9,15 significant figures)
c  OPT(5) = < Not used at present >
c  OPT(6) = < Not used at present >
c  OPT(7) = apply post-Newtonian correction? (0=no, 1=yes)
c  OPT(8) = apply user-defined force routine mfo_user? (0=no, 1=yes)
c
c File variables :
c --------------
c  OUTFILE  (1) = osculating coordinates/velocities and masses
c     "     (2) = close encounter details
c     "     (3) = information file
c  DUMPFILE (1) = Big-body data
c     "     (2) = Small-body data
c     "     (3) = integration parameters
c     "     (4) = restart file
c
c Flags :
c -----
c  NGFLAG = do any bodies experience non-grav. forces?
c                             ( 0 = no non-grav forces)
c                               1 = cometary jets only
c                               2 = radiation pressure/P-R drag only
c                               3 = both
c  OPFLAG = integration mode (-2 = synchronising epochs)
c                              -1 = integrating towards start epoch
c                               0 = main integration, normal output
c                               1 = main integration, full output
c
c-----------------------------------------------------------------------------
c
      implicit none
      include 'mercury.inc'
c
      integer j,algor,nbod,nbig,opt(8),stat(NMAX),lmem(NMESS)
      integer opflag,ngflag,ndump,nfun
      real*8 m(NMAX),xh(3,NMAX),vh(3,NMAX),s(3,NMAX),rho(NMAX)
      real*8 rceh(NMAX),epoch(NMAX),ngf(4,NMAX),rmax,rcen,jcen(3)
      real*8 cefac,time,tstart,tstop,dtout,h0,tol,en(3),am(3)
      character*8 id(NMAX)
      character*80 outfile(3), dumpfile(4), mem(NMESS)
      external mdt_mvs, mdt_bs1, mdt_bs2, mdt_ra15, mdt_hy
      external mco_dh2h,mco_h2dh
      external mco_b2h,mco_h2b,mco_h2mvs,mco_mvs2h,mco_iden
c
      data opt/0,1,1,2,0,1,0,0/
c
c-----------------------------------------------------------------------------
c
c Get initial conditions and integration parameters
      call mio_in (time,tstart,tstop,dtout,algor,h0,tol,rmax,rcen,jcen,
     %  en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,stat,id,
     %  epoch,ngf,opt,opflag,ngflag,outfile,dumpfile,lmem,mem)
c
c If this is a new integration, integrate all the objects to a common epoch.
      if (opflag.eq.-2) then
   20   open (23,file=outfile(3),status='old',access='append',err=20)
        write (23,'(/,a)') mem(55)(1:lmem(55))
        write (*,'(a)') mem(55)(1:lmem(55))
        call mxx_sync (time,tstart,h0,tol,jcen,nbod,nbig,m,xh,vh,s,rho,
     %    rceh,stat,id,epoch,ngf,opt,ngflag)
        write (23,'(/,a,/)') mem(56)(1:lmem(56))
        write (*,'(a)') mem(56)(1:lmem(56))
        opflag = -1
```

```
        close (23)
      end if
c
c Main integration
      if (algor.eq.1) call mal_hcon (time,tstart,tstop,dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,
     %  lmem,mdt_mvs,mco_h2mvs,mco_mvs2h)
c
      if (algor.eq.9) call mal_hcon (time,tstart,tstop,dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,
     %  lmem,mdt_mvs,mco_iden,mco_iden)
c
      if (algor.eq.2) call mal_hvar (time,tstart,tstop,dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,
     %  lmem,mdt_bs1)
c
      if (algor.eq.3) call mal_hvar (time,tstart,tstop,dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,
     %  lmem,mdt_bs2)
c
      if (algor.eq.4) call mal_hvar (time,tstart,tstop,dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,
     %  lmem,mdt_ra15)
c
      if (algor.eq.10) call mal_hcon (time,tstart,tstop,dtout,algor,h0,
     %  tol,jcen,rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,
     %  rho,rceh,stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,
     %  lmem,mdt_hy,mco_h2dh,mco_dh2h)
c
c Do a final data dump
      do j = 2, nbod
        epoch(j) = time
      end do
      call mio_dump (time,tstart,tstop,dtout,algor,h0,tol,jcen,rcen,
     %  rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,stat,
     %  id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
c
c Calculate and record the overall change in energy and ang. momentum
  50  open  (23, file=outfile(3), status='old', access='append',
     %  err=50)
      write (23,'(/,a)') mem(57)(1:lmem(57))
      call mxx_en (jcen,nbod,nbig,m,xh,vh,s,en(2),am(2))
c
      write (23,231) mem(58)(1:lmem(58)),
     %  abs((en(2) + en(3) - en(1)) / en(1))
      write (23,232) mem(59)(1:lmem(59)),
     %  abs((am(2) + am(3) - am(1)) / am(1))
      write (23,231) mem(60)(1:lmem(60)), abs(en(3) / en(1))
      write (23,232) mem(61)(1:lmem(61)), abs(am(3) / am(1))
      close (23)
      write (*,'(a)') mem(57)(1:lmem(57))
c
c-----------------------------------------------------------------------
c
 231  format (/,a,1p1e12.5)
 232  format (a,1p1e12.5)
      stop
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MFO_USER.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
```

```
c Applies an arbitrary force, defined by the user.
c
c If using with the symplectic algorithm MAL_MVS, the force should be
c small compared with the force from the central object.
c If using with the conservative Bulirsch-Stoer algorithm MAL_BS2, the
c force should not be a function of the velocities.
c
c N.B. All coordinates and velocities must be with respect to central body
c ===
c-----------------------------------------------------------------------------
c
      subroutine mfo_user (time,jcen,nbod,nbig,m,x,v,a)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig
      real*8 time,jcen(3),m(nbod),x(3,nbod),v(3,nbod),a(3,nbod)
c
c Local
      integer j
c
c-----------------------------------------------------------------------------
c
      do j = 1, nbod
        a(1,j) = 0.d0
        a(2,j) = 0.d0
        a(3,j) = 0.d0
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MAL_HVAR.FOR    (ErikSoft   4 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Does an integration using a variable-timestep integration algorithm. The
c particular integrator routine is ONESTEP and the algorithm must use
c coordinates with respect to the central body.
c
c N.B. This routine is also called by the synchronisation routine mxx_sync,
c ===  in which case OPFLAG = -2. Beware when making changes involving OPFLAG.
c
c-----------------------------------------------------------------------------
c
      subroutine mal_hvar (time,tstart,tstop,dtout,algor,h0,tol,jcen,
     %  rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,
     %  stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,lmem,onestep)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer algor,nbod,nbig,stat(nbod),opt(8),opflag,ngflag,ndump,nfun
      integer lmem(NMESS)
      real*8 time,tstart,tstop,dtout,h0,tol,jcen(3),rcen,rmax
      real*8 en(3),am(3),cefac,m(nbod),xh(3,nbod),vh(3,nbod)
      real*8 s(3,nbod),rho(nbod),rceh(nbod),ngf(4,nbod)
      character*8 id(nbod)
      character*80 outfile(3),dumpfile(4),mem(NMESS)
c
c Local
      integer i,j,k,n,itmp,nhit,ihit(CMAX),jhit(CMAX),chit(CMAX)
      integer dtflag,ejflag,nowflag,stopflag,nstored,ce(NMAX)
```

```
      integer nclo,iclo(CMAX),jclo(CMAX),nce,ice(NMAX),jce(NMAX)
      real*8 tmp0,h,hdid,tout,tdump,tfun,tlog,tsmall,dtdump,dtfun
      real*8 thit(CMAX),dhit(CMAX),thit1,x0(3,NMAX),v0(3,NMAX)
      real*8 rce(NMAX),rphys(NMAX),rcrit(NMAX),a(NMAX)
      real*8 dclo(CMAX),tclo(CMAX),epoch(NMAX)
      real*8 ixvclo(6,CMAX),jxvclo(6,CMAX)
      external mfo_all,onestep
c
c-----------------------------------------------------------------------------
c
c Initialize variables. DTFLAG = 0 implies first ever call to ONESTEP
      dtout  = abs(dtout)
      dtdump = abs(h0) * ndump
      dtfun  = abs(h0) * nfun
      dtflag = 0
      nstored = 0
      tsmall = h0 * 1.d-8
      h = h0
      do j = 2, nbod
        ce(j) = 0.d0
      end do
c
c Calculate close-encounter limits and physical radii for massive bodies
      call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %  m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),1)
c
c Set up time of next output, times of previous dump, log and periodic effect
      if (opflag.eq.-1) then
        tout = tstart
      else
        n = int (abs (time - tstart) / dtout) + 1
        tout = tstart  +  dtout * sign (dble(n), tstop - tstart)
        if ((tstop - tstart)*(tout - tstop).gt.0) tout = tstop
      end if
      tdump = time
      tfun  = time
      tlog  = time
c
c-----------------------------------------------------------------------------
c
c  MAIN  LOOP  STARTS  HERE
c
 100  continue
c
c Is it time for output ?
      if (abs(tout-time).lt.abs(tsmall).and.opflag.ge.-1) then
c
c Beware: the integration may change direction at this point!!!!
        if (opflag.eq.-1) dtflag = 0
c
c Output data for all bodies
        call mio_out (time,jcen,rcen,rmax,nbod,nbig,m,xh,vh,s,rho,
     %    stat,id,opt,opflag,algor,outfile(1))
        call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,stat,id,
     %    0,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,mem,lmem,
     %    outfile,nstored,0)
        tmp0 = tstop - tout
        tout = tout + sign( min( abs(tmp0), abs(dtout) ), tmp0 )
c
c Update the data dump files
        do j = 2, nbod
          epoch(j) = time
        end do
        call mio_dump (time,tstart,tstop,dtout,algor,h,tol,jcen,rcen,
     %    rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,stat,
     %    id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
        tdump = time
      end if
c
c If integration has finished return to the main part of programme
      if (abs(tstop-time).le.abs(tsmall).and.opflag.ne.-1) return
c
```

```
c Set the timestep
      if (opflag.eq.-1) tmp0 = tstart - time
      if (opflag.eq.-2) tmp0 = tstop  - time
      if (opflag.ge.0)  tmp0 = tout   - time
      h = sign ( max( min( abs(tmp0), abs(h) ), tsmall), tmp0 )
c
c Save the current coordinates and velocities
      call mco_iden (time,jcen,nbod,nbig,h,m,xh,vh,x0,v0,ngf,ngflag,opt)
c
c Advance one timestep
      call onestep (time,h,hdid,tol,jcen,nbod,nbig,m,xh,vh,s,rphys,
     %  rcrit,ngf,stat,dtflag,ngflag,opt,nce,ice,jce,mfo_all)
      time = time + hdid
c
c Check if close encounters or collisions occurred
      nclo = 0
      call mce_stat (time,h,rcen,nbod,nbig,m,x0,v0,xh,vh,rce,rphys,
     %  nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,nhit,ihit,jhit,
     %  chit,dhit,thit,thit1,nowflag,stat,outfile(3),mem,lmem)
c
c-----------------------------------------------------------------------
c
c  CLOSE  ENCOUNTERS
c
c If encounter minima occurred, output details and decide whether to stop
      if (nclo.gt.0.and.opflag.ge.-1) then
        itmp = 1
        if (nhit.ne.0) itmp = 0
        call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,stat,id,nclo,
     %    iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,mem,lmem,
     %    outfile,nstored,itmp)
        if (stopflag.eq.1) return
      end if
c
c-----------------------------------------------------------------------
c
c  COLLISIONS
c
c If a collision occurred, output details and resolve the collision
      if (nhit.gt.0.and.opt(2).ne.0) then
        do k = 1, nhit
          if (chit(k).eq.1) then
            i = ihit(k)
            j = jhit(k)
            call mce_coll (thit(k),tstart,en(3),jcen,i,j,nbod,nbig,m,xh,
     %        vh,s,rphys,stat,id,opt,mem,lmem,outfile(3))
          end if
        end do
c
c Remove lost objects, reset flags and recompute Hill and physical radii
        call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,ngf,stat,
     %    id,mem,lmem,outfile(3),itmp)
        dtflag = 1
        if (opflag.ge.0) opflag = 1
        call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %    m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),1)
      end if
c
c-----------------------------------------------------------------------
c
c  COLLISIONS  WITH  CENTRAL  BODY
c
c Check for collisions
      call mce_cent (time,hdid,rcen,jcen,2,nbod,nbig,m,x0,v0,xh,vh,nhit,
     %  jhit,thit,dhit,algor,ngf,ngflag)
c
c Resolve the collisions
      if (nhit.gt.0) then
        do k = 1, nhit
          i = 1
          j = jhit(k)
          call mce_coll (thit(k),tstart,en(3),jcen,i,j,nbod,nbig,m,xh,
```

```
     %      vh,s,rphys,stat,id,opt,mem,lmem,outfile(3))
         end do
c
c Remove lost objects, reset flags and recompute Hill and physical radii
         call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,ngf,stat,
     %     id,mem,lmem,outfile(3),itmp)
         dtflag = 1
         if (opflag.ge.0) opflag = 1
         call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %     m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),0)
       end if
c
c-----------------------------------------------------------------------
c
c  DATA  DUMP  AND  PROGRESS  REPORT
c
c Do the data dump
       if (abs(time-tdump).ge.abs(dtdump).and.opflag.ge.-1) then
         do j = 2, nbod
           epoch(j) = time
         end do
         call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,stat,id,
     %     0,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,mem,lmem,
     %     outfile,nstored,0)
         call mio_dump (time,tstart,tstop,dtout,algor,h,tol,jcen,rcen,
     %     rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,stat,
     %     id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
         tdump = time
       end if
c
c Write a progress report to the log file
       if (abs(time-tlog).ge.abs(dtdump).and.opflag.ge.0) then
         call mxx_en (jcen,nbod,nbig,m,xh,vh,s,en(2),am(2))
         call mio_log (time,tstart,en,am,opt,mem,lmem)
         tlog = time
       end if
c
c-----------------------------------------------------------------------
c
c  CHECK  FOR  EJECTIONS  AND  DO  OTHER  PERIODIC  EFFECTS
c
       if (abs(time-tfun).ge.abs(dtfun).and.opflag.ge.-1) then
c
c Recompute close encounter limits, to allow for changes in Hill radii
         call mce_hill (nbod,m,xh,vh,rce,a)
         do j = 2, nbod
           rce(j) = rce(j) * rceh(j)
         end do
c
c Check for ejections
         call mxx_ejec (time,tstart,rmax,en,am,jcen,2,nbod,nbig,m,xh,vh,
     %     s,stat,id,opt,ejflag,outfile(3),mem,lmem)
c
c Remove lost objects, reset flags and recompute Hill and physical radii
         if (ejflag.ne.0) then
           call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,ngf,stat,
     %       id,mem,lmem,outfile(3),itmp)
           dtflag = 1
           if (opflag.ge.0) opflag = 1
           call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %       m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),0)
         end if
         tfun = time
       end if
c
c Go on to the next time step
       goto 100
c
c-----------------------------------------------------------------------
c
       end
c
```

```
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MAL_HCON.FOR    (ErikSoft   28 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Does an integration using an integrator with a constant stepsize H.
c Input and output to this routine use coordinates XH, and velocities VH,
c with respect to the central body, but the integration algorithm uses
c its own internal coordinates X, and velocities V.
c
c The programme uses the transformation routines COORD and BCOORD to change
c to and from the internal coordinates, respectively.
c
c------------------------------------------------------------------------------
c
      subroutine mal_hcon (time,tstart,tstop,dtout,algor,h0,tol,jcen,
     %  rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,
     %  stat,id,ngf,opt,opflag,ngflag,outfile,dumpfile,mem,lmem,onestep,
     %  coord,bcoord)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer algor,nbod,nbig,stat(nbod),opt(8),opflag,ngflag
      integer lmem(NMESS),ndump,nfun
      real*8 time,tstart,tstop,dtout,h0,tol,jcen(3),rcen,rmax
      real*8 en(3),am(3),cefac,m(nbod),xh(3,nbod),vh(3,nbod)
      real*8 s(3,nbod),rho(nbod),rceh(nbod),ngf(4,nbod)
      character*8 id(nbod)
      character*80 outfile(3),dumpfile(4),mem(NMESS)
c
c Local
      integer i,j,k,n,itmp,nclo,nhit,jhit(CMAX),iclo(CMAX),jclo(CMAX)
      integer dtflag,ejflag,stopflag,colflag,nstored
      real*8 x(3,NMAX),v(3,NMAX),xh0(3,NMAX),vh0(3,NMAX)
      real*8 rce(NMAX),rphys(NMAX),rcrit(NMAX),epoch(NMAX)
      real*8 hby2,tout,tmp0,tdump,tfun,tlog,dtdump,dtfun
      real*8 dclo(CMAX),tclo(CMAX),dhit(CMAX),thit(CMAX)
      real*8 ixvclo(6,CMAX),jxvclo(6,CMAX),a(NMAX)
      external onestep,coord,bcoord
c
c------------------------------------------------------------------------------
c
c Initialize variables. DTFLAG = 0/2: first call ever/normal
      dtout  = abs(dtout)
      dtdump = abs(h0) * ndump
      dtfun  = abs(h0) * nfun
      dtflag = 0
      nstored = 0
      hby2 = 0.500001d0 * abs(h0)
c
c Calculate close-encounter limits and physical radii
      call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %  m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),1)
c
c Set up time of next output, times of previous dump, log and periodic effect
      if (opflag.eq.-1) then
        tout = tstart
      else
        n = int (abs (time-tstart) / dtout) + 1
        tout = tstart  +  dtout * sign (dble(n), tstop - tstart)
        if ((tstop-tstart)*(tout-tstop).gt.0) tout = tstop
      end if
      tdump = time
      tfun  = time
      tlog  = time
c
c Convert to internal coordinates and velocities
```

```
      call coord (time,jcen,nbod,nbig,h0,m,xh,vh,x,v,ngf,ngflag,opt)
c
c-----------------------------------------------------------------------------
c
c  MAIN  LOOP  STARTS  HERE
c
 100  continue
c
c Is it time for output ?
      if (abs(tout-time).le.hby2.and.opflag.ge.-1) then
c
c Beware: the integration may change direction at this point!!!!
        if (opflag.eq.-1.and.dtflag.ne.0) dtflag = 1
c
c Convert to heliocentric coordinates and output data for all bodies
        call bcoord (time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
        call mio_out (time,jcen,rcen,rmax,nbod,nbig,m,xh,vh,s,rho,
     %    stat,id,opt,opflag,algor,outfile(1))
        call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,stat,id,
     %    0,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,mem,lmem,
     %    outfile,nstored,0)
        tmp0 = tstop - tout
        tout = tout + sign( min( abs(tmp0), abs(dtout) ), tmp0 )
c
c Update the data dump files
        do j = 2, nbod
          epoch(j) = time
        end do
        call mio_dump (time,tstart,tstop,dtout,algor,h0,tol,jcen,rcen,
     %    rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,stat,
     %    id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
        tdump = time
      end if
c
c If integration has finished, convert to heliocentric coords and return
      if (abs(tstop-time).le.hby2.and.opflag.ge.0) then
        call bcoord (time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
        return
      end if
c
c Make sure the integration is heading in the right direction
 150  continue
      tmp0 = tstop - time
      if (opflag.eq.-1) tmp0 = tstart - time
      h0 = sign (h0, tmp0)
c
c Save the current heliocentric coordinates and velocities
      if (algor.eq.1) then
        call mco_iden (time,jcen,nbod,nbig,h0,m,x,v,xh0,vh0,ngf,ngflag,
     %    opt)
      else
        call bcoord(time,jcen,nbod,nbig,h0,m,x,v,xh0,vh0,ngf,ngflag,opt)
      end if
      call onestep (time,tstart,h0,tol,rmax,en,am,jcen,rcen,nbod,nbig,
     %  m,x,v,s,rphys,rcrit,rce,stat,id,ngf,algor,opt,dtflag,ngflag,
     %  opflag,colflag,nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,outfile,
     %  mem,lmem)
      time = time + h0
c
c-----------------------------------------------------------------------------
c
c  CLOSE  ENCOUNTERS
c
c If encounter minima occurred, output details and decide whether to stop
      if (nclo.gt.0.and.opflag.ge.-1) then
        itmp = 1
        if (colflag.ne.0) itmp = 0
        call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,stat,id,nclo,
     %    iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,mem,lmem,
     %    outfile,nstored,itmp)
        if (stopflag.eq.1) return
      end if
```

```
c
c-----------------------------------------------------------------------------
c
c  COLLISIONS
c
c If collisions occurred, output details and remove lost objects
      if (colflag.ne.0) then
c
c Reindex the surviving objects
        call bcoord (time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
        call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,ngf,stat,
     %    id,mem,lmem,outfile(3),itmp)
c
c Reset flags, and calculate new Hill radii and physical radii
        dtflag = 1
        if (opflag.ge.0) opflag = 1
        call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %    m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),1)
        call coord (time,jcen,nbod,nbig,h0,m,xh,vh,x,v,ngf,ngflag,opt)
      end if
c
c-----------------------------------------------------------------------------
c
c  COLLISIONS  WITH  CENTRAL  BODY
c
c Check for collisions with the central body
      if (algor.eq.1) then
        call mco_iden(time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
      else
        call bcoord (time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
      end if
      itmp = 2
      if (algor.eq.11.or.algor.eq.12) itmp = 3
      call mce_cent (time,h0,rcen,jcen,itmp,nbod,nbig,m,xh0,vh0,xh,vh,
     %  nhit,jhit,thit,dhit,algor,ngf,ngflag)
c
c If something hit the central body, restore the coords prior to this step
      if (nhit.gt.0) then
        call mco_iden (time,jcen,nbod,nbig,h0,m,xh0,vh0,xh,vh,ngf,
     %    ngflag,opt)
        time = time - h0
c
c Merge the object(s) with the central body
        do k = 1, nhit
          i = 1
          j = jhit(k)
          call mce_coll (thit(k),tstart,en(3),jcen,i,j,nbod,nbig,m,xh,
     %      vh,s,rphys,stat,id,opt,mem,lmem,outfile(3))
        end do
c
c Remove lost objects, reset flags and recompute Hill and physical radii
        call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,ngf,stat,
     %    id,mem,lmem,outfile(3),itmp)
        if (opflag.ge.0) opflag = 1
        dtflag = 1
        call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %    m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),0)
        if (algor.eq.1) then
          call mco_iden (time,jcen,nbod,nbig,h0,m,xh,vh,x,v,ngf,ngflag,
     %      opt)
        else
          call coord (time,jcen,nbod,nbig,h0,m,xh,vh,x,v,ngf,ngflag,opt)
        end if
c
c Redo that integration time step
        goto 150
      end if
c
c-----------------------------------------------------------------------------
c
c  DATA  DUMP  AND  PROGRESS  REPORT
c
```

```
c Convert to heliocentric coords and do the data dump
      if (abs(time-tdump).ge.abs(dtdump).and.opflag.ge.-1) then
        call bcoord (time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
        do j = 2, nbod
          epoch(j) = time
        end do
        call mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,stat,id,
     %    0,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,mem,lmem,
     %    outfile,nstored,0)
        call mio_dump (time,tstart,tstop,dtout,algor,h0,tol,jcen,rcen,
     %    rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,xh,vh,s,rho,rceh,stat,
     %    id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
        tdump = time
      end if
c
c Convert to heliocentric coords and write a progress report to the log file
      if (abs(time-tlog).ge.abs(dtdump).and.opflag.ge.0) then
        call bcoord (time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
        call mxx_en (jcen,nbod,nbig,m,xh,vh,s,en(2),am(2))
        call mio_log (time,tstart,en,am,opt,mem,lmem)
        tlog = time
      end if
c
c-----------------------------------------------------------------------
c
c  CHECK  FOR  EJECTIONS  AND  DO  OTHER  PERIODIC  EFFECTS
c
      if (abs(time-tfun).ge.abs(dtfun).and.opflag.ge.-1) then
        if (algor.eq.1) then
          call mco_iden (time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,
     %      opt)
        else
          call bcoord(time,jcen,nbod,nbig,h0,m,x,v,xh,vh,ngf,ngflag,opt)
        end if
c
c Recompute close encounter limits, to allow for changes in Hill radii
        call mce_hill (nbod,m,xh,vh,rce,a)
        do j = 2, nbod
          rce(j) = rce(j) * rceh(j)
        end do
c
c Check for ejections
        itmp = 2
        if (algor.eq.11.or.algor.eq.12) itmp = 3
        call mxx_ejec (time,tstart,rmax,en,am,jcen,itmp,nbod,nbig,m,xh,
     %    vh,s,stat,id,opt,ejflag,outfile(3),mem,lmem)
c
c Remove ejected objects, reset flags, calculate new Hill and physical radii
        if (ejflag.ne.0) then
          call mxx_elim (nbod,nbig,m,xh,vh,s,rho,rceh,rcrit,ngf,stat,
     %      id,mem,lmem,outfile(3),itmp)
          if (opflag.ge.0) opflag = 1
          dtflag = 1
          call mce_init (tstart,algor,h0,jcen,rcen,rmax,cefac,nbod,nbig,
     %      m,xh,vh,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile(2),0)
          if (algor.eq.1) then
            call mco_iden (time,jcen,nbod,nbig,h0,m,xh,vh,x,v,ngf,
     %        ngflag,opt)
          else
            call coord (time,jcen,nbod,nbig,h0,m,xh,vh,x,v,ngf,ngflag,
     %        opt)
          end if
        end if
        tfun = time
      end if
c
c Go on to the next time step
      goto 100
c
c-----------------------------------------------------------------------
c
      end
```

```
c
c*************************************************************************
c
c    MCE_BOX.FOR    (ErikSoft   30 September 2000)
c
c*************************************************************************
c
c Author: John E. Chambers
c
c Given initial and final coordinates and velocities, the routine returns
c the X and Y coordinates of a box bounding the motion in between the
c end points.
c
c If the X or Y velocity changes sign, the routine performs a quadratic
c interpolation to estimate the corresponding extreme value of X or Y.
c
c-------------------------------------------------------------------------
c
      subroutine mce_box (nbod,h,x0,v0,x1,v1,xmin,xmax,ymin,ymax)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod
      real*8 h,x0(3,nbod), x1(3,nbod), v0(3,nbod),v1(3,nbod)
      real*8   xmin(nbod), xmax(nbod), ymin(nbod),ymax(nbod)
c
c Local
      integer j
      real*8 temp
c
c-------------------------------------------------------------------------
c
      do j = 2, nbod
        xmin(j) = min (x0(1,j), x1(1,j))
        xmax(j) = max (x0(1,j), x1(1,j))
        ymin(j) = min (x0(2,j), x1(2,j))
        ymax(j) = max (x0(2,j), x1(2,j))
c
c If velocity changes sign, do an interpolation
        if ((v0(1,j).lt.0.and.v1(1,j).gt.0).or.
     %      (v0(1,j).gt.0.and.v1(1,j).lt.0)) then
          temp = (v0(1,j)*x1(1,j) - v1(1,j)*x0(1,j)
     %            - .5d0*h*v0(1,j)*v1(1,j)) / (v0(1,j) - v1(1,j))
          xmin(j) = min (xmin(j),temp)
          xmax(j) = max (xmax(j),temp)
        end if
c
        if ((v0(2,j).lt.0.and.v1(2,j).gt.0).or.
     %      (v0(2,j).gt.0.and.v1(2,j).lt.0)) then
          temp = (v0(2,j)*x1(2,j) - v1(2,j)*x0(2,j)
     %            - .5d0*h*v0(2,j)*v1(2,j)) / (v0(2,j) - v1(2,j))
          ymin(j) = min (ymin(j),temp)
          ymax(j) = max (ymax(j),temp)
        end if
      end do
c
c-------------------------------------------------------------------------
c
      return
      end
c
c*************************************************************************
c
c    MCE_CENT.FOR    (ErikSoft   4 March 2001)
c
c*************************************************************************
c
c Author: John E. Chambers
c
c Checks all objects with index I >= I0, to see if they have had a collision
```

```
c with the central body in a time interval H, when given the initial and
c final coordinates and velocities. The routine uses cubic interpolation
c to estimate the minimum separations.
c
c N.B. All coordinates & velocities must be with respect to the central body!!
c ===
c-----------------------------------------------------------------------------
c
      subroutine mce_cent (time,h,rcen,jcen,i0,nbod,nbig,m,x0,v0,x1,v1,
     %  nhit,jhit,thit,dhit,algor,ngf,ngflag)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i0, nbod, nbig, nhit, jhit(CMAX), algor, ngflag
      real*8 time,h,rcen,jcen(3),m(nbod),x0(3,nbod),v0(3,nbod)
      real*8 x1(3,nbod),v1(3,nbod),thit(CMAX),dhit(CMAX),ngf(4,nbod)
c
c Local
      integer j
      real*8 rcen2,mco_acsh,a,q,u0,uhit,m0,mhit,mm,r0,mcen
      real*8 hx,hy,hz,h2,p,rr0,rr1,rv0,rv1,temp,e,v2
      real*8 xu0(3,NMAX),xu1(3,NMAX),vu0(3,NMAX),vu1(3,NMAX)
c
c-----------------------------------------------------------------------------
c
      if (i0.le.0) i0 = 2
      nhit = 0
      rcen2 = rcen * rcen
      mcen = m(1)
c
c If using close-binary code, convert to coords with respect to the binary
c      if (algor.eq.11) then
c        mcen = m(1) + m(2)
c        call mco_h2ub (temp,jcen,nbod,nbig,h,m,x0,v0,xu0,vu0,ngf,ngflag)
c        call mco_h2ub (temp,jcen,nbod,nbig,h,m,x1,v1,xu1,vu1,ngf,ngflag)
c      end if
c
c Check for collisions with the central body
      do j = i0, nbod
        if (algor.eq.11) then
          rr0 = xu0(1,j)*xu0(1,j) + xu0(2,j)*xu0(2,j) +xu0(3,j)*xu0(3,j)
          rr1 = xu1(1,j)*xu1(1,j) + xu1(2,j)*xu1(2,j) +xu1(3,j)*xu1(3,j)
          rv0 = vu0(1,j)*xu0(1,j) + vu0(2,j)*xu0(2,j) +vu0(3,j)*xu0(3,j)
          rv1 = vu1(1,j)*xu1(1,j) + vu1(2,j)*xu1(2,j) +vu1(3,j)*xu1(3,j)
        else
          rr0 = x0(1,j)*x0(1,j) + x0(2,j)*x0(2,j) + x0(3,j)*x0(3,j)
          rr1 = x1(1,j)*x1(1,j) + x1(2,j)*x1(2,j) + x1(3,j)*x1(3,j)
          rv0 = v0(1,j)*x0(1,j) + v0(2,j)*x0(2,j) + v0(3,j)*x0(3,j)
          rv1 = v1(1,j)*x1(1,j) + v1(2,j)*x1(2,j) + v1(3,j)*x1(3,j)
        end if
c
c If inside the central body, or passing through pericentre, use 2-body approx.
        if ((rv0*h.le.0.and.rv1*h.ge.0).or.min(rr0,rr1).le.rcen2) then
          if (algor.eq.11) then
            hx = xu0(2,j) * vu0(3,j)  -  xu0(3,j) * vu0(2,j)
            hy = xu0(3,j) * vu0(1,j)  -  xu0(1,j) * vu0(3,j)
            hz = xu0(1,j) * vu0(2,j)  -  xu0(2,j) * vu0(1,j)
            v2 = vu0(1,j)*vu0(1,j) +vu0(2,j)*vu0(2,j) +vu0(3,j)*vu0(3,j)
          else
            hx = x0(2,j) * v0(3,j)  -  x0(3,j) * v0(2,j)
            hy = x0(3,j) * v0(1,j)  -  x0(1,j) * v0(3,j)
            hz = x0(1,j) * v0(2,j)  -  x0(2,j) * v0(1,j)
            v2 = v0(1,j)*v0(1,j) + v0(2,j)*v0(2,j) + v0(3,j)*v0(3,j)
          end if
          h2 = hx*hx + hy*hy + hz*hz
          p = h2 / (mcen + m(j))
          r0 = sqrt(rr0)
          temp = 1.d0 + p*(v2/(mcen + m(j)) - 2.d0/r0)
          e = sqrt( max(temp,0.d0) )
          q = p / (1.d0 + e)
```

```
c
c If the object hit the central body
          if (q.le.rcen) then
            nhit = nhit + 1
            jhit(nhit) = j
            dhit(nhit) = rcen
c
c Time of impact relative to the end of the timestep
            if (e.lt.1) then
              a = q / (1.d0 - e)
              uhit = sign (acos((1.d0 - rcen/a)/e), -h)
              u0   = sign (acos((1.d0 - r0/a  )/e), rv0)
              mhit = mod (uhit - e*sin(uhit) + PI, TWOPI) - PI
              m0   = mod (u0   - e*sin(u0)   + PI, TWOPI) - PI
            else
              a = q / (e - 1.d0)
              uhit = sign (mco_acsh((1.d0 - rcen/a)/e), -h)
              u0   = sign (mco_acsh((1.d0 - r0/a  )/e), rv0)
              mhit = mod (uhit - e*sinh(uhit) + PI, TWOPI) - PI
              m0   = mod (u0   - e*sinh(u0)   + PI, TWOPI) - PI
            end if
            mm = sqrt((mcen + m(j)) / (a*a*a))
            thit(nhit) = (mhit - m0) / mm + time
          end if
        end if
      end do
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCE_COLL.FOR    (ErikSoft   2 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Resolves a collision between two objects, using the collision model chosen
c by the user. Also writes a message to the information file, and updates the
c value of ELOST, the change in energy due to collisions and ejections.
c
c N.B. All coordinates and velocities must be with respect to central body.
c ===
c
c-----------------------------------------------------------------------
c
      subroutine mce_coll (time,tstart,elost,jcen,i,j,nbod,nbig,m,xh,
     %  vh,s,rphys,stat,id,opt,mem,lmem,outfile)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i,j,nbod,nbig,stat(nbod),opt(8),lmem(NMESS)
      real*8 time,tstart,elost,jcen(3)
      real*8 m(nbod),xh(3,nbod),vh(3,nbod),s(3,nbod),rphys(nbod)
      character*80 outfile,mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer year,month,itmp
      real*8 t1
      character*38 flost,fcol
      character*6 tstring
c
c-----------------------------------------------------------------------
c
c If two bodies collided, check that the less massive one is removed
c (unless the more massive one is a Small body)
```

```
      if (i.ne.0) then
        if (m(j).gt.m(i).and.j.le.nbig) then
          itmp = i
          i = j
          j = itmp
        end if
      end if
c
c Write message to info file (I=0 implies collision with the central body)
  10  open (23, file=outfile, status='old', access='append', err=10)
c
      if (opt(3).eq.1) then
        call mio_jd2y (time,year,month,t1)
        if (i.eq.0) then
          flost = '(1x,a8,a,i10,1x,i2,1x,f8.5)'
          write (23,flost) id(j),mem(67)(1:lmem(67)),year,month,t1
        else
          fcol  = '(1x,a8,a,a8,a,i10,1x,i2,1x,f4.1)'
          write (23,fcol) id(i),mem(69)(1:lmem(69)),id(j),
     %      mem(71)(1:lmem(71)),year,month,t1
        end if
      else
        if (opt(3).eq.3) then
          t1 = (time - tstart) / 365.25d0
          tstring = mem(2)
          flost = '(1x,a8,a,f18.7,a)'
          fcol  = '(1x,a8,a,a8,a,1x,f14.3,a)'
        else
          if (opt(3).eq.0) t1 = time
          if (opt(3).eq.2) t1 = time - tstart
          tstring = mem(1)
          flost = '(1x,a8,a,f18.5,a)'
          fcol  = '(1x,a8,a,a8,a,1x,f14.1,a)'
        end if
        if (i.eq.0.or.i.eq.1) then
          write (23,flost) id(j),mem(67)(1:lmem(67)),t1,tstring
        else
          write (23,fcol) id(i),mem(69)(1:lmem(69)),id(j),
     %      mem(71)(1:lmem(71)),t1,tstring
        end if
      end if
      close (23)
c
c Do the collision (inelastic merger)
      call mce_merg (jcen,i,j,nbod,nbig,m,xh,vh,s,stat,elost)
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCE_HILL.FOR    (ErikSoft   4 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates the Hill radii for all objects given their masses, M,
c coordinates, X, and velocities, V; plus the mass of the central body, M(1)
c Where HILL = a * (m/3*m(1))^(1/3)
c
c If the orbit is hyperbolic or parabolic, the Hill radius is calculated using:
c       HILL = r * (m/3*m(1))^(1/3)
c where R is the current distance from the central body.
c
c The routine also gives the semi-major axis, A, of each object's orbit.
c
c N.B. Designed to use heliocentric coordinates, but should be adequate using
c ===  barycentric coordinates.
c
```

```
c-----------------------------------------------------------------------------
c
      subroutine mce_hill (nbod,m,x,v,hill,a)
c
      implicit none
      include 'mercury.inc'
      real*8 THIRD
      parameter (THIRD = .3333333333333333d0)
c
c Input/Output
      integer nbod
      real*8 m(nbod),x(3,nbod),v(3,nbod),hill(nbod),a(nbod)
c
c Local
      integer j
      real*8 r, v2, gm
c
c-----------------------------------------------------------------------------
c
      do j = 2, nbod
        gm = m(1) + m(j)
        call mco_x2a (gm,x(1,j),x(2,j),x(3,j),v(1,j),v(2,j),v(3,j),a(j),
     %    r,v2)
c If orbit is hyperbolic, use the distance rather than the semi-major axis
        if (a(j).le.0) a(j) = r
        hill(j) = a(j) * (THIRD * m(j) / m(1))**THIRD
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCE_INIT.FOR    (ErikSoft   28 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates close-approach limits RCE (in AU) and physical radii RPHYS
c (in AU) for all objects, given their masses M, coordinates X, velocities
c V, densities RHO, and close-approach limits RCEH (in Hill radii).
c
c Also calculates the changeover distance RCRIT, used by the hybrid
c symplectic integrator. RCRIT is defined to be the larger of N1*HILL and
c N2*H*VMAX, where HILL is the Hill radius, H is the timestep, VMAX is the
c largest expected velocity of any body, and N1, N2 are parameters (see
c section 4.2 of Chambers 1999, Monthly Notices, vol 304, p793-799).
c
c N.B. Designed to use heliocentric coordinates, but should be adequate using
c ===  barycentric coordinates.
c
c-----------------------------------------------------------------------------
c
      subroutine mce_init (tstart,algor,h,jcen,rcen,rmax,cefac,nbod,
     %  nbig,m,x,v,s,rho,rceh,rphys,rce,rcrit,id,opt,outfile,rcritflag)
c
      implicit none
      include 'mercury.inc'
c
      real*8 N2,THIRD
      parameter (N2=.4d0,THIRD=.3333333333333333d0)
c
c Input/Output
      integer nbod,nbig,algor,opt(8),rcritflag
      real*8 tstart,h,jcen(3),rcen,rmax,cefac,m(nbod),x(3,nbod)
      real*8 v(3,nbod),s(3,nbod),rho(nbod),rceh(nbod),rphys(nbod)
      real*8 rce(nbod),rcrit(nbod)
      character*8 id(nbod)
      character*80 outfile
```

```
c
c Local
      integer j
      real*8 a(NMAX),hill(NMAX),temp,amin,vmax,k_2,rhocgs,rcen_2
      character*80 header,c(NMAX)
      character*8 mio_re2c, mio_fl2c
c
c-----------------------------------------------------------------------------
c
      rhocgs = AU * AU * AU * K2 / MSUN
      k_2 = 1.d0 / K2
      rcen_2 = 1.d0 / (rcen * rcen)
      amin = HUGE
c
c Calculate the Hill radii
      call mce_hill (nbod,m,x,v,hill,a)
c
c Determine the maximum close-encounter distances, and the physical radii
      temp = 2.25d0 * m(1) / PI
      do j = 2, nbod
        rce(j)   = hill(j) * rceh(j)
        rphys(j) = hill(j) / a(j) * (temp/rho(j))**THIRD
        amin = min (a(j), amin)
      end do
c
c If required, calculate the changeover distance used by hybrid algorithm
      if (rcritflag.eq.1) then
        vmax = sqrt (m(1) / amin)
        temp = N2 * h * vmax
        do j = 2, nbod
          rcrit(j) = max(hill(j) * cefac, temp)
        end do
      end if
c
c Write list of object's identities to close-encounter output file
      header(1:8)   = mio_fl2c (tstart)
      header(9:16)  = mio_re2c (dble(nbig - 1),   0.d0, 11239423.99d0)
      header(12:19) = mio_re2c (dble(nbod - nbig),0.d0, 11239423.99d0)
      header(15:22) = mio_fl2c (m(1) * k_2)
      header(23:30) = mio_fl2c (jcen(1) * rcen_2)
      header(31:38) = mio_fl2c (jcen(2) * rcen_2 * rcen_2)
      header(39:46) = mio_fl2c (jcen(3) * rcen_2 * rcen_2 * rcen_2)
      header(47:54) = mio_fl2c (rcen)
      header(55:62) = mio_fl2c (rmax)
c
      do j = 2, nbod
        c(j)(1:8) = mio_re2c (dble(j - 1), 0.d0, 11239423.99d0)
        c(j)(4:11) = id(j)
        c(j)(12:19) = mio_fl2c (m(j) * k_2)
        c(j)(20:27) = mio_fl2c (s(1,j) * k_2)
        c(j)(28:35) = mio_fl2c (s(2,j) * k_2)
        c(j)(36:43) = mio_fl2c (s(3,j) * k_2)
        c(j)(44:51) = mio_fl2c (rho(j) / rhocgs)
      end do
c
c Write compressed output to file
  50  open (22, file=outfile, status='old', access='append', err=50)
      write (22,'(a1,a2,i2,a62,i1)') char(12),'6a',algor,header(1:62),
     %  opt(4)
      do j = 2, nbod
        write (22,'(a51)') c(j)(1:51)
      end do
      close (22)
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCE_MERG.FOR    (ErikSoft   2 October 2000)
```

```
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% c
c Author: John E. Chambers
c
c Merges objects I and J inelastically to produce a single new body by
c conserving mass and linear momentum.
c   If J <= NBIG, then J is a Big body
c   If J >  NBIG, then J is a Small body
c   If I = 0, then I is the central body
c
c N.B. All coordinates and velocities must be with respect to central body.
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mce_merg (jcen,i,j,nbod,nbig,m,xh,vh,s,stat,elost)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i, j, nbod, nbig, stat(nbod)
      real*8 jcen(3),m(nbod),xh(3,nbod),vh(3,nbod),s(3,nbod),elost
c
c Local
      integer k
      real*8 tmp1, tmp2, dx, dy, dz, du, dv, dw, msum, mredu, msum_1
      real*8 e0, e1, l2
c
c-----------------------------------------------------------------------------
c
c If a body hits the central body
      if (i.le.1) then
        call mxx_en (jcen,nbod,nbig,m,xh,vh,s,e0,l2)
c
c If a body hit the central body...
        msum   = m(1) + m(j)
        msum_1 = 1.d0 / msum
        mredu  = m(1) * m(j) * msum_1
        dx = xh(1,j)
        dy = xh(2,j)
        dz = xh(3,j)
        du = vh(1,j)
        dv = vh(2,j)
        dw = vh(3,j)
c
c Calculate new spin angular momentum of the central body
        s(1,1) = s(1,1)  +  s(1,j)  +  mredu * (dy * dw  -  dz * dv)
        s(2,1) = s(2,1)  +  s(2,j)  +  mredu * (dz * du  -  dx * dw)
        s(3,1) = s(3,1)  +  s(3,j)  +  mredu * (dx * dv  -  dy * du)
c
c Calculate shift in barycentric coords and velocities of central body
        tmp2 = m(j) * msum_1
        xh(1,1) = tmp2 * xh(1,j)
        xh(2,1) = tmp2 * xh(2,j)
        xh(3,1) = tmp2 * xh(3,j)
        vh(1,1) = tmp2 * vh(1,j)
        vh(2,1) = tmp2 * vh(2,j)
        vh(3,1) = tmp2 * vh(3,j)
        m(1) = msum
        m(j) = 0.d0
        s(1,j) = 0.d0
        s(2,j) = 0.d0
        s(3,j) = 0.d0
c
c Shift the heliocentric coordinates and velocities of all bodies
        do k = 2, nbod
          xh(1,k) = xh(1,k) - xh(1,1)
          xh(2,k) = xh(2,k) - xh(2,1)
          xh(3,k) = xh(3,k) - xh(3,1)
          vh(1,k) = vh(1,k) - vh(1,1)
          vh(2,k) = vh(2,k) - vh(2,1)
```

```
            vh(3,k) = vh(3,k) - vh(3,1)
          end do
c
c Calculate energy loss due to the collision
          call mxx_en (jcen,nbod,nbig,m,xh,vh,s,e1,l2)
          elost = elost + (e0 - e1)
        else
c
c If two bodies collided...
          msum   = m(i) + m(j)
          msum_1 = 1.d0 / msum
          mredu  = m(i) * m(j) * msum_1
          dx = xh(1,i) - xh(1,j)
          dy = xh(2,i) - xh(2,j)
          dz = xh(3,i) - xh(3,j)
          du = vh(1,i) - vh(1,j)
          dv = vh(2,i) - vh(2,j)
          dw = vh(3,i) - vh(3,j)
c
c Calculate energy loss due to the collision
          elost = elost  +  .5d0 * mredu * (du*du + dv*dv + dw*dw)
     %          -  m(i) * m(j) / sqrt(dx*dx + dy*dy + dz*dz)
c
c Calculate spin angular momentum of the new body
          s(1,i) = s(1,i)  +  s(1,j)  +  mredu * (dy * dw  -  dz * dv)
          s(2,i) = s(2,i)  +  s(2,j)  +  mredu * (dz * du  -  dx * dw)
          s(3,i) = s(3,i)  +  s(3,j)  +  mredu * (dx * dv  -  dy * du)
c
c Calculate new coords and velocities by conserving centre of mass & momentum
          tmp1 = m(i) * msum_1
          tmp2 = m(j) * msum_1
          xh(1,i) = xh(1,i) * tmp1  +  xh(1,j) * tmp2
          xh(2,i) = xh(2,i) * tmp1  +  xh(2,j) * tmp2
          xh(3,i) = xh(3,i) * tmp1  +  xh(3,j) * tmp2
          vh(1,i) = vh(1,i) * tmp1  +  vh(1,j) * tmp2
          vh(2,i) = vh(2,i) * tmp1  +  vh(2,j) * tmp2
          vh(3,i) = vh(3,i) * tmp1  +  vh(3,j) * tmp2
          m(i) = msum
        end if
c
c Flag the lost body for removal, and move it away from the new body
        stat(j) = -2
        xh(1,j) = -xh(1,j)
        xh(2,j) = -xh(2,j)
        xh(3,j) = -xh(3,j)
        vh(1,j) = -vh(1,j)
        vh(2,j) = -vh(2,j)
        vh(3,j) = -vh(3,j)
        m(j)   = 0.d0
        s(1,j) = 0.d0
        s(2,j) = 0.d0
        s(3,j) = 0.d0
c
c-----------------------------------------------------------------------------
c
        return
        end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c       MCE_MIN.FOR    (ErikSoft  1 December 1998)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates minimum value of a quantity D, within an interval H, given initial
c and final values D0, D1, and their derivatives D0T, D1T, using third-order
c (i.e. cubic) interpolation.
c
c Also calculates the value of the independent variable T at which D is a
c minimum, with respect to the epoch of D1.
```

```
c
c N.B. The routine assumes that only one minimum is present in the interval H.
c ===
c-----------------------------------------------------------------------------
c
      subroutine mce_min (d0,d1,d0t,d1t,h,d2min,tmin)
c
      implicit none
c
c Input/Output
      real*8 d0,d1,d0t,d1t,h,d2min,tmin
c
c Local
      real*8 a,b,c,temp,tau
c
c-----------------------------------------------------------------------------
c
      if (d0t*h.gt.0.or.d1t*h.lt.0) then
        if (d0.le.d1) then
          d2min = d0
          tmin = -h
        else
          d2min = d1
          tmin = 0.d0
        end if
      else
        temp = 6.d0*(d0 - d1)
        a = temp + 3.d0*h*(d0t + d1t)
        b = temp + 2.d0*h*(d0t + 2.d0*d1t)
        c = h * d1t
c
        temp =-.5d0*(b + sign (sqrt(max(b*b - 4.d0*a*c,0.d0)), b) )
        if (temp.eq.0) then
          tau = 0.d0
        else
          tau = c / temp
        end if
c
c Make sure TAU falls in the interval -1 < TAU < 0
        tau = min(tau, 0.d0)
        tau = max(tau, -1.d0)
c
c Calculate TMIN and D2MIN
        tmin = tau * h
        temp = 1.d0 + tau
        d2min = tau*tau*((3.d0+2.d0*tau)*d0 + temp*h*d0t)
     %        + temp*temp*((1.d0-2.d0*tau)*d1 + tau*h*d1t)
c
c Make sure D2MIN is not negative
        d2min = max(d2min, 0.d0)
      end if
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c    MCE_SNIF.FOR    (ErikSoft   3 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Given initial and final coordinates and velocities X and V, and a timestep
c H, the routine estimates which objects were involved in a close encounter
c during the timestep. The routine examines all objects with indices I >= I0.
c
c Returns an array CE, which for each object is:
c                           0 if it will undergo no encounters
c                           2 if it will pass within RCRIT of a Big body
```

```
c
c Also returns arrays ICE and JCE, containing the indices of each pair of
c objects estimated to have undergone an encounter.
c
c N.B. All coordinates must be with respect to the central body!!!!
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mce_snif (h,i0,nbod,nbig,x0,v0,x1,v1,rcrit,ce,nce,ice,
     %  jce)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i0,nbod,nbig,ce(nbod),nce,ice(NMAX),jce(NMAX)
      real*8 x0(3,nbod),v0(3,nbod),x1(3,nbod),v1(3,nbod),h,rcrit(nbod)
c
c Local
      integer i,j
      real*8 d0,d1,d0t,d1t,d2min,temp,tmin,rc,rc2
      real*8 dx0,dy0,dz0,du0,dv0,dw0,dx1,dy1,dz1,du1,dv1,dw1
      real*8 xmin(NMAX),xmax(NMAX),ymin(NMAX),ymax(NMAX)
c
c-----------------------------------------------------------------------------
c
      if (i0.le.0) i0 = 2
      nce = 0
      do j = 2, nbod
        ce(j) = 0
      end do
c
c Calculate maximum and minimum values of x and y coordinates
      call mce_box (nbod,h,x0,v0,x1,v1,xmin,xmax,ymin,ymax)
c
c Adjust values for the Big bodies by symplectic close-encounter distance
      do j = i0, nbig
        xmin(j) = xmin(j) - rcrit(j)
        xmax(j) = xmax(j) + rcrit(j)
        ymin(j) = ymin(j) - rcrit(j)
        ymax(j) = ymax(j) + rcrit(j)
      end do
c
c Identify pairs whose X-Y boxes overlap, and calculate minimum separation
      do i = i0, nbig
        do j = i + 1, nbod
          if (xmax(i).ge.xmin(j).and.xmax(j).ge.xmin(i)
     %      .and.ymax(i).ge.ymin(j).and.ymax(j).ge.ymin(i)) then
c
c Determine the maximum separation that would qualify as an encounter
            rc = max(rcrit(i), rcrit(j))
            rc2 = rc * rc
c
c Calculate initial and final separations
            dx0 = x0(1,i) - x0(1,j)
            dy0 = x0(2,i) - x0(2,j)
            dz0 = x0(3,i) - x0(3,j)
            dx1 = x1(1,i) - x1(1,j)
            dy1 = x1(2,i) - x1(2,j)
            dz1 = x1(3,i) - x1(3,j)
            d0 = dx0*dx0 + dy0*dy0 + dz0*dz0
            d1 = dx1*dx1 + dy1*dy1 + dz1*dz1
c
c Check for a possible minimum in between
            du0 = v0(1,i) - v0(1,j)
            dv0 = v0(2,i) - v0(2,j)
            dw0 = v0(3,i) - v0(3,j)
            du1 = v1(1,i) - v1(1,j)
            dv1 = v1(2,i) - v1(2,j)
            dw1 = v1(3,i) - v1(3,j)
            d0t = (dx0*du0 + dy0*dv0 + dz0*dw0) * 2.d0
```

```
            d1t = (dx1*du1 + dy1*dv1 + dz1*dw1) * 2.d0
c
c If separation derivative changes sign, find the minimum separation
            d2min = HUGE
            if (d0t*h.le.0.and.d1t*h.ge.0) call mce_min (d0,d1,d0t,d1t,
     %        h,d2min,tmin)
c
c If minimum separation is small enough, flag this as a possible encounter
            temp = min (d0,d1,d2min)
            if (temp.le.rc2) then
              ce(i) = 2
              ce(j) = 2
              nce = nce + 1
              ice(nce) = i
              jce(nce) = j
            end if
          end if
        end do
      end do
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c    MCE_STAT.FOR    (ErikSoft   1 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Returns details of all close-encounter minima involving at least one Big
c body during a timestep. The routine estimates minima using the initial
c and final coordinates X(0),X(1) and velocities V(0),V(1) of the step, and
c the stepsize H.
c
c  ICLO, JCLO contain the indices of the objects
c  DCLO is their minimum separation
c  TCLO is the time of closest approach relative to current time
c
c The routine also checks for collisions/near misses given the physical radii
c RPHYS, and returns the time THIT of the collision/near miss closest to the
c start of the timestep, and the identities IHIT and JHIT of the objects
c involved.
c
c  NHIT = +1 implies a collision
c         -1    "    a near miss
c
c N.B. All coordinates & velocities must be with respect to the central body!!
c ===
c-----------------------------------------------------------------------
c
      subroutine mce_stat (time,h,rcen,nbod,nbig,m,x0,v0,x1,v1,rce,
     %  rphys,nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,nhit,ihit,jhit,
     %  chit,dhit,thit,thit1,nowflag,stat,outfile,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,stat(nbod),nowflag
      integer nclo,iclo(CMAX),jclo(CMAX)
      integer nhit,ihit(CMAX),jhit(CMAX),chit(CMAX),lmem(NMESS)
      real*8 time,h,rcen,m(nbod),x0(3,nbod),v0(3,nbod)
      real*8 x1(3,nbod),v1(3,nbod),rce(nbod),rphys(nbod)
      real*8 dclo(CMAX),tclo(CMAX),thit(CMAX),dhit(CMAX),thit1
      real*8 ixvclo(6,CMAX),jxvclo(6,CMAX)
      character*80 outfile,mem(NMESS)
c
c Local
```

```
      integer i,j
      real*8 d0,d1,d0t,d1t,hm1,tmp0,tmp1
      real*8 dx0,dy0,dz0,du0,dv0,dw0,dx1,dy1,dz1,du1,dv1,dw1
      real*8 xmin(NMAX),xmax(NMAX),ymin(NMAX),ymax(NMAX)
      real*8 d2min,d2ce,d2near,d2hit,temp,tmin
c
c-----------------------------------------------------------------------------
c
      nhit = 0
      thit1 = sign(HUGE, h)
      hm1 = 1.d0 / h
c
c Calculate maximum and minimum values of x and y coords for each object
      call mce_box (nbod,h,x0,v0,x1,v1,xmin,xmax,ymin,ymax)
c
c Adjust values by the maximum close-encounter radius plus a fudge factor
      do j = 2, nbod
        temp = rce(j) * 1.2d0
        xmin(j) = xmin(j)  -  temp
        xmax(j) = xmax(j)  +  temp
        ymin(j) = ymin(j)  -  temp
        ymax(j) = ymax(j)  +  temp
      end do
c
c Check for close encounters between each pair of objects
      do i = 2, nbig
        do j = i + 1, nbod
          if (   xmax(i).ge.xmin(j).and.xmax(j).ge.xmin(i)
     %      .and.ymax(i).ge.ymin(j).and.ymax(j).ge.ymin(i)
     %      .and.stat(i).ge.0.and.stat(j).ge.0) then
c
c If the X-Y boxes for this pair overlap, check circumstances more closely
            dx0 = x0(1,i) - x0(1,j)
            dy0 = x0(2,i) - x0(2,j)
            dz0 = x0(3,i) - x0(3,j)
            du0 = v0(1,i) - v0(1,j)
            dv0 = v0(2,i) - v0(2,j)
            dw0 = v0(3,i) - v0(3,j)
            d0t = (dx0*du0 + dy0*dv0 + dz0*dw0) * 2.d0
c
            dx1 = x1(1,i) - x1(1,j)
            dy1 = x1(2,i) - x1(2,j)
            dz1 = x1(3,i) - x1(3,j)
            du1 = v1(1,i) - v1(1,j)
            dv1 = v1(2,i) - v1(2,j)
            dw1 = v1(3,i) - v1(3,j)
            d1t = (dx1*du1 + dy1*dv1 + dz1*dw1) * 2.d0
c
c Estimate minimum separation during the time interval, using interpolation
            d0 = dx0*dx0 + dy0*dy0 + dz0*dz0
            d1 = dx1*dx1 + dy1*dy1 + dz1*dz1
            call mce_min (d0,d1,d0t,d1t,h,d2min,tmin)
            d2ce  = max (rce(i), rce(j))
            d2hit = rphys(i) + rphys(j)
            d2ce   = d2ce  * d2ce
            d2hit  = d2hit * d2hit
            d2near = d2hit * 4.d0
c
c If the minimum separation qualifies as an encounter or if a collision
c is in progress, store details
            if ((d2min.le.d2ce.and.d0t*h.le.0.and.d1t*h.ge.0)
     %        .or.(d2min.le.d2hit)) then
              nclo = nclo + 1
              if (nclo.gt.CMAX) then
 230            open (23,file=outfile,status='old',access='append',
     %             err=230)
                write (23,'(/,2a,/,a)') mem(121)(1:lmem(121)),
     %            mem(132)(1:lmem(132)),mem(82)(1:lmem(82))
                close (23)
              else
                tclo(nclo) = tmin + time
                dclo(nclo) = sqrt (max(0.d0,d2min))
```

```
               iclo(nclo) = i
               jclo(nclo) = j
c
c Make sure the more massive body is listed first
               if (m(j).gt.m(i).and.j.le.nbig) then
                 iclo(nclo) = j
                 jclo(nclo) = i
               end if
c
c Make linear interpolation to get coordinates at time of closest approach
               tmp0 = 1.d0 + tmin*hm1
               tmp1 = -tmin*hm1
               ixvclo(1,nclo) = tmp0 * x0(1,i)  +  tmp1 * x1(1,i)
               ixvclo(2,nclo) = tmp0 * x0(2,i)  +  tmp1 * x1(2,i)
               ixvclo(3,nclo) = tmp0 * x0(3,i)  +  tmp1 * x1(3,i)
               ixvclo(4,nclo) = tmp0 * v0(1,i)  +  tmp1 * v1(1,i)
               ixvclo(5,nclo) = tmp0 * v0(2,i)  +  tmp1 * v1(2,i)
               ixvclo(6,nclo) = tmp0 * v0(3,i)  +  tmp1 * v1(3,i)
               jxvclo(1,nclo) = tmp0 * x0(1,j)  +  tmp1 * x1(1,j)
               jxvclo(2,nclo) = tmp0 * x0(2,j)  +  tmp1 * x1(2,j)
               jxvclo(3,nclo) = tmp0 * x0(3,j)  +  tmp1 * x1(3,j)
               jxvclo(4,nclo) = tmp0 * v0(1,j)  +  tmp1 * v1(1,j)
               jxvclo(5,nclo) = tmp0 * v0(2,j)  +  tmp1 * v1(2,j)
               jxvclo(6,nclo) = tmp0 * v0(3,j)  +  tmp1 * v1(3,j)
             end if
           end if
c
c Check for a near miss or collision
           if (d2min.le.d2near) then
             nhit = nhit + 1
             ihit(nhit) = i
             jhit(nhit) = j
             thit(nhit) = tmin + time
             dhit(nhit) = sqrt(d2min)
             chit(nhit) = -1
             if (d2min.le.d2hit) chit(nhit) = 1
c
c Make sure the more massive body is listed first
             if (m(jhit(nhit)).gt.m(ihit(nhit)).and.j.le.nbig) then
               ihit(nhit) = j
               jhit(nhit) = i
             end if
c
c Is this the collision closest to the start of the time step?
             if ((tmin-thit1)*h.lt.0) then
               thit1 = tmin
               nowflag = 0
               if (d1.le.d2hit) nowflag = 1
             end if
           end if
         end if
c
c Move on to the next pair of objects
       end do
     end do
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_ACSH.FOR    (ErikSoft  2 March 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates inverse hyperbolic cosine of an angle X (in radians).
c
c-----------------------------------------------------------------------
```

```
c
      function mco_acsh (x)
c
      implicit none
c
c Input/Output
      real*8 x,mco_acsh
c
c-----------------------------------------------------------------------------
c
      if (x.ge.1.d0) then
        mco_acsh = log (x + sqrt(x*x - 1.d0))
      else
        mco_acsh = 0.d0
      end if
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_B2H.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts barycentric coordinates to coordinates with respect to the central
c body.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_b2h (time,jcen,nbod,nbig,h,m,x,v,xh,vh,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),x(3,nbod),v(3,nbod),xh(3,nbod)
      real*8 vh(3,nbod),ngf(4,nbod)
c
c Local
      integer j
c
c-----------------------------------------------------------------------------
c
      do j = 2, nbod
        xh(1,j) = x(1,j) - x(1,1)
        xh(2,j) = x(2,j) - x(2,1)
        xh(3,j) = x(3,j) - x(3,1)
        vh(1,j) = v(1,j) - v(1,1)
        vh(2,j) = v(2,j) - v(2,1)
        vh(3,j) = v(3,j) - v(3,1)
      enddo
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_DH2H.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts democratic heliocentric coordinates to coordinates with respect
```

```
c to the central body.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_dh2h (time,jcen,nbod,nbig,h,m,x,v,xh,vh,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),x(3,nbod),v(3,nbod),xh(3,nbod)
      real*8 vh(3,nbod),ngf(4,nbod)
c
c Local
      integer j
      real*8 mvsum(3),temp
c
c-----------------------------------------------------------------------------
c
      mvsum(1) = 0.d0
      mvsum(2) = 0.d0
      mvsum(3) = 0.d0
c
      do j = 2, nbod
        xh(1,j) = x(1,j)
        xh(2,j) = x(2,j)
        xh(3,j) = x(3,j)
        mvsum(1) = mvsum(1)  +  m(j) * v(1,j)
        mvsum(2) = mvsum(2)  +  m(j) * v(2,j)
        mvsum(3) = mvsum(3)  +  m(j) * v(3,j)
      end do
c
      temp = 1.d0 / m(1)
      mvsum(1) = temp * mvsum(1)
      mvsum(2) = temp * mvsum(2)
      mvsum(3) = temp * mvsum(3)
c
      do j = 2, nbod
        vh(1,j) = v(1,j) + mvsum(1)
        vh(2,j) = v(2,j) + mvsum(2)
        vh(3,j) = v(3,j) + mvsum(3)
      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 (time,jcen,nbod,nbig,h,m,xh,vh,x,v,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),x(3,nbod),v(3,nbod),xh(3,nbod)
      real*8 vh(3,nbod),ngf(4,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_MVS2H.FOR    (ErikSoft   28 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Applies a symplectic corrector, which converts coordinates for a second-
c order mixed-variable symplectic integrator to ones with respect to the
c central body.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_mvs2h (time,jcen,nbod,nbig,h,m,x,v,xh,vh,ngf,
     %  ngflag,opt)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),x(3,nbod),v(3,nbod),xh(3,nbod)
      real*8 vh(3,nbod),ngf(4,nbod)
c
c Local
      integer j,k,iflag,stat(NMAX)
      real*8 minside,msofar,gm(NMAX),a(3,NMAX),xj(3,NMAX),vj(3,NMAX)
      real*8 ha(2),hb(2),rt10,angf(3,NMAX),ausr(3,NMAX)
c
c-----------------------------------------------------------------------------
c
      rt10 = sqrt(10.d0)
      ha(1) =  h * rt10 * 3.d0 / 10.d0
      hb(1) = -h * rt10 / 72.d0
      ha(2) =  h * rt10 / 5.d0
      hb(2) =  h * rt10 / 24.d0
      do j = 2, nbod
        angf(1,j) = 0.d0
        angf(2,j) = 0.d0
        angf(3,j) = 0.d0
        ausr(1,j) = 0.d0
        ausr(2,j) = 0.d0
        ausr(3,j) = 0.d0
      end do
      call mco_iden (time,jcen,nbod,nbig,h,m,x,v,xh,vh,ngf,ngflag,opt)
c
c Calculate effective central masses for Kepler drifts
      minside = m(1)
      do j = 2, nbig
        msofar = minside + m(j)
        gm(j) = m(1) * msofar / minside
        minside = msofar
      end do
```

```
c
c Two step corrector
      do k = 1, 2
c
c Advance Keplerian Hamiltonian (Jacobi/helio coords for Big/Small bodies)
        call mco_h2j(time,jcen,nbig,nbig,h,m,xh,vh,xj,vj,ngf,ngflag,opt)
        do j = 2, nbig
          iflag = 0
          call drift_one (gm(j),xj(1,j),xj(2,j),xj(3,j),vj(1,j),
     %      vj(2,j),vj(3,j),ha(k),iflag)
        end do
        do j = nbig + 1, nbod
          iflag = 0
          call drift_one (m(1),xh(1,j),xh(2,j),xh(3,j),vh(1,j),vh(2,j),
     %      vh(3,j),ha(k),iflag)
        end do
c
c Advance Interaction Hamiltonian
        call mco_j2h(time,jcen,nbig,nbig,h,m,xj,vj,xh,vh,ngf,ngflag,opt)
        call mfo_mvs (jcen,nbod,nbig,m,xh,xj,a,stat)
c
c If required, apply non-gravitational and user-defined forces
        if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,xh,vh,
     %    ausr)
        if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,xh,vh,angf,
     %    ngf)
c
        do j = 2, nbod
          vh(1,j) = vh(1,j)  -  hb(k) * (angf(1,j) + ausr(1,j) + a(1,j))
          vh(2,j) = vh(2,j)  -  hb(k) * (angf(2,j) + ausr(2,j) + a(2,j))
          vh(3,j) = vh(3,j)  -  hb(k) * (angf(3,j) + ausr(3,j) + a(3,j))
        end do
c
c Advance Keplerian Hamiltonian (Jacobi/helio coords for Big/Small bodies)
        call mco_h2j(time,jcen,nbig,nbig,h,m,xh,vh,xj,vj,ngf,ngflag,opt)
        do j = 2, nbig
          iflag = 0
          call drift_one (gm(j),xj(1,j),xj(2,j),xj(3,j),vj(1,j),
     %      vj(2,j),vj(3,j),-2.d0*ha(k),iflag)
        end do
        do j = nbig + 1, nbod
          iflag = 0
         call drift_one (m(1),xh(1,j),xh(2,j),xh(3,j),vh(1,j),vh(2,j),
     %     vh(3,j),-2.d0*ha(k),iflag)
        end do
c
c Advance Interaction Hamiltonian
        call mco_j2h(time,jcen,nbig,nbig,h,m,xj,vj,xh,vh,ngf,ngflag,opt)
        call mfo_mvs (jcen,nbod,nbig,m,xh,xj,a,stat)
c
c If required, apply non-gravitational and user-defined forces
        if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,xh,vh,
     %    ausr)
        if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,xh,vh,angf,
     %    ngf)
c
        do j = 2, nbod
          vh(1,j) = vh(1,j)  +  hb(k) * (angf(1,j) + ausr(1,j) + a(1,j))
          vh(2,j) = vh(2,j)  +  hb(k) * (angf(2,j) + ausr(2,j) + a(2,j))
          vh(3,j) = vh(3,j)  +  hb(k) * (angf(3,j) + ausr(3,j) + a(3,j))
        end do
c
c Advance Keplerian Hamiltonian (Jacobi/helio coords for Big/Small bodies)
        call mco_h2j(time,jcen,nbig,nbig,h,m,xh,vh,xj,vj,ngf,ngflag,opt)
        do j = 2, nbig
          iflag = 0
          call drift_one (gm(j),xj(1,j),xj(2,j),xj(3,j),vj(1,j),
     %      vj(2,j),vj(3,j),ha(k),iflag)
        end do
        do j = nbig + 1, nbod
          iflag = 0
          call drift_one (m(1),xh(1,j),xh(2,j),xh(3,j),vh(1,j),vh(2,j),
```

```
     %      vh(3,j),ha(k),iflag)
         end do
         call mco_j2h(time,jcen,nbig,nbig,h,m,xj,vj,xh,vh,ngf,ngflag,opt)
       end do
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  gm = 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(gm/a) )
c
c-----------------------------------------------------------------------------
c
      subroutine mco_el2x (gm,q,e,i,p,n,l,x,y,z,u,v,w)
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 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(gm/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*gm/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(gm/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_H2B.FOR    (ErikSoft   2 March 2001)
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 (time,jcen,nbod,nbig,h,m,xh,vh,x,v,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),xh(3,nbod),vh(3,nbod),x(3,nbod)
      real*8 v(3,nbod),ngf(4,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_H2MVS.FOR    (ErikSoft   28 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Applies an inverse symplectic corrector, which converts coordinates with
c respect to the central body to integrator coordinates for a second-order
c mixed-variable symplectic integrator.
c
c-----------------------------------------------------------------------
c
      subroutine mco_h2mvs (time,jcen,nbod,nbig,h,m,xh,vh,x,v,ngf,
     %  ngflag,opt)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),xh(3,nbod),vh(3,nbod),x(3,nbod)
      real*8 v(3,nbod),ngf(4,nbod)
c
c Local
      integer j,k,iflag,stat(NMAX)
      real*8 minside,msofar,gm(NMAX),a(3,NMAX),xj(3,NMAX),vj(3,NMAX)
      real*8 ha(2),hb(2),rt10,angf(3,NMAX),ausr(3,NMAX)
c
```

```
c-----------------------------------------------------------------------------
c
      rt10 = sqrt(10.d0)
      ha(1) = -h * rt10 / 5.d0
      hb(1) = -h * rt10 / 24.d0
      ha(2) = -h * rt10 * 3.d0 / 10.d0
      hb(2) =  h * rt10 / 72.d0
      do j = 2, nbod
        angf(1,j) = 0.d0
        angf(2,j) = 0.d0
        angf(3,j) = 0.d0
        ausr(1,j) = 0.d0
        ausr(2,j) = 0.d0
        ausr(3,j) = 0.d0
      end do
      call mco_iden (time,jcen,nbod,nbig,h,m,xh,vh,x,v,ngf,ngflag,opt)
c
c Calculate effective central masses for Kepler drifts
      minside = m(1)
      do j = 2, nbig
        msofar = minside + m(j)
        gm(j) = m(1) * msofar / minside
        minside = msofar
      end do
c
      do k = 1, 2
c
c Advance Keplerian Hamiltonian (Jacobi/helio coords for Big/Small bodies)
        call mco_h2j (time,jcen,nbig,nbig,h,m,x,v,xj,vj,ngf,ngflag,opt)
        do j = 2, nbig
          iflag = 0
          call drift_one (gm(j),xj(1,j),xj(2,j),xj(3,j),vj(1,j),
     %      vj(2,j),vj(3,j),ha(k),iflag)
        end do
        do j = nbig + 1, nbod
          iflag = 0
          call drift_one (m(1),x(1,j),x(2,j),x(3,j),v(1,j),v(2,j),
     %      v(3,j),ha(k),iflag)
        end do
c
c Advance Interaction Hamiltonian
        call mco_j2h (time,jcen,nbig,nbig,h,m,xj,vj,x,v,ngf,ngflag,opt)
        call mfo_mvs (jcen,nbod,nbig,m,x,xj,a,stat)
c
c If required, apply non-gravitational and user-defined forces
        if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,x,v,ausr)
        if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,x,v,angf,ngf)
c
        do j = 2, nbod
          v(1,j) = v(1,j)  +  hb(k) * (angf(1,j) + ausr(1,j) + a(1,j))
          v(2,j) = v(2,j)  +  hb(k) * (angf(2,j) + ausr(2,j) + a(2,j))
          v(3,j) = v(3,j)  +  hb(k) * (angf(3,j) + ausr(3,j) + a(3,j))
        end do
c
c Advance Keplerian Hamiltonian (Jacobi/helio coords for Big/Small bodies)
        call mco_h2j (time,jcen,nbig,nbig,h,m,x,v,xj,vj,ngf,ngflag,opt)
        do j = 2, nbig
          iflag = 0
          call drift_one (gm(j),xj(1,j),xj(2,j),xj(3,j),vj(1,j),
     %      vj(2,j),vj(3,j),-2.d0*ha(k),iflag)
        end do
        do j = nbig + 1, nbod
          iflag = 0
          call drift_one (m(1),x(1,j),x(2,j),x(3,j),v(1,j),v(2,j),
     %      v(3,j),-2.d0*ha(k),iflag)
        end do
c
c Advance Interaction Hamiltonian
        call mco_j2h (time,jcen,nbig,nbig,h,m,xj,vj,x,v,ngf,ngflag,opt)
        call mfo_mvs (jcen,nbod,nbig,m,x,xj,a,stat)
c
c If required, apply non-gravitational and user-defined forces
```

```
        if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,x,v,ausr)
        if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,x,v,angf,ngf)
c
        do j = 2, nbod
          v(1,j) = v(1,j)  -  hb(k) * (angf(1,j) + ausr(1,j) + a(1,j))
          v(2,j) = v(2,j)  -  hb(k) * (angf(2,j) + ausr(2,j) + a(2,j))
          v(3,j) = v(3,j)  -  hb(k) * (angf(3,j) + ausr(3,j) + a(3,j))
        end do
c
c Advance Keplerian Hamiltonian (Jacobi/helio coords for Big/Small bodies)
        call mco_h2j (time,jcen,nbig,nbig,h,m,x,v,xj,vj,ngf,ngflag,opt)
        do j = 2, nbig
          iflag = 0
          call drift_one (gm(j),xj(1,j),xj(2,j),xj(3,j),vj(1,j),
     %       vj(2,j),vj(3,j),ha(k),iflag)
        end do
        do j = nbig + 1, nbod
          iflag = 0
          call drift_one (m(1),x(1,j),x(2,j),x(3,j),v(1,j),v(2,j),
     %       v(3,j),ha(k),iflag)
        end do
        call mco_j2h (time,jcen,nbig,nbig,h,m,xj,vj,x,v,ngf,ngflag,opt)
      end do
c
c--------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_H2DH.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Convert coordinates with respect to the central body to democratic
c heliocentric coordinates.
c
c--------------------------------------------------------------------------
c
      subroutine mco_h2dh (time,jcen,nbod,nbig,h,m,xh,vh,x,v,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),xh(3,nbod),vh(3,nbod),x(3,nbod)
      real*8 v(3,nbod),ngf(4,nbod)
c
c Local
      integer j
      real*8 mtot,temp,mvsum(3)
c
c--------------------------------------------------------------------------
c
      mtot = 0.d0
      mvsum(1) = 0.d0
      mvsum(2) = 0.d0
      mvsum(3) = 0.d0
c
      do j = 2, nbod
        x(1,j) = xh(1,j)
        x(2,j) = xh(2,j)
        x(3,j) = xh(3,j)
        mtot = mtot + 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
```

```
c
      temp = 1.d0 / (m(1) + mtot)
c
      mvsum(1) = temp * mvsum(1)
      mvsum(2) = temp * mvsum(2)
      mvsum(3) = temp * mvsum(3)
c
      do j = 2, nbod
        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 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts coordinates with respect to the central body to Jacobi coordinates.
c
c N.B. The coordinates respect to the central body for the small bodies
c ===  are assumed to be equal to their Jacobi coordinates.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_h2j (time,jcen,nbod,nbig,h,m,xh,vh,x,v,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbig),xh(3,nbig),vh(3,nbig),x(3,nbig)
      real*8 v(3,nbig),ngf(4,nbod)
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_J2H.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts Jacobi coordinates to coordinates with respect to the central
c body.
c
c N.B. The Jacobi coordinates of the small bodies are assumed to be equal
c ===  to their coordinates with respect to the central body.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_j2h (time,jcen,nbod,nbig,h,m,x,v,xh,vh,ngf,ngflag,
     %  opt)
c
      implicit none
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8)
      real*8 time,jcen(3),h,m(nbod),x(3,nbod),v(3,nbod),xh(3,nbod)
      real*8 vh(3,nbod),ngf(4,nbod)
c
c Local
      integer j
      real*8 mtot, mx, my, mz, mu, mv, mw, temp
c
c-----------------------------------------------------------------------------
c
      xh(1,2) = x(1,2)
      xh(2,2) = x(2,2)
      xh(3,2) = x(3,2)
      vh(1,2) = v(1,2)
      vh(2,2) = v(2,2)
      vh(3,2) = v(3,2)
      mtot = m(2)
      temp = m(2) / (mtot + m(1))
      mx = temp * x(1,2)
```

```
      my = temp * x(2,2)
      mz = temp * x(3,2)
      mu = temp * v(1,2)
      mv = temp * v(2,2)
      mw = temp * v(3,2)
c
      do j = 3, nbig - 1
        xh(1,j) = x(1,j) + mx
        xh(2,j) = x(2,j) + my
        xh(3,j) = x(3,j) + mz
        vh(1,j) = v(1,j) + mu
        vh(2,j) = v(2,j) + mv
        vh(3,j) = v(3,j) + mw
        mtot = mtot + m(j)
        temp = m(j) / (mtot + m(1))
        mx = mx  +  temp * x(1,j)
        my = my  +  temp * x(2,j)
        mz = mz  +  temp * x(3,j)
        mu = mu  +  temp * v(1,j)
        mv = mv  +  temp * v(2,j)
        mw = mw  +  temp * v(3,j)
      enddo
c
      if (nbig.gt.2) then
        xh(1,nbig) = x(1,nbig) + mx
        xh(2,nbig) = x(2,nbig) + my
        xh(3,nbig) = x(3,nbig) + mz
        vh(1,nbig) = v(1,nbig) + mu
        vh(2,nbig) = v(2,nbig) + mv
        vh(3,nbig) = v(3,nbig) + mw
      end if
c
      do j = nbig + 1, nbod
        xh(1,j) = x(1,j)
        xh(2,j) = x(2,j)
        xh(3,j) = x(3,j)
        vh(1,j) = v(1,j)
        vh(2,j) = v(2,j)
        vh(3,j) = v(3,j)
      end do
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      MCO_X2A.FOR    (ErikSoft   4 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates an object's orbital semi-major axis given its Cartesian coords.
c
c-----------------------------------------------------------------------------
c
      subroutine mco_x2a (gm,x,y,z,u,v,w,a,r,v2)
c
      implicit none
```

```
c
c Input/Output
      real*8 gm,x,y,z,u,v,w,a,r,v2
c
c-----------------------------------------------------------------------
c
      r  = sqrt(x * x  +  y * y  +  z * z)
      v2 =      u * u  +  v * v  +  w * w
      a  = gm * r / (2.d0 * gm  -  r * v2)
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_X2OV.FOR    (ErikSoft   20 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates output variables for an object given its coordinates and
c velocities. 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_x2ov (rcen,rmax,mcen,m,x,y,z,u,v,w,fr,theta,phi,fv,
     %  vtheta,vphi)
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,v2,v1,be,ke,temp
c
c-----------------------------------------------------------------------
c
        r = sqrt(x*x + y*y + z*z)
        v2 =     u*u + v*v + w*w
        v1 = sqrt(v2)
        be = (mcen + m) / r
        ke = .5d0 * v2
c
        fr = log10 (min(max(r, rcen), rmax) / rcen)
        temp = ke / be
        fv = 1.d0 / (1.d0 + 2.d0*temp*temp)
c
        theta  = mod (acos (z / r) + TWOPI, TWOPI)
        vtheta = mod (acos (w / v1) + TWOPI, TWOPI)
        phi  = mod (atan2 (y, x) + TWOPI, TWOPI)
        vphi = mod (atan2 (v, u) + TWOPI, TWOPI)
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MCO_X2EL.FOR    (ErikSoft  23 January 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) l = l + TWOPI
      if (l.gt.TWOPI) l = mod (l, TWOPI)
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MDT_BS1.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Integrates NBOD bodies (of which NBIG are Big) for one timestep H0
c using the Bulirsch-Stoer method. The accelerations are calculated using the
c subroutine FORCE. The accuracy of the step is approximately determined
c by the tolerance parameter TOL.
c
c N.B. Input/output must be in coordinates with respect to the central body.
c ===
c
c-----------------------------------------------------------------------
c
      subroutine mdt_bs1 (time,h0,hdid,tol,jcen,nbod,nbig,mass,x0,v0,s,
     %  rphys,rcrit,ngf,stat,dtflag,ngflag,opt,nce,ice,jce,force)
c
      implicit none
      include 'mercury.inc'
c
      real*8 SHRINK,GROW
      parameter (SHRINK=.55d0,GROW=1.3d0)
c
c Input/Output
      integer nbod, nbig, opt(8), stat(nbod), dtflag, ngflag
      integer nce, ice(nce), jce(nce)
      real*8 time,h0,hdid,tol,jcen(3),mass(nbod),x0(3,nbod),v0(3,nbod)
      real*8 s(3,nbod),ngf(4,nbod),rphys(nbod),rcrit(nbod)
      external force
```

```
c
c Local
      integer j, j1, k, n
      real*8 tmp0,tmp1,tmp2,errmax,tol2,h,hx2,h2(8)
      real*8 x(3,NMAX),v(3,NMAX),xend(3,NMAX),vend(3,NMAX)
      real*8 a(3,NMAX),a0(3,NMAX),d(6,NMAX,8),xscal(NMAX),vscal(NMAX)
c
c-----------------------------------------------------------------------------
c
      tol2 = tol * tol
c
c Calculate arrays used to scale the relative error (R^2 for position and
c V^2 for velocity).
      do k = 2, nbod
        tmp1 = x0(1,k)*x0(1,k) + x0(2,k)*x0(2,k) + x0(3,k)*x0(3,k)
        tmp2 = v0(1,k)*v0(1,k) + v0(2,k)*v0(2,k) + v0(3,k)*v0(3,k)
        xscal(k) = 1.d0 / tmp1
        vscal(k) = 1.d0 / tmp2
      end do
c
c Calculate accelerations at the start of the step
      call force (time,jcen,nbod,nbig,mass,x0,v0,s,rcrit,a0,stat,ngf,
     %  ngflag,opt,nce,ice,jce)
c
 100  continue
c
c For each value of N, do a modified-midpoint integration with 2N substeps
      do n = 1, 8
        h = h0 / (2.d0 * float(n))
        h2(n) = .25d0 / (n*n)
        hx2 = h * 2.d0
c
        do k = 2, nbod
          x(1,k) = x0(1,k) + h*v0(1,k)
          x(2,k) = x0(2,k) + h*v0(2,k)
          x(3,k) = x0(3,k) + h*v0(3,k)
          v(1,k) = v0(1,k) + h*a0(1,k)
          v(2,k) = v0(2,k) + h*a0(2,k)
          v(3,k) = v0(3,k) + h*a0(3,k)
        end do
        call force (time,jcen,nbod,nbig,mass,x,v,s,rcrit,a,stat,ngf,
     %    ngflag,opt,nce,ice,jce)
        do k = 2, nbod
          xend(1,k) = x0(1,k) + hx2*v(1,k)
          xend(2,k) = x0(2,k) + hx2*v(2,k)
          xend(3,k) = x0(3,k) + hx2*v(3,k)
          vend(1,k) = v0(1,k) + hx2*a(1,k)
          vend(2,k) = v0(2,k) + hx2*a(2,k)
          vend(3,k) = v0(3,k) + hx2*a(3,k)
        end do
c
        do j = 2, n
          call force (time,jcen,nbod,nbig,mass,xend,vend,s,rcrit,a,stat,
     %      ngf,ngflag,opt,nce,ice,jce)
          do k = 2, nbod
            x(1,k) = x(1,k) + hx2*vend(1,k)
            x(2,k) = x(2,k) + hx2*vend(2,k)
            x(3,k) = x(3,k) + hx2*vend(3,k)
            v(1,k) = v(1,k) + hx2*a(1,k)
            v(2,k) = v(2,k) + hx2*a(2,k)
            v(3,k) = v(3,k) + hx2*a(3,k)
          end do
          call force (time,jcen,nbod,nbig,mass,x,v,s,rcrit,a,stat,ngf,
     %      ngflag,opt,nce,ice,jce)
          do k = 2, nbod
            xend(1,k) = xend(1,k) + hx2*v(1,k)
            xend(2,k) = xend(2,k) + hx2*v(2,k)
            xend(3,k) = xend(3,k) + hx2*v(3,k)
            vend(1,k) = vend(1,k) + hx2*a(1,k)
            vend(2,k) = vend(2,k) + hx2*a(2,k)
            vend(3,k) = vend(3,k) + hx2*a(3,k)
          end do
```

```
         end do
c
         call force (time,jcen,nbod,nbig,mass,xend,vend,s,rcrit,a,stat,
     %     ngf,ngflag,opt,nce,ice,jce)
c
         do k = 2, nbod
           d(1,k,n) = .5d0*(xend(1,k) + x(1,k) + h*vend(1,k))
           d(2,k,n) = .5d0*(xend(2,k) + x(2,k) + h*vend(2,k))
           d(3,k,n) = .5d0*(xend(3,k) + x(3,k) + h*vend(3,k))
           d(4,k,n) = .5d0*(vend(1,k) + v(1,k) + h*a(1,k))
           d(5,k,n) = .5d0*(vend(2,k) + v(2,k) + h*a(2,k))
           d(6,k,n) = .5d0*(vend(3,k) + v(3,k) + h*a(3,k))
         end do
c
c Update the D array, used for polynomial extrapolation
         do j = n - 1, 1, -1
           j1 = j + 1
           tmp0 = 1.d0 / (h2(j) - h2(n))
           tmp1 = tmp0 * h2(j1)
           tmp2 = tmp0 * h2(n)
           do k = 2, nbod
             d(1,k,j) = tmp1 * d(1,k,j1)  -  tmp2 * d(1,k,j)
             d(2,k,j) = tmp1 * d(2,k,j1)  -  tmp2 * d(2,k,j)
             d(3,k,j) = tmp1 * d(3,k,j1)  -  tmp2 * d(3,k,j)
             d(4,k,j) = tmp1 * d(4,k,j1)  -  tmp2 * d(4,k,j)
             d(5,k,j) = tmp1 * d(5,k,j1)  -  tmp2 * d(5,k,j)
             d(6,k,j) = tmp1 * d(6,k,j1)  -  tmp2 * d(6,k,j)
           end do
         end do
c
c After several integrations, test the relative error on extrapolated values
         if (n.gt.3) then
           errmax = 0.d0
c
c Maximum relative position and velocity errors (last D term added)
           do k = 2, nbod
             tmp1 = max( d(1,k,1)*d(1,k,1), d(2,k,1)*d(2,k,1),
     %                   d(3,k,1)*d(3,k,1) )
             tmp2 = max( d(4,k,1)*d(4,k,1), d(5,k,1)*d(5,k,1),
     %                   d(6,k,1)*d(6,k,1) )
             errmax = max(errmax, tmp1*xscal(k), tmp2*vscal(k))
           end do
c
c If error is smaller than TOL, update position and velocity arrays, and exit
           if (errmax.le.tol2) then
             do k = 2, nbod
               x0(1,k) = d(1,k,1)
               x0(2,k) = d(2,k,1)
               x0(3,k) = d(3,k,1)
               v0(1,k) = d(4,k,1)
               v0(2,k) = d(5,k,1)
               v0(3,k) = d(6,k,1)
             end do
c
             do j = 2, n
               do k = 2, nbod
                 x0(1,k) = x0(1,k) + d(1,k,j)
                 x0(2,k) = x0(2,k) + d(2,k,j)
                 x0(3,k) = x0(3,k) + d(3,k,j)
                 v0(1,k) = v0(1,k) + d(4,k,j)
                 v0(2,k) = v0(2,k) + d(5,k,j)
                 v0(3,k) = v0(3,k) + d(6,k,j)
               end do
             end do
c
c Save the actual stepsize used
             hdid = h0
c
c Recommend a new stepsize for the next call to this subroutine
             if (n.eq.8) h0 = h0 * SHRINK
             if (n.lt.7) h0 = h0 * GROW
             return
```

```
         end if
        end if
c
      end do
c
c If errors were too large, redo the step with half the previous step size.
      h0 = h0 * .5d0
      goto 100
c
c-----------------------------------------------------------------------------
c
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MDT_BS2.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Integrates NBOD bodies (of which NBIG are Big) for one timestep H0
c using the Bulirsch-Stoer method. The accelerations are calculated using the
c subroutine FORCE. The accuracy of the step is approximately determined
c by the tolerance parameter TOL.
c
c N.B. This version only works for conservative systems (i.e. force is a
c ===  function of position only) !!!! Hence, non-gravitational forces
c      and post-Newtonian corrections cannot be used.
c
c N.B. Input/output must be in coordinates with respect to the central body.
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mdt_bs2 (time,h0,hdid,tol,jcen,nbod,nbig,mass,x0,v0,s,
     %  rphys,rcrit,ngf,stat,dtflag,ngflag,opt,nce,ice,jce,force)
c
      implicit none
      include 'mercury.inc'
c
      real*8 SHRINK,GROW
      parameter (SHRINK=.55d0,GROW=1.3d0)
c
c Input/Output
      integer nbod, nbig, opt(8), stat(nbod), dtflag, ngflag
      real*8 time,h0,hdid,tol,jcen(3),mass(nbod),x0(3,nbod),v0(3,nbod)
      real*8 s(3,nbod),ngf(4,nbod),rphys(nbod),rcrit(nbod)
      integer nce,ice(nce),jce(nce)
      external force
c
c Local
      integer j,j1,k,n
      real*8 tmp0,tmp1,tmp2,errmax,tol2,h,h2(12),hby2,h2by2
      real*8 xend(3,NMAX),b(3,NMAX),c(3,NMAX)
      real*8 a(3,NMAX),a0(3,NMAX),d(6,NMAX,12),xscal(NMAX),vscal(NMAX)
c
c-----------------------------------------------------------------------------
c
      tol2 = tol * tol
c
c Calculate arrays used to scale the relative error (R^2 for position and
c V^2 for velocity).
      do k = 2, nbod
        tmp1 = x0(1,k)*x0(1,k) + x0(2,k)*x0(2,k) + x0(3,k)*x0(3,k)
        tmp2 = v0(1,k)*v0(1,k) + v0(2,k)*v0(2,k) + v0(3,k)*v0(3,k)
        xscal(k) = 1.d0 / tmp1
        vscal(k) = 1.d0 / tmp2
      end do
c
c Calculate accelerations at the start of the step
      call force (time,jcen,nbod,nbig,mass,x0,v0,s,rcrit,a0,stat,ngf,
```

```
     %  ngflag,opt,nce,ice,jce)
c
 100  continue
c
c For each value of N, do a modified-midpoint integration with N substeps
      do n = 1, 12
        h = h0 / (dble(n))
        hby2  = .5d0 * h
        h2(n) = h * h
        h2by2 = .5d0 * h2(n)
c
        do k = 2, nbod
          b(1,k) = .5d0*a0(1,k)
          b(2,k) = .5d0*a0(2,k)
          b(3,k) = .5d0*a0(3,k)
          c(1,k) = 0.d0
          c(2,k) = 0.d0
          c(3,k) = 0.d0
          xend(1,k) = h2by2 * a0(1,k)  +  h * v0(1,k)  +  x0(1,k)
          xend(2,k) = h2by2 * a0(2,k)  +  h * v0(2,k)  +  x0(2,k)
          xend(3,k) = h2by2 * a0(3,k)  +  h * v0(3,k)  +  x0(3,k)
        end do
c
        do j = 2, n
          call force (time,jcen,nbod,nbig,mass,xend,v0,s,rcrit,a,stat,
     %      ngf,ngflag,opt,nce,ice,jce)
          tmp0 = h * dble(j)
          do k = 2, nbod
            b(1,k) = b(1,k) + a(1,k)
            b(2,k) = b(2,k) + a(2,k)
            b(3,k) = b(3,k) + a(3,k)
            c(1,k) = c(1,k) + b(1,k)
            c(2,k) = c(2,k) + b(2,k)
            c(3,k) = c(3,k) + b(3,k)
            xend(1,k) = h2(n)*c(1,k) + h2by2*a0(1,k) + tmp0*v0(1,k)
     %                + x0(1,k)
            xend(2,k) = h2(n)*c(2,k) + h2by2*a0(2,k) + tmp0*v0(2,k)
     %                + x0(2,k)
            xend(3,k) = h2(n)*c(3,k) + h2by2*a0(3,k) + tmp0*v0(3,k)
     %                + x0(3,k)
          end do
        end do
c
        call force (time,jcen,nbod,nbig,mass,xend,v0,s,rcrit,a,stat,ngf,
     %    ngflag,opt,nce,ice,jce)
c
        do k = 2, nbod
          d(1,k,n) = xend(1,k)
          d(2,k,n) = xend(2,k)
          d(3,k,n) = xend(3,k)
          d(4,k,n) = h*b(1,k) + hby2*a(1,k) + v0(1,k)
          d(5,k,n) = h*b(2,k) + hby2*a(2,k) + v0(2,k)
          d(6,k,n) = h*b(3,k) + hby2*a(3,k) + v0(3,k)
        end do
c
c Update the D array, used for polynomial extrapolation
        do j = n - 1, 1, -1
          j1 = j + 1
          tmp0 = 1.d0 / (h2(j) - h2(n))
          tmp1 = tmp0 * h2(j1)
          tmp2 = tmp0 * h2(n)
          do k = 2, nbod
            d(1,k,j) = tmp1 * d(1,k,j1)  -  tmp2 * d(1,k,j)
            d(2,k,j) = tmp1 * d(2,k,j1)  -  tmp2 * d(2,k,j)
            d(3,k,j) = tmp1 * d(3,k,j1)  -  tmp2 * d(3,k,j)
            d(4,k,j) = tmp1 * d(4,k,j1)  -  tmp2 * d(4,k,j)
            d(5,k,j) = tmp1 * d(5,k,j1)  -  tmp2 * d(5,k,j)
            d(6,k,j) = tmp1 * d(6,k,j1)  -  tmp2 * d(6,k,j)
          end do
        end do
c
c After several integrations, test the relative error on extrapolated values
```

```
        if (n.gt.3) then
          errmax = 0.d0
c
c Maximum relative position and velocity errors (last D term added)
          do k = 2, nbod
            tmp1 = max( d(1,k,1)*d(1,k,1), d(2,k,1)*d(2,k,1),
     %                  d(3,k,1)*d(3,k,1) )
            tmp2 = max( d(4,k,1)*d(4,k,1), d(5,k,1)*d(2,k,1),
     %                  d(6,k,1)*d(6,k,1) )
            errmax = max( errmax, tmp1*xscal(k), tmp2*vscal(k) )
          end do
c
c If error is smaller than TOL, update position and velocity arrays and exit
          if (errmax.le.tol2) then
            do k = 2, nbod
              x0(1,k) = d(1,k,1)
              x0(2,k) = d(2,k,1)
              x0(3,k) = d(3,k,1)
              v0(1,k) = d(4,k,1)
              v0(2,k) = d(5,k,1)
              v0(3,k) = d(6,k,1)
            end do
c
            do j = 2, n
              do k = 2, nbod
                x0(1,k) = x0(1,k) + d(1,k,j)
                x0(2,k) = x0(2,k) + d(2,k,j)
                x0(3,k) = x0(3,k) + d(3,k,j)
                v0(1,k) = v0(1,k) + d(4,k,j)
                v0(2,k) = v0(2,k) + d(5,k,j)
                v0(3,k) = v0(3,k) + d(6,k,j)
              end do
            end do
c
c Save the actual stepsize used
            hdid = h0
c
c Recommend a new stepsize for the next call to this subroutine
            if (n.ge.8) h0 = h0 * SHRINK
            if (n.lt.7) h0 = h0 * GROW
            return
          end if
        end if
c
      end do
c
c If errors were too large, redo the step with half the previous step size.
      h0 = h0 * .5d0
      goto 100
c
c-----------------------------------------------------------------------------
c
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MDT_HY.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Integrates NBOD bodies (of which NBIG are Big) for one timestep H
c using a second-order hybrid-symplectic integrator algorithm
c
c DTFLAG = 0 implies first ever call to this subroutine,
c        = 1 implies first call since number/masses of objects changed.
c        = 2 normal call
c
c N.B. Input/output must be in democratic heliocentric coordinates.
c ===
c
```

```
c-----------------------------------------------------------------------------
c
      subroutine mdt_hy (time,tstart,h0,tol,rmax,en,am,jcen,rcen,nbod,
     %  nbig,m,x,v,s,rphys,rcrit,rce,stat,id,ngf,algor,opt,dtflag,
     %  ngflag,opflag,colflag,nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,
     %  outfile,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,stat(nbod),algor,opt(8),dtflag,ngflag,opflag
      integer colflag,lmem(NMESS),nclo,iclo(CMAX),jclo(CMAX)
      real*8 time,tstart,h0,tol,rmax,en(3),am(3),jcen(3),rcen
      real*8 m(nbod),x(3,nbod),v(3,nbod),s(3,nbod),rphys(nbod)
      real*8 rce(nbod),rcrit(nbod),ngf(4,nbod),tclo(CMAX),dclo(CMAX)
      real*8 ixvclo(6,CMAX),jxvclo(6,CMAX)
      character*80 outfile(3),mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer j,nce,ice(NMAX),jce(NMAX),ce(NMAX),iflag
      real*8 a(3,NMAX),hby2,hrec,x0(3,NMAX),v0(3,NMAX),mvsum(3),temp
      real*8 angf(3,NMAX),ausr(3,NMAX)
      external mfo_hkce
c
c-----------------------------------------------------------------------------
c
      save a, hrec, angf, ausr
      hby2 = h0 * .5d0
      nclo = 0
      colflag = 0
c
c If accelerations from previous call are not valid, calculate them now
      if (dtflag.ne.2) then
        if (dtflag.eq.0) hrec = h0
        call mfo_hy (jcen,nbod,nbig,m,x,rcrit,a,stat)
        dtflag = 2
        do j = 2, nbod
          angf(1,j) = 0.d0
          angf(2,j) = 0.d0
          angf(3,j) = 0.d0
          ausr(1,j) = 0.d0
          ausr(2,j) = 0.d0
          ausr(3,j) = 0.d0
        end do
c If required, apply non-gravitational and user-defined forces
        if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,x,v,ausr)
        if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,x,v,angf,ngf)
      end if
c
c Advance interaction Hamiltonian for H/2
      do j = 2, nbod
        v(1,j) = v(1,j)  +  hby2 * (angf(1,j) + ausr(1,j) + a(1,j))
        v(2,j) = v(2,j)  +  hby2 * (angf(2,j) + ausr(2,j) + a(2,j))
        v(3,j) = v(3,j)  +  hby2 * (angf(3,j) + ausr(3,j) + a(3,j))
      end do
c
c Advance solar Hamiltonian for H/2
      mvsum(1) = 0.d0
      mvsum(2) = 0.d0
      mvsum(3) = 0.d0
      do j = 2, nbod
        mvsum(1) = mvsum(1)  +  m(j) * v(1,j)
        mvsum(2) = mvsum(2)  +  m(j) * v(2,j)
        mvsum(3) = mvsum(3)  +  m(j) * v(3,j)
      end do
c
      temp = hby2 / m(1)
      mvsum(1) = temp * mvsum(1)
      mvsum(2) = temp * mvsum(2)
      mvsum(3) = temp * mvsum(3)
```

```
      do j = 2, nbod
        x(1,j) = x(1,j)  +  mvsum(1)
        x(2,j) = x(2,j)  +  mvsum(2)
        x(3,j) = x(3,j)  +  mvsum(3)
      end do
c
c Save the current coordinates and velocities
      call mco_iden (time,jcen,nbod,nbig,h0,m,x,v,x0,v0,ngf,ngflag,opt)
c
c Advance H_K for H
      do j = 2, nbod
        iflag = 0
        call drift_one (m(1),x(1,j),x(2,j),x(3,j),v(1,j),v(2,j),
     %    v(3,j),h0,iflag)
      end do
c
c Check whether any object separations were < R_CRIT whilst advancing H_K
      call mce_snif (h0,2,nbod,nbig,x0,v0,x,v,rcrit,ce,nce,ice,jce)
c
c If objects had close encounters, advance H_K using Bulirsch-Stoer instead
      if (nce.gt.0) then
        do j = 2, nbod
          if (ce(j).ne.0) then
            x(1,j) = x0(1,j)
            x(2,j) = x0(2,j)
            x(3,j) = x0(3,j)
            v(1,j) = v0(1,j)
            v(2,j) = v0(2,j)
            v(3,j) = v0(3,j)
          end if
        end do
        call mdt_hkce (time,tstart,h0,hrec,tol,rmax,en(3),jcen,rcen,
     %    nbod,nbig,m,x,v,s,rphys,rcrit,rce,stat,id,ngf,algor,opt,
     %    ngflag,colflag,ce,nce,ice,jce,nclo,iclo,jclo,dclo,tclo,ixvclo,
     %    jxvclo,outfile,mem,lmem,mfo_hkce)
      end if
c
c Advance solar Hamiltonian for H/2
      mvsum(1) = 0.d0
      mvsum(2) = 0.d0
      mvsum(3) = 0.d0
      do j = 2, nbod
        mvsum(1) = mvsum(1)  +  m(j) * v(1,j)
        mvsum(2) = mvsum(2)  +  m(j) * v(2,j)
        mvsum(3) = mvsum(3)  +  m(j) * v(3,j)
      end do
c
      temp = hby2 / m(1)
      mvsum(1) = temp * mvsum(1)
      mvsum(2) = temp * mvsum(2)
      mvsum(3) = temp * mvsum(3)
      do j = 2, nbod
        x(1,j) = x(1,j)  +  mvsum(1)
        x(2,j) = x(2,j)  +  mvsum(2)
        x(3,j) = x(3,j)  +  mvsum(3)
      end do
c
c Advance interaction Hamiltonian for H/2
      call mfo_hy (jcen,nbod,nbig,m,x,rcrit,a,stat)
      if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,x,v,ausr)
      if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,x,v,angf,ngf)
c
      do j = 2, nbod
        v(1,j) = v(1,j)  +  hby2 * (angf(1,j) + ausr(1,j) + a(1,j))
        v(2,j) = v(2,j)  +  hby2 * (angf(2,j) + ausr(2,j) + a(2,j))
        v(3,j) = v(3,j)  +  hby2 * (angf(3,j) + ausr(3,j) + a(3,j))
      end do
c
c-----------------------------------------------------------------------
c
      return
      end
```

```
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MDT_HKCE.FOR    (ErikSoft   1 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Integrates NBOD bodies (of which NBIG are Big) for one timestep H under
c the Hamiltonian H_K, including close-encounter terms.
c
c------------------------------------------------------------------------------
c
      subroutine mdt_hkce (time,tstart,h0,hrec,tol,rmax,elost,jcen,
     %  rcen,nbod,nbig,m,x,v,s,rphy,rcrit,rce,stat,id,ngf,algor,opt,
     %  ngflag,colflag,ce,nce,ice,jce,nclo,iclo,jclo,dclo,tclo,ixvclo,
     %  jxvclo,outfile,mem,lmem,force)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,nce,ice(nce),jce(nce),stat(nbod),ngflag,ce(nbod)
      integer algor,opt(8),colflag,lmem(NMESS),nclo,iclo(CMAX)
      integer jclo(CMAX)
      real*8 time,tstart,h0,hrec,tol,rmax,elost,jcen(3),rcen
      real*8 m(nbod),x(3,nbod),v(3,nbod),s(3,nbod)
      real*8 rce(nbod),rphy(nbod),rcrit(nbod),ngf(4,nbod)
      real*8 tclo(CMAX),dclo(CMAX),ixvclo(6,CMAX),jxvclo(6,CMAX)
      character*80 outfile(3),mem(NMESS)
      character*8 id(nbod)
      external force
c
c Local
      integer iback(NMAX),index(NMAX),ibs(NMAX),jbs(NMAX),nclo_old
      integer i,j,k,nbs,nbsbig,statbs(NMAX)
      integer nhit,ihit(CMAX),jhit(CMAX),chit(CMAX),nowflag,dtflag
      real*8 tlocal,hlocal,hdid,tmp0
      real*8 mbs(NMAX),xbs(3,NMAX),vbs(3,NMAX),sbs(3,NMAX)
      real*8 rcritbs(NMAX),rcebs(NMAX),rphybs(NMAX)
      real*8 ngfbs(4,NMAX),x0(3,NMAX),v0(3,NMAX)
      real*8 thit(CMAX),dhit(CMAX),thit1,temp
      character*8 idbs(NMAX)
c
c------------------------------------------------------------------------------
c
c N.B. Don't set nclo to zero!!
      nbs = 1
      nbsbig = 0
      mbs(1) = m(1)
      if (algor.eq.11) mbs(1) = m(1) + m(2)
      sbs(1,1) = s(1,1)
      sbs(2,1) = s(2,1)
      sbs(3,1) = s(3,1)
c
c Put data for close-encounter bodies into local arrays for use with BS routine
      do j = 2, nbod
        if (ce(j).ne.0) then
          nbs = nbs + 1
          if (j.le.nbig) nbsbig = nbs
          mbs(nbs)   = m(j)
          xbs(1,nbs) = x(1,j)
          xbs(2,nbs) = x(2,j)
          xbs(3,nbs) = x(3,j)
          vbs(1,nbs) = v(1,j)
          vbs(2,nbs) = v(2,j)
          vbs(3,nbs) = v(3,j)
          sbs(1,nbs) = s(1,j)
          sbs(2,nbs) = s(2,j)
          sbs(3,nbs) = s(3,j)
          rcebs(nbs) = rce(j)
```

```
            rphybs(nbs) = rphy(j)
            statbs(nbs) = stat(j)
            rcritbs(nbs) = rcrit(j)
            idbs(nbs) = id(j)
            index(nbs) = j
            iback(j) = nbs
          end if
        end do
c
        do k = 1, nce
          ibs(k) = iback(ice(k))
          jbs(k) = iback(jce(k))
        end do
c
        tlocal = 0.d0
        hlocal = sign(hrec,h0)
c
c Begin the Bulirsch-Stoer integration
  50    continue
          tmp0 = abs(h0) - abs(tlocal)
          hrec = hlocal
          if (abs(hlocal).gt.tmp0) hlocal = sign (tmp0, h0)
c
c Save old coordinates and integrate
          call mco_iden (time,jcen,nbs,0,h0,mbs,xbs,vbs,x0,v0,ngf,ngflag,
     %      opt)
          call mdt_bs2 (time,hlocal,hdid,tol,jcen,nbs,nbsbig,mbs,xbs,vbs,
     %      sbs,rphybs,rcritbs,ngfbs,statbs,dtflag,ngflag,opt,nce,
     %      ibs,jbs,force)
          tlocal = tlocal + hdid
c
c Check for close-encounter minima
          nclo_old = nclo
          temp = time + tlocal
          call mce_stat (temp,hdid,rcen,nbs,nbsbig,mbs,x0,v0,xbs,vbs,
     %      rcebs,rphybs,nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,nhit,ihit,
     %      jhit,chit,dhit,thit,thit1,nowflag,statbs,outfile(3),mem,lmem)
c
c If collisions occurred, resolve the collision and return a flag
          if (nhit.gt.0.and.opt(2).ne.0) then
            do k = 1, nhit
              if (chit(k).eq.1) then
                i = ihit(k)
                j = jhit(k)
                call mce_coll (thit(k),tstart,elost,jcen,i,j,nbs,nbsbig,
     %            mbs,xbs,vbs,sbs,rphybs,statbs,idbs,opt,mem,lmem,
     %            outfile(3))
                colflag = colflag + 1
              end if
            end do
          end if
c
c If necessary, continue integrating objects undergoing close encounters
        if ((tlocal - h0)*h0.lt.0) goto 50
c
c Return data for the close-encounter objects to global arrays
        do k = 2, nbs
          j = index(k)
          m(j)   = mbs(k)
          x(1,j) = xbs(1,k)
          x(2,j) = xbs(2,k)
          x(3,j) = xbs(3,k)
          v(1,j) = vbs(1,k)
          v(2,j) = vbs(2,k)
          v(3,j) = vbs(3,k)
          s(1,j) = sbs(1,k)
          s(2,j) = sbs(2,k)
          s(3,j) = sbs(3,k)
          stat(j) = statbs(k)
        end do
        do k = 1, nclo
          iclo(k) = index(iclo(k))
```

```
        jclo(k) = index(jclo(k))
      end do
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MDT_MVS.FOR    (ErikSoft   28 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Integrates NBOD bodies (of which NBIG are Big) for one timestep H
c using a second-order mixed-variable symplectic integrator.
c
c DTFLAG = 0 implies first ever call to this subroutine,
c        = 1 implies first call since number/masses of objects changed.
c        = 2 normal call
c
c N.B. Input/output must be in coordinates with respect to the central body.
c ===
c
c-----------------------------------------------------------------------
c
      subroutine mdt_mvs (time,tstart,h0,tol,rmax,en,am,jcen,rcen,nbod,
     %  nbig,m,x,v,s,rphys,rcrit,rce,stat,id,ngf,algor,opt,dtflag,
     %  ngflag,opflag,colflag,nclo,iclo,jclo,dclo,tclo,ixvclo,jxvclo,
     %  outfile,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,stat(nbod),algor,opt(8),dtflag,ngflag,opflag
      integer colflag,lmem(NMESS),nclo,iclo(CMAX),jclo(CMAX)
      real*8 time,tstart,h0,tol,rmax,en(3),am(3),jcen(3),rcen
      real*8 m(nbod),x(3,nbod),v(3,nbod),s(3,nbod),rphys(nbod)
      real*8 rce(nbod),rcrit(nbod),ngf(4,nbod),tclo(CMAX),dclo(CMAX)
      real*8 ixvclo(6,CMAX),jxvclo(6,CMAX)
      character*80 outfile(3),mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer j,iflag,nhit,ihit(CMAX),jhit(CMAX),chit(CMAX),nowflag
      real*8 xj(3,NMAX),vj(3,NMAX),a(3,NMAX),gm(NMAX),hby2,thit1,temp
      real*8 msofar,minside,x0(3,NMAX),v0(3,NMAX),dhit(CMAX),thit(CMAX)
      real*8 angf(3,NMAX),ausr(3,NMAX)
c
c-----------------------------------------------------------------------
c
      save a, xj, gm, angf, ausr
      hby2 = .5d0 * h0
      nclo = 0
c
c If accelerations from previous call are not valid, calculate them now,
c and also the Jacobi coordinates XJ, and effective central masses GM.
      if (dtflag.ne.2) then
        dtflag = 2
        call mco_h2j (time,jcen,nbig,nbig,h0,m,x,v,xj,vj,ngf,ngflag,opt)
        call mfo_mvs (jcen,nbod,nbig,m,x,xj,a,stat)
c
        minside = m(1)
        do j = 2, nbig
          msofar = minside + m(j)
          gm(j) = m(1) * msofar / minside
          minside = msofar
          angf(1,j) = 0.d0
          angf(2,j) = 0.d0
```

```
          angf(3,j) = 0.d0
          ausr(1,j) = 0.d0
          ausr(2,j) = 0.d0
          ausr(3,j) = 0.d0
        end do
c If required, apply non-gravitational and user-defined forces
        if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,x,v,ausr)
        if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,x,v,angf,ngf)
      end if
c
c Advance interaction Hamiltonian for H/2
      do j = 2, nbod
        v(1,j) = v(1,j)  +  hby2 * (angf(1,j) + ausr(1,j) + a(1,j))
        v(2,j) = v(2,j)  +  hby2 * (angf(2,j) + ausr(2,j) + a(2,j))
        v(3,j) = v(3,j)  +  hby2 * (angf(3,j) + ausr(3,j) + a(3,j))
      end do
c
c Save current coordinates and velocities
      call mco_iden (time,jcen,nbod,nbig,h0,m,x,v,x0,v0,ngf,ngflag,opt)
c
c Advance Keplerian Hamiltonian (Jacobi/helio coords for Big/Small bodies)
      call mco_h2j (time,jcen,nbig,nbig,h0,m,x,v,xj,vj,ngf,ngflag,opt)
      do j = 2, nbig
        iflag = 0
        call drift_one (gm(j),xj(1,j),xj(2,j),xj(3,j),vj(1,j),
     %    vj(2,j),vj(3,j),h0,iflag)
      end do
      do j = nbig + 1, nbod
        iflag = 0
        call drift_one (m(1),x(1,j),x(2,j),x(3,j),v(1,j),v(2,j),
     %    v(3,j),h0,iflag)
      end do
      call mco_j2h (time,jcen,nbig,nbig,h0,m,xj,vj,x,v,ngf,ngflag,opt)
c
c Check for close-encounter minima during drift step
      temp = time + h0
      call mce_stat (temp,h0,rcen,nbod,nbig,m,x0,v0,x,v,rce,rphys,nclo,
     %  iclo,jclo,dclo,tclo,ixvclo,jxvclo,nhit,ihit,jhit,chit,dhit,thit,
     %  thit1,nowflag,stat,outfile(3),mem,lmem)
c
c Advance interaction Hamiltonian for H/2
      call mfo_mvs (jcen,nbod,nbig,m,x,xj,a,stat)
      if (opt(8).eq.1) call mfo_user (time,jcen,nbod,nbig,m,x,v,ausr)
      if (ngflag.eq.1.or.ngflag.eq.3) call mfo_ngf (nbod,x,v,angf,ngf)
c
      do j = 2, nbod
        v(1,j) = v(1,j)  +  hby2 * (angf(1,j) + ausr(1,j) + a(1,j))
        v(2,j) = v(2,j)  +  hby2 * (angf(2,j) + ausr(2,j) + a(2,j))
        v(3,j) = v(3,j)  +  hby2 * (angf(3,j) + ausr(3,j) + a(3,j))
      end do
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MDT_RA15.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Integrates NBOD bodies (of which NBIG are Big) for one timestep H0 using
c Everhart's RA15 integrator algorithm. The accelerations are calculated
c using the subroutine FORCE. The accuracy of the step is approximately
c determined by the tolerance parameter TOL.
c
c Based on RADAU by E. Everhart, Physics Department, University of Denver.
c Comments giving equation numbers refer to Everhart (1985) ''An Efficient
c Integrator that Uses Gauss-Radau Spacings'', in The Dynamics of Comets:
```

```
c Their Origin and Evolution, p185-202, eds. A. Carusi & G. B. Valsecchi,
c pub Reidel. (A listing of the original subroutine is also given in this
c paper.)
c
c DTFLAG = 0 implies first ever call to this subroutine,
c        = 1 implies first call since number/masses of objects changed.
c        = 2 normal call
c
c N.B. Input/output must be in coordinates with respect to the central body.
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mdt_ra15 (time,t,tdid,tol,jcen,nbod,nbig,mass,x1,v1,
     %  spin,rphys,rcrit,ngf,stat,dtflag,ngflag,opt,nce,ice,jce,force)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,dtflag,ngflag,opt(8),stat(nbod)
      integer nce,ice(nce),jce(nce)
      real*8 time,t,tdid,tol,jcen(3),mass(nbod)
      real*8 x1(3*nbod),v1(3*nbod),spin(3*nbod)
      real*8 ngf(4,nbod),rphys(nbod),rcrit(nbod)
      external force
c
c Local
      integer nv,niter,j,k,n
      real*8 x(3*NMAX),v(3*NMAX),a(3*NMAX),a1(3*NMAX)
      real*8 g(7,3*NMAX),b(7,3*NMAX),e(7,3*NMAX)
      real*8 h(8),xc(8),vc(7),c(21),d(21),r(28),s(9)
      real*8 q,q2,q3,q4,q5,q6,q7,temp,gk
c
c-----------------------------------------------------------------------------
c
      save h,xc,vc,c,d,r,b,e
c
c Gauss-Radau spacings for substeps within a sequence, for the 15th order
c integrator. The sum of the H values should be 3.733333333333333
c
      data h/          0.d0,.0562625605369221d0,.1802406917368924d0,
     %  .3526247171131696d0,.5471536263305554d0,.7342101772154105d0,
     %  .8853209468390958d0,.9775206135612875d0/
c
c Constant coefficients used in series expansions for X and V
c  XC: 1/2,  1/6,  1/12, 1/20, 1/30, 1/42, 1/56, 1/72
c  VC: 1/2,  1/3,  1/4,  1/5,  1/6,  1/7,  1/8
      data xc/.5d0,.1666666666666667d0,.08333333333333333d0,.05d0,
     %  .03333333333333333d0,.02380952380952381d0,.01785714285714286d0,
     %  .01388888888888889d0/
      data vc/.5d0,.3333333333333333d0,.25d0,.2d0,
     %  .1666666666666667d0,.1428571428571429d0,.125d0/
c
c If this is first call to the subroutine, set values of the constant arrays
c (R = R21, R31, R32, R41, R42, R43 in Everhart's paper.)
      if (dtflag.eq.0) then
        n = 0
        do j = 2, 8
          do k = 1, j - 1
            n = n + 1
            r(n) = 1.d0 / (h(j) - h(k))
          end do
        end do
c
c Constants to convert between B and G arrays (C = C21, C31, C32, C41, C42...)
        c(1) = - h(2)
        d(1) =   h(2)
        n = 1
        do j = 3, 7
          n = n + 1
          c(n) = -h(j) * c(n-j+2)
```

```
          d(n) =  h(2) * d(n-j+2)
          do k = 3, j - 1
            n = n + 1
            c(n) = c(n-j+1)  -  h(j) * c(n-j+2)
            d(n) = d(n-j+1)  +  h(k) * d(n-j+2)
          end do
          n = n + 1
          c(n) = c(n-j+1) - h(j)
          d(n) = d(n-j+1) + h(j)
        end do
c
        dtflag = 1
      end if
c
      nv = 3 * nbod
  100 continue
c
c If this is first call to subroutine since number/masses of objects changed
c do 6 iterations and initialize B, E arrays, otherwise do 2 iterations.
      if (dtflag.eq.1) then
        niter = 6
        do j = 4, nv
          do k = 1, 7
            b (k,j) = 0.d0
            e (k,j) = 0.d0
          end do
        end do
      else
        niter = 2
      end if
c
c Calculate forces at the start of the sequence
      call force (time,jcen,nbod,nbig,mass,x1,v1,spin,rcrit,a1,stat,ngf,
     %  ngflag,opt,nce,ice,jce)
c
c Find G values from B values predicted at the last call (Eqs. 7 of Everhart)
      do k = 4, nv
        g(1,k) = b(7,k)*d(16) + b(6,k)*d(11) + b(5,k)*d(7)
     %           + b(4,k)*d(4)  + b(3,k)*d(2)  + b(2,k)*d(1)  + b(1,k)
        g(2,k) = b(7,k)*d(17) + b(6,k)*d(12) + b(5,k)*d(8)
     %           + b(4,k)*d(5)  + b(3,k)*d(3)  + b(2,k)
        g(3,k) = b(7,k)*d(18) + b(6,k)*d(13) + b(5,k)*d(9)
     %           + b(4,k)*d(6)  + b(3,k)
        g(4,k) = b(7,k)*d(19) + b(6,k)*d(14) + b(5,k)*d(10) + b(4,k)
        g(5,k) = b(7,k)*d(20) + b(6,k)*d(15) + b(5,k)
        g(6,k) = b(7,k)*d(21) + b(6,k)
        g(7,k) = b(7,k)
      end do
c
c-----------------------------------------------------------------------------
c
c  MAIN  LOOP  STARTS  HERE
c
c For each iteration (six for first call to subroutine, two otherwise)...
      do n = 1, niter
c
c For each substep within a sequence...
        do j = 2, 8
c
c Calculate position predictors using Eqn. 9 of Everhart
          s(1) = t * h(j)
          s(2) = s(1) * s(1) * .5d0
          s(3) = s(2) * h(j) * .3333333333333333d0
          s(4) = s(3) * h(j) * .5d0
          s(5) = s(4) * h(j) * .6d0
          s(6) = s(5) * h(j) * .6666666666666667d0
          s(7) = s(6) * h(j) * .7142857142857143d0
          s(8) = s(7) * h(j) * .75d0
          s(9) = s(8) * h(j) * .7777777777777778d0
c
          do k = 4, nv
            x(k) = s(9)*b(7,k) + s(8)*b(6,k) + s(7)*b(5,k)
```

```
     %           + s(6)*b(4,k) + s(5)*b(3,k) + s(4)*b(2,k)
     %           + s(3)*b(1,k) + s(2)*a1(k)  + s(1)*v1(k) + x1(k)
           end do
c
c If necessary, calculate velocity predictors too, from Eqn. 10 of Everhart
           if (ngflag.ne.0) then
             s(1) = t * h(j)
             s(2) = s(1) * h(j) * .5d0
             s(3) = s(2) * h(j) * .6666666666666667d0
             s(4) = s(3) * h(j) * .75d0
             s(5) = s(4) * h(j) * .8d0
             s(6) = s(5) * h(j) * .8333333333333333d0
             s(7) = s(6) * h(j) * .8571428571428571d0
             s(8) = s(7) * h(j) * .875d0
c
             do k = 4, nv
               v(k) = s(8)*b(7,k) + s(7)*b(6,k) + s(6)*b(5,k)
     %             + s(5)*b(4,k) + s(4)*b(3,k) + s(3)*b(2,k)
     %             + s(2)*b(1,k) + s(1)*a1(k)  + v1(k)
             end do
           end if
c
c Calculate forces at the current substep
           call force (time,jcen,nbod,nbig,mass,x,v,spin,rcrit,a,stat,
     %       ngf,ngflag,opt,nce,ice,jce)
c
c Update G values using Eqs. 4 of Everhart, and update B values using Eqs. 5
           if (j.eq.2) then
             do k = 4, nv
               temp = g(1,k)
               g(1,k) = (a(k) - a1(k)) * r(1)
               b(1,k) = b(1,k) + g(1,k) - temp
             end do
             goto 300
           end if
           if (j.eq.3) then
             do k = 4, nv
               temp = g(2,k)
               gk = a(k) - a1(k)
               g(2,k) = (gk*r(2) - g(1,k))*r(3)
               temp = g(2,k) - temp
               b(1,k) = b(1,k)  +  temp * c(1)
               b(2,k) = b(2,k)  +  temp
             end do
             goto 300
           end if
           if (j.eq.4) then
             do k = 4, nv
               temp = g(3,k)
               gk = a(k) - a1(k)
               g(3,k) = ((gk*r(4) - g(1,k))*r(5) - g(2,k))*r(6)
               temp = g(3,k) - temp
               b(1,k) = b(1,k)  +  temp * c(2)
               b(2,k) = b(2,k)  +  temp * c(3)
               b(3,k) = b(3,k)  +  temp
             end do
             goto 300
           end if
           if (j.eq.5) then
             do k = 4, nv
               temp = g(4,k)
               gk = a(k) - a1(k)
               g(4,k) = (((gk*r(7) - g(1,k))*r(8) - g(2,k))*r(9)
     %                 - g(3,k))*r(10)
               temp = g(4,k) - temp
               b(1,k) = b(1,k)  +  temp * c(4)
               b(2,k) = b(2,k)  +  temp * c(5)
               b(3,k) = b(3,k)  +  temp * c(6)
               b(4,k) = b(4,k)  +  temp
             end do
             goto 300
           end if
```

```
          if (j.eq.6) then
            do k = 4, nv
              temp = g(5,k)
              gk = a(k) - a1(k)
              g(5,k) =  ((((gk*r(11) - g(1,k))*r(12) - g(2,k))*r(13)
     %               - g(3,k))*r(14) - g(4,k))*r(15)
              temp = g(5,k) - temp
              b(1,k) = b(1,k)  +  temp * c(7)
              b(2,k) = b(2,k)  +  temp * c(8)
              b(3,k) = b(3,k)  +  temp * c(9)
              b(4,k) = b(4,k)  +  temp * c(10)
              b(5,k) = b(5,k)  +  temp
            end do
            goto 300
          end if
          if (j.eq.7) then
            do k = 4, nv
              temp = g(6,k)
              gk = a(k) - a1(k)
              g(6,k) = (((((gk*r(16) - g(1,k))*r(17) - g(2,k))*r(18)
     %               - g(3,k))*r(19) - g(4,k))*r(20) - g(5,k))*r(21)
              temp = g(6,k) - temp
              b(1,k) = b(1,k)  +  temp * c(11)
              b(2,k) = b(2,k)  +  temp * c(12)
              b(3,k) = b(3,k)  +  temp * c(13)
              b(4,k) = b(4,k)  +  temp * c(14)
              b(5,k) = b(5,k)  +  temp * c(15)
              b(6,k) = b(6,k)  +  temp
            end do
            goto 300
          end if
          if (j.eq.8) then
            do k = 4, nv
              temp = g(7,k)
              gk = a(k) - a1(k)
              g(7,k) = ((((((gk*r(22) - g(1,k))*r(23) - g(2,k))*r(24)
     %                - g(3,k))*r(25) - g(4,k))*r(26) - g(5,k))*r(27)
     %                - g(6,k))*r(28)
              temp = g(7,k) - temp
              b(1,k) = b(1,k)  +  temp * c(16)
              b(2,k) = b(2,k)  +  temp * c(17)
              b(3,k) = b(3,k)  +  temp * c(18)
              b(4,k) = b(4,k)  +  temp * c(19)
              b(5,k) = b(5,k)  +  temp * c(20)
              b(6,k) = b(6,k)  +  temp * c(21)
              b(7,k) = b(7,k)  +  temp
            end do
          end if
 300      continue
        end do
      end do
c
c-------------------------------------------------------------------------------
c
c  END  OF  MAIN  LOOP
c
c Estimate suitable sequence size for the next call to subroutine (Eqs. 15, 16)
      temp = 0.d0
      do k = 4, nv
        temp = max( temp, abs( b(7,k) ) )
      end do
      temp = temp / (72.d0 * abs(t)**7)
      tdid = t
      if (temp.eq.0) then
        t = tdid * 1.4d0
      else
        t = sign( (tol/temp)**(1.d0/9.d0), tdid )
      end if
c
c If sequence size for the first subroutine call is too big, go back and redo
c the sequence using a smaller size.
      if (dtflag.eq.1.and.abs(t/tdid).lt.1) then
```

```
        t = t * .8d0
        goto 100
      end if
c
c If new sequence size is much bigger than the current one, reduce it
      if (abs(t/tdid).gt.1.4d0) t = tdid * 1.4d0
c
c Find new position and velocity values at end of the sequence (Eqs. 11, 12)
      temp = tdid * tdid
      do k = 4 , nv
        x1(k) = (xc(8)*b(7,k) + xc(7)*b(6,k) + xc(6)*b(5,k)
     %        +  xc(5)*b(4,k) + xc(4)*b(3,k) + xc(3)*b(2,k)
     %        +  xc(2)*b(1,k) + xc(1)*a1(k))*temp + v1(k)*tdid + x1(k)
c
        v1(k) = (vc(7)*b(7,k) + vc(6)*b(6,k) + vc(5)*b(5,k)
     %        +  vc(4)*b(4,k) + vc(3)*b(3,k) + vc(2)*b(2,k)
     %        +  vc(1)*b(1,k) + a1(k))*tdid + v1(k)
      end do
c
c Predict new B values to use at the start of the next sequence. The predicted
c values from the last call are saved as E. The correction, BD, between the
c actual and predicted values of B is applied in advance as a correction.
      q = t / tdid
      q2 = q  * q
      q3 = q  * q2
      q4 = q2 * q2
      q5 = q2 * q3
      q6 = q3 * q3
      q7 = q3 * q4
c
      do k = 4, nv
        s(1) = b(1,k) - e(1,k)
        s(2) = b(2,k) - e(2,k)
        s(3) = b(3,k) - e(3,k)
        s(4) = b(4,k) - e(4,k)
        s(5) = b(5,k) - e(5,k)
        s(6) = b(6,k) - e(6,k)
        s(7) = b(7,k) - e(7,k)
c
c Estimate B values for the next sequence (Eqs. 13 of Everhart).
        e(1,k) = q* (b(7,k)* 7.d0 + b(6,k)* 6.d0 + b(5,k)* 5.d0
     %           +     b(4,k)* 4.d0 + b(3,k)* 3.d0 + b(2,k)*2.d0 + b(1,k))
        e(2,k) = q2*(b(7,k)*21.d0 + b(6,k)*15.d0 + b(5,k)*10.d0
     %           +     b(4,k)* 6.d0 + b(3,k)* 3.d0 + b(2,k))
        e(3,k) = q3*(b(7,k)*35.d0 + b(6,k)*20.d0 + b(5,k)*10.d0
     %           +     b(4,k)*4.d0  + b(3,k))
        e(4,k) = q4*(b(7,k)*35.d0 + b(6,k)*15.d0 + b(5,k)*5.d0 + b(4,k))
        e(5,k) = q5*(b(7,k)*21.d0 + b(6,k)*6.d0  + b(5,k))
        e(6,k) = q6*(b(7,k)*7.d0  + b(6,k))
        e(7,k) = q7* b(7,k)
c
        b(1,k) = e(1,k) + s(1)
        b(2,k) = e(2,k) + s(2)
        b(3,k) = e(3,k) + s(3)
        b(4,k) = e(4,k) + s(4)
        b(5,k) = e(5,k) + s(5)
        b(6,k) = e(6,k) + s(6)
        b(7,k) = e(7,k) + s(7)
      end do
      dtflag = 2
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MFO_ALL.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
```

```
c Author: John E. Chambers
c
c Calculates accelerations on a set of NBOD bodies (of which NBIG are Big)
c due to Newtonian gravitational perturbations, post-Newtonian
c corrections (if required), cometary non-gravitational forces (if required)
c and user-defined forces (if required).
c
c N.B. Input/output must be in coordinates with respect to the central body.
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mfo_all (time,jcen,nbod,nbig,m,x,v,s,rcrit,a,stat,ngf,
     %  ngflag,opt,nce,ice,jce)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,ngflag,stat(nbod),opt(8),nce,ice(nce),jce(nce)
      real*8 time,jcen(3),m(nbod),x(3,nbod),v(3,nbod),s(3,nbod)
      real*8 a(3,nbod),ngf(4,nbod),rcrit(nbod)
c
c Local
      integer j
      real*8 acor(3,NMAX),acen(3)
c
c-----------------------------------------------------------------------------
c
c Newtonian gravitational forces
      call mfo_grav (nbod,nbig,m,x,v,a,stat)
c
c Correct for oblateness of the central body
      if (jcen(1).ne.0.or.jcen(2).ne.0.or.jcen(3).ne.0) then
        call mfo_obl (jcen,nbod,m,x,acor,acen)
        do j = 2, nbod
          a(1,j) = a(1,j) + (acor(1,j) - acen(1))
          a(2,j) = a(2,j) + (acor(2,j) - acen(2))
          a(3,j) = a(3,j) + (acor(3,j) - acen(3))
        end do
      end if
c
c Include non-gravitational (cometary jet) accelerations if necessary
      if (ngflag.eq.1.or.ngflag.eq.3) then
        call mfo_ngf (nbod,x,v,acor,ngf)
        do j = 2, nbod
          a(1,j) = a(1,j) + acor(1,j)
          a(2,j) = a(2,j) + acor(2,j)
          a(3,j) = a(3,j) + acor(3,j)
        end do
      end if
c
c Include radiation pressure/Poynting-Robertson drag if necessary
      if (ngflag.eq.2.or.ngflag.eq.3) then
        call mfo_pr (nbod,nbig,m,x,v,acor,ngf)
        do j = 2, nbod
          a(1,j) = a(1,j) + acor(1,j)
          a(2,j) = a(2,j) + acor(2,j)
          a(3,j) = a(3,j) + acor(3,j)
        end do
      end if
c
c Include post-Newtonian corrections if required
      if (opt(7).eq.1) then
        call mfo_pn (nbod,nbig,m,x,v,acor)
        do j = 2, nbod
          a(1,j) = a(1,j) + acor(1,j)
          a(2,j) = a(2,j) + acor(2,j)
          a(3,j) = a(3,j) + acor(3,j)
        end do
      end if
c
```

```
c Include user-defined accelerations if required
      if (opt(8).eq.1) then
        call mfo_user (time,jcen,nbod,nbig,m,x,v,acor)
        do j = 2, nbod
          a(1,j) = a(1,j) + acor(1,j)
          a(2,j) = a(2,j) + acor(2,j)
          a(3,j) = a(3,j) + acor(3,j)
        end do
      end if
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MFO_GRAV.FOR    (ErikSoft   3 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates accelerations on a set of NBOD bodies (NBIG of which are Big)
c due to gravitational perturbations by all the other bodies, except that
c Small bodies do not interact with one another.
c
c The positions and velocities are stored in arrays X, V with the format
c (x,y,z) and (vx,vy,vz) for each object in succession. The accelerations
c are stored in the array A (ax,ay,az).
c
c N.B. All coordinates and velocities must be with respect to central body!!!!
c ===
c-----------------------------------------------------------------------
c
      subroutine mfo_grav (nbod,nbig,m,x,v,a,stat)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig, stat(nbod)
      real*8 m(nbod), x(3,nbod), v(3,nbod), a(3,nbod)
c
c Local
      integer i, j
      real*8 sx, sy, sz, dx, dy, dz, tmp1, tmp2, s_1, s2, s_3, r3(NMAX)
c
c-----------------------------------------------------------------------
c
      sx = 0.d0
      sy = 0.d0
      sz = 0.d0
      do i = 2, nbod
        a(1,i) = 0.d0
        a(2,i) = 0.d0
        a(3,i) = 0.d0
        s2 = x(1,i)*x(1,i) + x(2,i)*x(2,i) + x(3,i)*x(3,i)
        s_1  = 1.d0 / sqrt(s2)
        r3(i) = s_1 * s_1 * s_1
      end do
c
      do i = 2, nbod
        tmp1 = m(i) * r3(i)
        sx = sx  -  tmp1 * x(1,i)
        sy = sy  -  tmp1 * x(2,i)
        sz = sz  -  tmp1 * x(3,i)
      end do
c
c Direct terms
      do i = 2, nbig
        do j = i + 1, nbod
```

```
          dx = x(1,j) - x(1,i)
          dy = x(2,j) - x(2,i)
          dz = x(3,j) - x(3,i)
          s2 = dx*dx + dy*dy + dz*dz
          s_1 = 1.d0 / sqrt(s2)
          s_3 = s_1 * s_1 * s_1
          tmp1 = s_3 * m(i)
          tmp2 = s_3 * m(j)
          a(1,j) = a(1,j)  -  tmp1 * dx
          a(2,j) = a(2,j)  -  tmp1 * dy
          a(3,j) = a(3,j)  -  tmp1 * dz
          a(1,i) = a(1,i)  +  tmp2 * dx
          a(2,i) = a(2,i)  +  tmp2 * dy
          a(3,i) = a(3,i)  +  tmp2 * dz
        end do
      end do
c
c Indirect terms (add these on last to reduce roundoff error)
      do i = 2, nbod
        tmp1 = m(1) * r3(i)
        a(1,i) = a(1,i)  +  sx  -  tmp1 * x(1,i)
        a(2,i) = a(2,i)  +  sy  -  tmp1 * x(2,i)
        a(3,i) = a(3,i)  +  sz  -  tmp1 * x(3,i)
      end do
c
c-----------------------------------------------------------------------
c
      return
      end
c

c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c     MFO_DRCT.FOR    (ErikSoft   27 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates direct accelerations between bodies in the interaction part
c of the Hamiltonian of a symplectic integrator that partitions close
c encounter terms (e.g. hybrid symplectic algorithms or SyMBA).
c The routine calculates accelerations between all pairs of bodies with
c indices I >= I0.
c
c-----------------------------------------------------------------------
c
      subroutine mfo_drct (i0,nbod,nbig,m,x,rcrit,a,stat)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i0, nbod, nbig, stat(nbod)
      real*8 m(nbod), x(3,nbod), a(3,nbod), rcrit(nbod)
c
c Local
      integer i,j
      real*8 dx,dy,dz,s,s_1,s2,s_3,rc,rc2,q,q2,q3,q4,q5,tmp2,faci,facj
c
c-----------------------------------------------------------------------
c
      if (i0.le.0) i0 = 2
c
      do i = i0, nbig
        do j = i + 1, nbod
          dx = x(1,j) - x(1,i)
          dy = x(2,j) - x(2,i)
          dz = x(3,j) - x(3,i)
          s2 = dx * dx  +  dy * dy  +  dz * dz
          rc = max(rcrit(i), rcrit(j))
          rc2 = rc * rc
```

```
c
          if (s2.ge.rc2) then
            s_1 = 1.d0 / sqrt(s2)
            tmp2 = s_1 * s_1 * s_1
          else if (s2.le.0.01*rc2) then
            tmp2 = 0.d0
          else
            s_1 = 1.d0 / sqrt(s2)
            s   = 1.d0 / s_1
            s_3 = s_1 * s_1 * s_1
            q = (s - 0.1d0*rc) / (0.9d0 * rc)
            q2 = q  * q
            q3 = q  * q2
            q4 = q2 * q2
            q5 = q2 * q3
            tmp2 = (10.d0*q3 - 15.d0*q4 + 6.d0*q5) * s_3
          end if
c
          faci = tmp2 * m(i)
          facj = tmp2 * m(j)
          a(1,j) = a(1,j)  -  faci * dx
          a(2,j) = a(2,j)  -  faci * dy
          a(3,j) = a(3,j)  -  faci * dz
          a(1,i) = a(1,i)  +  facj * dx
          a(2,i) = a(2,i)  +  facj * dy
          a(3,i) = a(3,i)  +  facj * dz
        end do
      end do
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c     MFO_HY.FOR    (ErikSoft   2 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates accelerations due to the Interaction part of the Hamiltonian
c of a hybrid symplectic integrator for a set of NBOD bodies (NBIG of which
c are Big), where Small bodies do not interact with one another.
c
c-----------------------------------------------------------------------
c
      subroutine mfo_hy (jcen,nbod,nbig,m,x,rcrit,a,stat)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig, stat(nbod)
      real*8 jcen(3), m(nbod), x(3,nbod), a(3,nbod), rcrit(nbod)
c
c Local
      integer k
      real*8 aobl(3,NMAX),acen(3)
c
c-----------------------------------------------------------------------
c
c Initialize accelerations to zero
      do k = 1, nbod
        a(1,k) = 0.d0
        a(2,k) = 0.d0
        a(3,k) = 0.d0
      end do
c
c Calculate direct terms
      call mfo_drct (2,nbod,nbig,m,x,rcrit,a,stat)
```

```
c
c Add accelerations due to oblateness of the central body
      if (jcen(1).ne.0.or.jcen(2).ne.0.or.jcen(3).ne.0) then
        call mfo_obl (jcen,nbod,m,x,aobl,acen)
        do k = 2, nbod
          a(1,k) = a(1,k) + aobl(1,k) - acen(1)
          a(2,k) = a(2,k) + aobl(2,k) - acen(2)
          a(3,k) = a(3,k) + aobl(3,k) - acen(3)
        end do
      end if
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c     MFO_HKCE.FOR    (ErikSoft   27 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates accelerations due to the Keplerian part of the Hamiltonian
c of a hybrid symplectic integrator, when close encounters are taking place,
c for a set of NBOD bodies (NBIG of which are Big). Note that Small bodies
c do not interact with one another.
c
c-----------------------------------------------------------------------------
c
      subroutine mfo_hkce (time,jcen,nbod,nbig,m,x,v,spin,rcrit,a,stat,
     %  ngf,ngflag,opt,nce,ice,jce)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,stat(nbod),ngflag,opt(8),nce,ice(nce),jce(nce)
      real*8 time,jcen(3),rcrit(nbod),ngf(4,nbod),m(nbod)
      real*8 x(3,nbod),v(3,nbod),a(3,nbod),spin(3,nbod)
c
c Local
      integer i, j, k
      real*8 tmp2,dx,dy,dz,s,s_1,s2,s_3,faci,facj,rc,rc2,q,q2,q3,q4,q5
c
c-----------------------------------------------------------------------------
c
c Initialize accelerations
      do j = 1, nbod
        a(1,j) = 0.d0
        a(2,j) = 0.d0
        a(3,j) = 0.d0
      end do
c
c Direct terms
      do k = 1, nce
        i = ice(k)
        j = jce(k)
        dx = x(1,j) - x(1,i)
        dy = x(2,j) - x(2,i)
        dz = x(3,j) - x(3,i)
        s2 = dx * dx  +  dy * dy  +  dz * dz
        rc = max (rcrit(i), rcrit(j))
        rc2 = rc * rc
c
        if (s2.lt.rc2) then
          s_1 = 1.d0 / sqrt(s2)
          s_3 = s_1 * s_1 * s_1
          if (s2.le.0.01*rc2) then
            tmp2 = s_3
          else
```

```
            s = 1.d0 / s_1
            q = (s - 0.1d0*rc) / (0.9d0 * rc)
            q2 = q * q
            q3 = q * q2
            q4 = q2 * q2
            q5 = q2 * q3
            tmp2 = (1.d0 - 10.d0*q3 + 15.d0*q4 - 6.d0*q5) * s_3
          end if
c
          faci = tmp2 * m(i)
          facj = tmp2 * m(j)
          a(1,j) = a(1,j)  -  faci * dx
          a(2,j) = a(2,j)  -  faci * dy
          a(3,j) = a(3,j)  -  faci * dz
          a(1,i) = a(1,i)  +  facj * dx
          a(2,i) = a(2,i)  +  facj * dy
          a(3,i) = a(3,i)  +  facj * dz
        end if
      end do
c
c Solar terms
      do i = 2, nbod
        s2 = x(1,i)*x(1,i) + x(2,i)*x(2,i) + x(3,i)*x(3,i)
        s_1 = 1.d0 / sqrt(s2)
        tmp2 = m(1) * s_1 * s_1 * s_1
        a(1,i) = a(1,i)  -  tmp2 * x(1,i)
        a(2,i) = a(2,i)  -  tmp2 * x(2,i)
        a(3,i) = a(3,i)  -  tmp2 * x(3,i)
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c     MFO_MVS.FOR    (ErikSoft   2 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates accelerations on a set of NBOD bodies (of which NBIG are Big)
c due to gravitational perturbations by all the other bodies.
c This routine is designed for use with a mixed-variable symplectic
c integrator using Jacobi coordinates.
c
c Based upon routines from Levison and Duncan's SWIFT integrator.
c
c-----------------------------------------------------------------------------
c
      subroutine mfo_mvs (jcen,nbod,nbig,m,x,xj,a,stat)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig, stat(nbod)
      real*8 jcen(3), m(nbod), x(3,nbod), xj(3,nbod), a(3,nbod)
c
c Local
      integer i,j,k,k1
      real*8 fac0,fac1,fac12,fac2,minside,dx,dy,dz,s_1,s2,s_3,faci,facj
      real*8 a0(3),a0tp(3),a1(3,NMAX),a2(3,NMAX),a3(3,NMAX),aobl(3,NMAX)
      real*8 r,r2,r3,rj,rj2,rj3,q,q2,q3,q4,q5,q6,q7,acen(3)
c
c-----------------------------------------------------------------------------
c
c Initialize variables
      a0(1) = 0.d0
      a0(2) = 0.d0
```

```
      a0(3) = 0.d0
      a1(1,2) = 0.d0
      a1(2,2) = 0.d0
      a1(3,2) = 0.d0
      a2(1,2) = 0.d0
      a2(2,2) = 0.d0
      a2(3,2) = 0.d0
      minside = 0.d0
c
c Calculate acceleration terms
      do k = 3, nbig
        k1 = k - 1
        minside = minside + m(k1)
        r2   = x(1,k)  * x(1,k)  +  x(2,k) * x(2,k)  +  x(3,k) * x(3,k)
        rj2  = xj(1,k) * xj(1,k) + xj(2,k) * xj(2,k) + xj(3,k) * xj(3,k)
        r  = 1.d0 / sqrt(r2)
        rj = 1.d0 / sqrt(rj2)
        r3  = r  * r  * r
        rj3 = rj * rj * rj
c
        fac0 = m(k) * r3
        fac12 = m(1) * rj3
        fac2 = m(k) * fac12 / (minside + m(1))
        q = (r2 - rj2) * .5d0 / rj2
        q2 = q  * q
        q3 = q  * q2
        q4 = q2 * q2
        q5 = q2 * q3
        q6 = q3 * q3
        q7 = q3 * q4
        fac1 = 402.1875d0*q7 - 187.6875d0*q6 + 86.625d0*q5
     %       - 39.375d0*q4 + 17.5d0*q3 - 7.5d0*q2 + 3.d0*q - 1.d0
c
c Add to A0 term
        a0(1) = a0(1)  -  fac0 * x(1,k)
        a0(2) = a0(2)  -  fac0 * x(2,k)
        a0(3) = a0(3)  -  fac0 * x(3,k)
c
c Calculate A1 for this body
        a1(1,k) = fac12 * (xj(1,k) + fac1*x(1,k))
        a1(2,k) = fac12 * (xj(2,k) + fac1*x(2,k))
        a1(3,k) = fac12 * (xj(3,k) + fac1*x(3,k))
c
c Calculate A2 for this body
        a2(1,k) = a2(1,k1)  +  fac2 * xj(1,k)
        a2(2,k) = a2(2,k1)  +  fac2 * xj(2,k)
        a2(3,k) = a2(3,k1)  +  fac2 * xj(3,k)
      end do
c
      r2   = x(1,2)  * x(1,2)  +  x(2,2) * x(2,2)  +  x(3,2) * x(3,2)
      r  = 1.d0 / sqrt(r2)
      r3  = r  * r  * r
      fac0 = m(2) * r3
      a0tp(1) = a0(1)  -  fac0 * x(1,2)
      a0tp(2) = a0(2)  -  fac0 * x(2,2)
      a0tp(3) = a0(3)  -  fac0 * x(3,2)
c
c Calculate A3 (direct terms)
      do k = 2, nbod
        a3(1,k) = 0.d0
        a3(2,k) = 0.d0
        a3(3,k) = 0.d0
      end do
      do i = 2, nbig
        do j = i + 1, nbig
          dx = x(1,j) - x(1,i)
          dy = x(2,j) - x(2,i)
          dz = x(3,j) - x(3,i)
          s2 = dx*dx + dy*dy + dz*dz
          s_1 = 1.d0 / sqrt(s2)
          s_3 = s_1 * s_1 * s_1
          faci = m(i) * s_3
```

```
          facj = m(j) * s_3
          a3(1,j) = a3(1,j)  -  faci * dx
          a3(2,j) = a3(2,j)  -  faci * dy
          a3(3,j) = a3(3,j)  -  faci * dz
          a3(1,i) = a3(1,i)  +  facj * dx
          a3(2,i) = a3(2,i)  +  facj * dy
          a3(3,i) = a3(3,i)  +  facj * dz
        end do
c
        do j = nbig + 1, nbod
          dx = x(1,j) - x(1,i)
          dy = x(2,j) - x(2,i)
          dz = x(3,j) - x(3,i)
          s2 = dx*dx + dy*dy + dz*dz
          s_1 = 1.d0 / sqrt(s2)
          s_3 = s_1 * s_1 * s_1
          faci = m(i) * s_3
          a3(1,j) = a3(1,j)  -  faci * dx
          a3(2,j) = a3(2,j)  -  faci * dy
          a3(3,j) = a3(3,j)  -  faci * dz
        end do
      end do
c
c Big-body accelerations
      do k = 2, nbig
        a(1,k) = a0(1) + a1(1,k) + a2(1,k) + a3(1,k)
        a(2,k) = a0(2) + a1(2,k) + a2(2,k) + a3(2,k)
        a(3,k) = a0(3) + a1(3,k) + a2(3,k) + a3(3,k)
      end do
c
c Small-body accelerations
      do k = nbig + 1, nbod
        a(1,k) = a0tp(1) + a3(1,k)
        a(2,k) = a0tp(2) + a3(2,k)
        a(3,k) = a0tp(3) + a3(3,k)
      end do
c
c Correct for oblateness of the central body
      if (jcen(1).ne.0.or.jcen(2).ne.0.or.jcen(3).ne.0) then
        call mfo_obl (jcen,nbod,m,x,aobl,acen)
        do k = 2, nbod
          a(1,k) = a(1,k) + (aobl(1,k) - acen(1))
          a(2,k) = a(2,k) + (aobl(2,k) - acen(2))
          a(3,k) = a(3,k) + (aobl(3,k) - acen(3))
        end do
      end if
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MFO_NGF.FOR    (ErikSoft  29 November 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates accelerations on a set of NBOD bodies due to cometary
c non-gravitational jet forces. The positions and velocities are stored in
c arrays X, V with the format (x,y,z) and (vx,vy,vz) for each object in
c succession. The accelerations are stored in the array A (ax,ay,az). The
c non-gravitational accelerations follow a force law described by Marsden
c et al. (1973) Astron. J. 211-225, with magnitude determined by the
c parameters NGF(1,2,3) for each object.
c
c N.B. All coordinates and velocities must be with respect to central body!!!!
c ===
c-----------------------------------------------------------------------
c
```

```
      subroutine mfo_ngf (nbod,x,v,a,ngf)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod
      real*8 x(3,nbod), v(3,nbod), a(3,nbod), ngf(4,nbod)
c
c Local
      integer j
      real*8 r2,r,rv,q,g,tx,ty,tz,nx,ny,nz,a1,a2,a3
c
c-----------------------------------------------------------------------------
c
      do j = 2, nbod
        r2 = x(1,j)*x(1,j) + x(2,j)*x(2,j) +x(3,j)*x(3,j)
c
c Only calculate accelerations if body is close to the Sun (R < 9.36 AU),
c or if the non-gravitational force parameters are exceptionally large.
        if (r2.lt.88.d0.or.abs(ngf(1,j)).gt.1d-7
     %    .or.abs(ngf(2,j)).gt.1d-7.or.abs(ngf(3,j)).gt.1d-7) then
          r = sqrt(r2)
          rv = x(1,j)*v(1,j) + x(2,j)*v(2,j) + x(3,j)*v(3,j)
c
c Calculate Q = R / R0, where R0 = 2.808 AU
          q = r * .3561253561253561d0
          g = .111262d0 * q**(-2.15d0) * (1.d0+q**5.093d0)**(-4.6142d0)
c
c Within-orbital-plane transverse vector components
          tx = r2*v(1,j) - rv*x(1,j)
          ty = r2*v(2,j) - rv*x(2,j)
          tz = r2*v(3,j) - rv*x(3,j)
c
c Orbit-normal vector components
          nx = x(2,j)*v(3,j) - x(3,j)*v(2,j)
          ny = x(3,j)*v(1,j) - x(1,j)*v(3,j)
          nz = x(1,j)*v(2,j) - x(2,j)*v(1,j)
c
c Multiplication factors
          a1 = ngf(1,j) * g / r
          a2 = ngf(2,j) * g / sqrt(tx*tx + ty*ty + tz*tz)
          a3 = ngf(3,j) * g / sqrt(nx*nx + ny*ny + nz*nz)
c
c X,Y and Z components of non-gravitational acceleration
          a(1,j) = a1*x(1,j) + a2*tx + a3*nx
          a(2,j) = a1*x(2,j) + a2*ty + a3*ny
          a(3,j) = a1*x(3,j) + a2*tz + a3*nz
        else
          a(1,j) = 0.d0
          a(2,j) = 0.d0
          a(3,j) = 0.d0
        end if
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c     MFO_OBL.FOR    (ErikSoft   2 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates barycentric accelerations of NBOD bodies due to oblateness of
c the central body. Also returns the corresponding barycentric acceleration
c of the central body.
c
```

```
c N.B. All coordinates must be with respect to the central body!!!!
c ===
c-----------------------------------------------------------------------------
c
      subroutine mfo_obl (jcen,nbod,m,x,a,acen)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod
      real*8 jcen(3), m(nbod), x(3,nbod), a(3,nbod), acen(3)
c
c Local
      integer i
      real*8 jr2,jr4,jr6,r2,r_1,r_2,r_3,u2,u4,u6,tmp1,tmp2,tmp3,tmp4
c
c-----------------------------------------------------------------------------
c
      acen(1) = 0.d0
      acen(2) = 0.d0
      acen(3) = 0.d0
c
      do i = 2, nbod
c
c Calculate barycentric accelerations on the objects
        r2 = x(1,i)*x(1,i) + x(2,i)*x(2,i) + x(3,i)*x(3,i)
        r_1 = 1.d0 / sqrt(r2)
        r_2 = r_1 * r_1
        r_3 = r_2 * r_1
        jr2 = jcen(1) * r_2
        jr4 = jcen(2) * r_2 * r_2
        jr6 = jcen(3) * r_2 * r_2 * r_2
        u2 = x(3,i) * x(3,i) * r_2
        u4 = u2 * u2
        u6 = u4 * u2
c
        tmp1 = m(1) * r_3
        tmp2 =jr2*(7.5d0*u2 - 1.5d0)
     %       +jr4*(39.375d0*u4 - 26.25d0*u2 + 1.875d0)
     %       +jr6*(187.6875d0*u6 -216.5625d0*u4 +59.0625d0*u2 -2.1875d0)
        tmp3 = jr2*3.d0 + jr4*(17.5d0*u2 - 7.5d0)
     %       + jr6*(86.625d0*u4 - 78.75d0*u2 + 13.125d0)
c
        a(1,i) = x(1,i) * tmp1 * tmp2
        a(2,i) = x(2,i) * tmp1 * tmp2
        a(3,i) = x(3,i) * tmp1 * (tmp2 - tmp3)
c
c Calculate barycentric accelerations on the central body
        tmp4 = m(i) / m(1)
        acen(1) = acen(1)  -  tmp4 * a(1,i)
        acen(2) = acen(2)  -  tmp4 * a(2,i)
        acen(3) = acen(3)  -  tmp4 * a(3,i)
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MFO_PN.FOR    (ErikSoft   3 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c ****** To be completed at a later date ******
c
c Calculates post-Newtonian relativistic corrective accelerations for a set
c of NBOD bodies (NBIG of which are Big).
```

```
c
c This routine should not be called from the symplectic algorithm MAL_MVS
c or the conservative Bulirsch-Stoer algorithm MAL_BS2.
c
c N.B. All coordinates and velocities must be with respect to central body!!!!
c ===
c-----------------------------------------------------------------------------
c
      subroutine mfo_pn (nbod,nbig,m,x,v,a)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig
      real*8 m(nbod), x(3,nbod), v(3,nbod), a(3,nbod)
c
c Local
      integer j
c
c-----------------------------------------------------------------------------
c
      do j = 1, nbod
        a(1,j) = 0.d0
        a(2,j) = 0.d0
        a(3,j) = 0.d0
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MFO_PR.FOR    (ErikSoft   3 October 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c ****** To be completed at a later date ******
c
c Calculates radiation pressure and Poynting-Robertson drag for a set
c of NBOD bodies (NBIG of which are Big).
c
c This routine should not be called from the symplectic algorithm MAL_MVS
c or the conservative Bulirsch-Stoer algorithm MAL_BS2.
c
c N.B. All coordinates and velocities must be with respect to central body!!!!
c ===
c-----------------------------------------------------------------------------
c
      subroutine mfo_pr (nbod,nbig,m,x,v,a,ngf)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig
      real*8 m(nbod), x(3,nbod), v(3,nbod), a(3,nbod), ngf(4,nbod)
c
c Local
      integer j
c
c-----------------------------------------------------------------------------
c
      do j = 1, nbod
        a(1,j) = 0.d0
        a(2,j) = 0.d0
        a(3,j) = 0.d0
      end do
```

```
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_C2FL.FOR    (ErikSoft  1 July 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Converts a 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
      integer ex
      real*8 x,mio_c2re
c
c-----------------------------------------------------------------------
c
      x = mio_c2re (c(1:8), 0.d0, 1.d0, 7)
      x = x * 2.d0 - 1.d0
      ex = ichar(c(8:8)) - 32 - 112
      mio_c2fl = x * (10.d0**dble(ex))
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_C2RE.FOR    (ErikSoft  1 July 1999)
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(ichar(c(j:j)) - 32)) / 224.d0
      end do
c
      mio_c2re = xmin  +  y * (xmax - xmin)
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_CE.FOR    (ErikSoft   1 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Writes details of close encounter minima to an output file, and decides how
c to continue the integration depending upon the close-encounter option
c chosen by the user. Close encounter details are stored until either 100
c have been accumulated, or a data dump is done, at which point the stored
c encounter details are also output.
c
c For each encounter, the routine outputs the time and distance of closest
c approach, the identities of the objects involved, and the output
c variables of the objects at this time. The output variables are:
c expressed as
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 mio_ce (time,tstart,rcen,rmax,nbod,nbig,m,stat,id,
     %  nclo,iclo,jclo,opt,stopflag,tclo,dclo,ixvclo,jxvclo,mem,
     %  lmem,outfile,nstored,ceflush)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,opt(8),stat(nbod),lmem(NMESS),stopflag
      integer nclo,iclo(nclo),jclo(nclo),nstored,ceflush
      real*8 time,tstart,rcen,rmax,m(nbod),tclo(nclo),dclo(nclo)
      real*8 ixvclo(6,nclo),jxvclo(6,nclo)
      character*80 outfile(3),mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer k,year,month
      real*8 tmp0,t1,rfac,fr,fv,theta,phi,vtheta,vphi
      character*80 c(200)
      character*38 fstop
      character*8 mio_fl2c, mio_re2c
      character*6 tstring
c
c-----------------------------------------------------------------------
c
      save c
c
```

```
c Scaling factor (maximum possible range) for distances
      rfac = log10 (rmax / rcen)
c
c Store details of each new close-encounter minimum
      do k = 1, nclo
        nstored = nstored + 1
        c(nstored)(1:8)   = mio_fl2c(tclo(k))
        c(nstored)(9:16)  = mio_re2c(dble(iclo(k)-1),0.d0,11239423.99d0)
        c(nstored)(12:19) = mio_re2c(dble(jclo(k)-1),0.d0,11239423.99d0)
        c(nstored)(15:22) = mio_fl2c(dclo(k))
c
        call mco_x2ov (rcen,rmax,m(1),0.d0,ixvclo(1,k),ixvclo(2,k),
     %    ixvclo(3,k),ixvclo(4,k),ixvclo(5,k),ixvclo(6,k),fr,theta,phi,
     %    fv,vtheta,vphi)
        c(nstored)(23:30) = mio_re2c (fr    , 0.d0, rfac)
        c(nstored)(27:34) = mio_re2c (theta , 0.d0, PI)
        c(nstored)(31:38) = mio_re2c (phi   , 0.d0, TWOPI)
        c(nstored)(35:42) = mio_re2c (fv    , 0.d0, 1.d0)
        c(nstored)(39:46) = mio_re2c (vtheta, 0.d0, PI)
        c(nstored)(43:50) = mio_re2c (vphi  , 0.d0, TWOPI)
c
        call mco_x2ov (rcen,rmax,m(1),0.d0,jxvclo(1,k),jxvclo(2,k),
     %    jxvclo(3,k),jxvclo(4,k),jxvclo(5,k),jxvclo(6,k),fr,theta,phi,
     %    fv,vtheta,vphi)
        c(nstored)(47:54) = mio_re2c (fr    , 0.d0, rfac)
        c(nstored)(51:58) = mio_re2c (theta , 0.d0, PI)
        c(nstored)(55:62) = mio_re2c (phi   , 0.d0, TWOPI)
        c(nstored)(59:66) = mio_re2c (fv    , 0.d0, 1.d0)
        c(nstored)(63:74) = mio_re2c (vtheta, 0.d0, PI)
        c(nstored)(67:78) = mio_re2c (vphi  , 0.d0, TWOPI)
      end do
c
c If required, output the stored close encounter details
      if (nstored.ge.100.or.ceflush.eq.0) then
  10    open (22, file=outfile(2), status='old', access='append',err=10)
        do k = 1, nstored
          write (22,'(a1,a2,a70)') char(12),'6b',c(k)(1:70)
        end do
        close (22)
        nstored = 0
      end if
c
c If new encounter minima have occurred, decide whether to stop integration
      stopflag = 0
      if (opt(1).eq.1.and.nclo.gt.0) then
  20    open (23, file=outfile(3), status='old', access='append',err=20)
c If time style is Gregorian date then...
        tmp0 = tclo(1)
        if (opt(3).eq.1) then
          fstop = '(5a,/,9x,a,i10,1x,i2,1x,f4.1)'
          call mio_jd2y (tmp0,year,month,t1)
          write (23,fstop) mem(121)(1:lmem(121)),mem(126)
     %      (1:lmem(126)),id(iclo(1)),',',id(jclo(1)),
     %      mem(71)(1:lmem(71)),year,month,t1
c Otherwise...
        else
          if (opt(3).eq.3) then
            tstring = mem(2)
            fstop = '(5a,/,9x,a,f14.3,a)'
            t1 = (tmp0 - tstart) / 365.25d0
          else
            tstring = mem(1)
            fstop = '(5a,/,9x,a,f14.1,a)'
            if (opt(3).eq.0) t1 = tmp0
            if (opt(3).eq.2) t1 = tmp0 - tstart
          end if
          write (23,fstop) mem(121)(1:lmem(121)),mem(126)
     %      (1:lmem(126)),id(iclo(1)),',',id(jclo(1)),
     %      mem(71)(1:lmem(71)),t1,tstring
        end if
        stopflag = 1
        close(23)
```

```
      end if
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_DUMP.FOR    (ErikSoft   21 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Writes masses, coordinates, velocities etc. of all objects, and integration
c parameters, to dump files. Also updates a restart file containing other
c variables used internally by MERCURY.
c
c-----------------------------------------------------------------------
c
      subroutine mio_dump (time,tstart,tstop,dtout,algor,h0,tol,jcen,
     %  rcen,rmax,en,am,cefac,ndump,nfun,nbod,nbig,m,x,v,s,rho,rceh,
     %  stat,id,ngf,epoch,opt,opflag,dumpfile,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer algor,nbod,nbig,stat(nbod),opt(8),opflag,ndump,nfun
      integer lmem(NMESS)
      real*8 time,tstart,tstop,dtout,h0,tol,rmax,en(3),am(3)
      real*8 jcen(3),rcen,cefac,m(nbod),x(3,nbod),v(3,nbod)
      real*8 s(3,nbod),rho(nbod),rceh(nbod),ngf(4,nbod),epoch(nbod)
      character*80 dumpfile(4),mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer idp,i,j,k,len,j1,j2
      real*8 rhocgs,k_2,rcen_2,rcen_4,rcen_6,x0(3,NMAX),v0(3,NMAX)
      character*150 c
c
c-----------------------------------------------------------------------
c
      rhocgs = AU * AU * AU * K2 / MSUN
      k_2 = 1.d0 / K2
      rcen_2 = 1.d0 / (rcen * rcen)
      rcen_4 = rcen_2 * rcen_2
      rcen_6 = rcen_4 * rcen_2
c
c If using close-binary star, convert to user coordinates
c      if (algor.eq.11) call mco_h2ub (time,jcen,nbod,nbig,h0,m,x,v,
c     %   x0,v0)
c
c Dump to temporary files (idp=1) and real dump files (idp=2)
      do idp = 1, 2
c
c Dump data for the Big (i=1) and Small (i=2) bodies
        do i = 1, 2
          if (idp.eq.1) then
            if (i.eq.1) c(1:12) = 'big.tmp     '
            if (i.eq.2) c(1:12) = 'small.tmp   '
  20        open (31, file=c(1:12), status='unknown', err=20)
          else
  25        open (31, file=dumpfile(i), status='old', err=25)
          end if
c
c Write header lines, data style (and epoch for Big bodies)
          write (31,'(a)') mem(151+i)(1:lmem(151+i))
          if (i.eq.1) then
            j1 = 2
            j2 = nbig
```

```
          else
            j1 = nbig + 1
            j2 = nbod
          end if
          write (31,'(a)') mem(154)(1:lmem(154))
          write (31,'(a)') mem(155)(1:lmem(155))
          write (31,*) mem(156)(1:lmem(156)),'Cartesian'
          if (i.eq.1) write (31,*) mem(157)(1:lmem(157)),time
          write (31,'(a)') mem(155)(1:lmem(155))
c
c For each body...
          do j = j1, j2
            len = 37
            c(1:8) = id(j)
            write (c(9:37),'(1p,a3,e11.5,a3,e11.5)') ' r=',rceh(j),
     %        ' d=',rho(j)/rhocgs
            if (m(j).gt.0) then
              write (c(len+1:len+25),'(a3,e22.15)') ' m=',m(j)*k_2
              len = len + 25
            end if
            do k = 1, 3
              if (ngf(k,j).ne.0) then
                write (c(len+1:len+16),'(a2,i1,a1,e12.5)') ' a',k,'=',
     %            ngf(k,j)
                len = len + 16
              end if
            end do
            if (ngf(4,j).ne.0) then
              write (c(len+1:len+15),'(a3,e12.5)') ' b=',ngf(4,j)
              len = len + 15
            end if
            write (31,'(a)') c(1:len)
            if (algor.eq.11) then
              write (31,312) x0(1,j), x0(2,j), x0(3,j)
              write (31,312) v0(1,j), v0(2,j), v0(3,j)
            else
              write (31,312) x(1,j), x(2,j), x(3,j)
              write (31,312) v(1,j), v(2,j), v(3,j)
            end if
            write (31,312) s(1,j)*k_2, s(2,j)*k_2, s(3,j)*k_2
          enddo
          close (31)
        end do
c
c Dump the integration parameters
  40    if (idp.eq.1) open (33,file='param.tmp',status='unknown',err=40)
  45    if (idp.eq.2) open (33, file=dumpfile(3), status='old', err=45)
c
c Important parameters
        write (33,'(a)') mem(151)(1:lmem(151))
        write (33,'(a)') mem(154)(1:lmem(154))
        write (33,'(a)') mem(155)(1:lmem(155))
        write (33,'(a)') mem(158)(1:lmem(158))
        write (33,'(a)') mem(155)(1:lmem(155))
        if (algor.eq.1) then
          write (33,*) mem(159)(1:lmem(159)),'MVS'
        else if (algor.eq.2) then
          write (33,*) mem(159)(1:lmem(159)),'BS'
        else if (algor.eq.3) then
          write (33,*) mem(159)(1:lmem(159)),'BS2'
        else if (algor.eq.4) then
          write (33,*) mem(159)(1:lmem(159)),'RADAU'
        else if (algor.eq.10) then
          write (33,*) mem(159)(1:lmem(159)),'HYBRID'
        else if (algor.eq.11) then
          write (33,*) mem(159)(1:lmem(159)),'CLOSE'
        else if (algor.eq.12) then
          write (33,*) mem(159)(1:lmem(159)),'WIDE'
        else
          write (33,*) mem(159)(1:lmem(159)),'0'
        end if
        write (33,*) mem(160)(1:lmem(160)),tstart
```

```
        write (33,*) mem(161)(1:lmem(161)),tstop
        write (33,*) mem(162)(1:lmem(162)),dtout
        write (33,*) mem(163)(1:lmem(163)),h0
        write (33,*) mem(164)(1:lmem(164)),tol
c
c Integration options
        write (33,'(a)') mem(155)(1:lmem(155))
        write (33,'(a)') mem(165)(1:lmem(165))
        write (33,'(a)') mem(155)(1:lmem(155))
        if (opt(1).eq.0) then
          write (33,'(2a)') mem(166)(1:lmem(166)),mem(5)(1:lmem(5))
        else
          write (33,'(2a)') mem(166)(1:lmem(166)),mem(6)(1:lmem(6))
        end if
        if (opt(2).eq.0) then
          write (33,'(2a)') mem(167)(1:lmem(167)),mem(5)(1:lmem(5))
          write (33,'(2a)') mem(168)(1:lmem(168)),mem(5)(1:lmem(5))
        else if (opt(2).eq.2) then
          write (33,'(2a)') mem(167)(1:lmem(167)),mem(6)(1:lmem(6))
          write (33,'(2a)') mem(168)(1:lmem(168)),mem(6)(1:lmem(6))
        else
          write (33,'(2a)') mem(167)(1:lmem(167)),mem(6)(1:lmem(6))
          write (33,'(2a)') mem(168)(1:lmem(168)),mem(5)(1:lmem(5))
        end if
        if (opt(3).eq.0.or.opt(3).eq.2) then
          write (33,'(2a)') mem(169)(1:lmem(169)),mem(1)(1:lmem(1))
        else
          write (33,'(2a)') mem(169)(1:lmem(169)),mem(2)(1:lmem(2))
        end if
        if (opt(3).eq.2.or.opt(3).eq.3) then
          write (33,'(2a)') mem(170)(1:lmem(170)),mem(6)(1:lmem(6))
        else
          write (33,'(2a)') mem(170)(1:lmem(170)),mem(5)(1:lmem(5))
        end if
        if (opt(4).eq.1) then
          write (33,'(2a)') mem(171)(1:lmem(171)),mem(7)(1:lmem(7))
        else if (opt(4).eq.3) then
          write (33,'(2a)') mem(171)(1:lmem(171)),mem(9)(1:lmem(9))
        else
          write (33,'(2a)') mem(171)(1:lmem(171)),mem(8)(1:lmem(8))
        end if
        write (33,'(a)') mem(172)(1:lmem(172))
        if (opt(7).eq.1) then
          write (33,'(2a)') mem(173)(1:lmem(173)),mem(6)(1:lmem(6))
        else
          write (33,'(2a)') mem(173)(1:lmem(173)),mem(5)(1:lmem(5))
        end if
        if (opt(8).eq.1) then
          write (33,'(2a)') mem(174)(1:lmem(174)),mem(6)(1:lmem(6))
        else
          write (33,'(2a)') mem(174)(1:lmem(174)),mem(5)(1:lmem(5))
        end if
c
c Infrequently-changed parameters
        write (33,'(a)') mem(155)(1:lmem(155))
        write (33,'(a)') mem(175)(1:lmem(175))
        write (33,'(a)') mem(155)(1:lmem(155))
        write (33,*) mem(176)(1:lmem(176)),rmax
        write (33,*) mem(177)(1:lmem(177)),rcen
        write (33,*) mem(178)(1:lmem(178)),m(1) * k_2
        write (33,*) mem(179)(1:lmem(179)),jcen(1) * rcen_2
        write (33,*) mem(180)(1:lmem(180)),jcen(2) * rcen_4
        write (33,*) mem(181)(1:lmem(181)),jcen(3) * rcen_6
        write (33,*) mem(182)(1:lmem(182))
        write (33,*) mem(183)(1:lmem(183))
        write (33,*) mem(184)(1:lmem(184)),cefac
        write (33,*) mem(185)(1:lmem(185)),ndump
        write (33,*) mem(186)(1:lmem(186)),nfun
        close (33)
c
c Create new version of the restart file
  60    if (idp.eq.1) open (35, file='restart.tmp', status='unknown',
```

```
     %    err=60)
  65    if (idp.eq.2) open (35, file=dumpfile(4), status='old', err=65)
        write (35,'(1x,i2)') opflag
        write (35,*) en(1) * k_2
        write (35,*) am(1) * k_2
        write (35,*) en(3) * k_2
        write (35,*) am(3) * k_2
        write (35,*) s(1,1) * k_2
        write (35,*) s(2,1) * k_2
        write (35,*) s(3,1) * k_2
        close (35)
      end do
c
c-----------------------------------------------------------------------
c
 311  format (1x,a8,1x,a1,1p,e22.15,2(1x,e11.5))
 312  format (1p,3(1x,e22.15),1x,i8)
 313  format (1p,1x,e22.15,0p,2x,a)
 314  format (1x,a8,1x,a1,1p,e22.15,4(1x,e12.5),1x,e22.15,2(1x,e11.5))
      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 (*,'(/,2a)') ' ERROR: Programme terminated. See information'
     %  ,' file for details.'
c
      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      MIO_FL2C.FOR    (ErikSoft  1 July 1998)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts a (floating point) REAL*8 variable X, into a CHARACTER*8 ASCII
c string, using the new format compression:
c
c X is first converted to base 224, and then each base 224 digit is converted
c to an ASCII character, such that 0 -> character 32, 1 -> character 33...
c and 223 -> character 255.
c The first 7 characters in the string are used to store the mantissa, and the
c eighth character is used for the exponent.
c
```

```
c ASCII characters 0 - 31 (CTRL characters) are not used, because they
c cause problems when using some operating systems.
c
c N.B. X must lie in the range -1.e112 < X < 1.e112
c ===
c
c-----------------------------------------------------------------------------
c
      function mio_fl2c (x)
c
      implicit none
c
c Input/Output
      real*8 x
      character*8 mio_fl2c
c
c Local
      integer ex
      real*8 ax,y
      character*8 mio_re2c
c
c-----------------------------------------------------------------------------
c
      if (x.eq.0) then
        y = .5d0
      else
        ax = abs(x)
        ex = int(log10(ax))
        if (ax.ge.1) ex = ex + 1
        y = ax*(10.d0**(-ex))
        if (y.eq.1) then
          y = y * .1d0
          ex = ex + 1
        end if
        y = sign(y,x) *.5d0 + .5d0
      end if
c
      mio_fl2c(1:8) = mio_re2c (y, 0.d0, 1.d0)
      ex = ex + 112
      if (ex.gt.223) ex = 223
      if (ex.lt.0) ex = 0
      mio_fl2c(8:8) = char(ex+32)
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_IN.FOR    (ErikSoft   4 May 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Reads names, masses, coordinates and velocities of all the bodies,
c and integration parameters for the MERCURY integrator package.
c If DUMPFILE(4) exists, the routine assumes this is a continuation of
c an old integration, and reads all the data from the dump files instead
c of the input files.
c
c N.B. All coordinates are with respect to the central body!!
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mio_in (time,tstart,tstop,dtout,algor,h0,tol,rmax,rcen,
     %  jcen,en,am,cefac,ndump,nfun,nbod,nbig,m,x,v,s,rho,rceh,stat,id,
     %  epoch,ngf,opt,opflag,ngflag,outfile,dumpfile,lmem,mem)
c
      implicit none
```

```
      include 'mercury.inc'
c
c Input/Output
      integer algor,nbod,nbig,stat(NMAX),opt(8),opflag,ngflag
      integer lmem(NMESS),ndump,nfun
      real*8 time,tstart,tstop,dtout,h0,tol,rmax,rcen,jcen(3)
      real*8 en(3),am(3),m(NMAX),x(3,NMAX),v(3,NMAX),s(3,NMAX)
      real*8 rho(NMAX),rceh(NMAX),epoch(NMAX),ngf(4,NMAX),cefac
      character*80 outfile(3),dumpfile(4), mem(NMESS)
      character*8 id(NMAX)
c
c Local
      integer j,k,itmp,jtmp,informat,lim(2,10),nsub,year,month,lineno
      real*8 q,a,e,i,p,n,l,temp,tmp2,tmp3,rhocgs,t1,tmp4,tmp5,tmp6
c      real*8 v0(3,NMAX),x0(3,NMAX)
      logical test,oldflag,flag1,flag2
      character*1 c1
      character*3 c3,alg(60)
      character*80 infile(3),filename,c80
      character*150 string
c
c-----------------------------------------------------------------------------
c
      data alg/'MVS','Mvs','mvs','mvs','mvs','BS ','Bs ','bs ','Bul',
     %   'bul','BS2','Bs2','bs2','Bu2','bu2','RAD','Rad','rad','RA ',
     %   'ra ','xxx','xxx','xxx','xxx','xxx','xxx','xxx','xxx','xxx',
     %   'xxx','xxx','xxx','xxx','xxx','xxx','xxx','xxx','xxx','xxx',
     %   'xxx','TES','Tes','tes','Tst','tst','HYB','Hyb','hyb','HY ',
     %   'hy ','CLO','Clo','clo','CB ','cb ','WID','Wid','wid','WB ',
     %   'wb '/
c
      rhocgs = AU * AU * AU * K2 / MSUN
      do j = 1, 80
        filename(j:j) = ' '
      end do
      do j = 1, 3
        infile(j)   = filename
        outfile(j)  = filename
        dumpfile(j) = filename
      end do
      dumpfile(4) = filename
c
c Read in output messages
      inquire (file='message.in', exist=test)
      if (.not.test) then
        write (*,'(/,2a)') ' ERROR: This file is needed to start',
     %     ' the integration:  message.in'
        stop
      end if
      open (16, file='message.in', status='old')
  10  read (16,'(i3,1x,i2,1x,a80)',end=20) j,lmem(j),mem(j)
      goto 10
  20  close (16)
c
c Read in filenames and check for duplicate filenames
      inquire (file='files.in', exist=test)
      if (.not.test) call mio_err (6,mem(81),lmem(81),mem(88),lmem(88),
     %  ' ',1,'files.in',8)
      open (15, file='files.in', status='old')
c
c Input files
      do j = 1, 3
        read (15,'(a150)') string
        call mio_spl (150,string,nsub,lim)
        infile(j)(1:(lim(2,1)-lim(1,1)+1)) = string(lim(1,1):lim(2,1))
        do k = 1, j - 1
          if (infile(j).eq.infile(k)) call mio_err (6,mem(81),lmem(81),
     %      mem(89),lmem(89),infile(j),80,mem(86),lmem(86))
        end do
      end do
c
c Output files
```

```
      do j = 1, 3
        read (15,'(a150)') string
        call mio_spl (150,string,nsub,lim)
        outfile(j)(1:(lim(2,1)-lim(1,1)+1)) = string(lim(1,1):lim(2,1))
        do k = 1, j - 1
          if (outfile(j).eq.outfile(k)) call mio_err (6,mem(81),
     %      lmem(81),mem(89),lmem(89),outfile(j),80,mem(86),lmem(86))
        end do
        do k = 1, 3
          if (outfile(j).eq.infile(k)) call mio_err (6,mem(81),lmem(81),
     %      mem(89),lmem(89),outfile(j),80,mem(86),lmem(86))
        end do
      end do
c
c Dump files
      do j = 1, 4
        read (15,'(a150)') string
        call mio_spl (150,string,nsub,lim)
        dumpfile(j)(1:(lim(2,1)-lim(1,1)+1)) = string(lim(1,1):lim(2,1))
        do k = 1, j - 1
          if (dumpfile(j).eq.dumpfile(k)) call mio_err (6,mem(81),
     %      lmem(81),mem(89),lmem(89),dumpfile(j),80,mem(86),lmem(86))
        end do
        do k = 1, 3
          if (dumpfile(j).eq.infile(k)) call mio_err (6,mem(81),
     %      lmem(81),mem(89),lmem(89),dumpfile(j),80,mem(86),lmem(86))
        end do
        do k = 1, 3
          if (dumpfile(j).eq.outfile(k)) call mio_err (6,mem(81),
     %      lmem(81),mem(89),lmem(89),dumpfile(j),80,mem(86),lmem(86))
        end do
      end do
      close (15)
c
c Find out if this is an old integration (i.e. does the restart file exist)
      inquire (file=dumpfile(4), exist=oldflag)
c
c Check if information file exists, and append a continuation message
      if (oldflag) then
        inquire (file=outfile(3), exist=test)
        if (.not.test) call mio_err (6,mem(81),lmem(81),mem(88),
     %    lmem(88),' ',1,outfile(3),80)
 320    open(23,file=outfile(3),status='old',access='append',err=320)
      else
c
c If new integration, check information file doesn't exist, and then create it
        inquire (file=outfile(3), exist=test)
        if (test) call mio_err (6,mem(81),lmem(81),mem(87),lmem(87),
     %    ' ',1,outfile(3),80)
 410    open(23, file = outfile(3), status = 'new', err=410)
      end if
c
c--------------------------------------------------------------------------
c
c  READ  IN  INTEGRATION  PARAMETERS
c
c Check if the file containing integration parameters exists, and open it
      filename = infile(3)
      if (oldflag) filename = dumpfile(3)
      inquire (file=filename, exist=test)
      if (.not.test) call mio_err (23,mem(81),lmem(81),mem(88),lmem(88),
     %  ' ',1,filename,80)
  30  open  (13, file=filename, status='old', err=30)
c
c Read integration parameters
      lineno = 0
      do j = 1, 26
  40    lineno = lineno + 1
        read (13,'(a150)') string
        if (string(1:1).eq.')') goto 40
        call mio_spl (150,string,nsub,lim)
        c80(1:3) = '   '
```

```
        c80 = string(lim(1,nsub):lim(2,nsub))
        if (j.eq.1) then
          algor = 0
           do k = 1, 60
            if (c80(1:3).eq.alg(k)) algor = (k + 4) / 5
          end do
          if (algor.eq.0) call mio_err (23,mem(81),lmem(81),mem(98),
     %      lmem(98),c80(lim(1,nsub):lim(2,nsub)),lim(2,nsub)-
     %      lim(1,nsub)+1,mem(85),lmem(85))
        end if
        if (j.eq.2) read (c80,*,err=661) tstart
        if (j.eq.3) read (c80,*,err=661) tstop
        if (j.eq.4) read (c80,*,err=661) dtout
        if (j.eq.5) read (c80,*,err=661) h0
        if (j.eq.6) read (c80,*,err=661) tol
        c1 = c80(1:1)
        if (j.eq.7.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(1) = 1
        if (j.eq.8.and.(c1.eq.'n'.or.c1.eq.'N')) opt(2) = 0
        if (j.eq.9.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(2) = 2
        if (j.eq.10.and.(c1.eq.'d'.or.c1.eq.'D')) opt(3) = 0
        if (j.eq.11.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(3) = opt(3) + 2
        if (j.eq.12) then
          if(c1.eq.'l'.or.c1.eq.'L') then
            opt(4) = 1
          else if (j.eq.12.and.(c1.eq.'m'.or.c1.eq.'M')) then
            opt(4) = 2
          else if (j.eq.12.and.(c1.eq.'h'.or.c1.eq.'H')) then
            opt(4) = 3
          else
            goto 661
          end if
        end if
        if (j.eq.15.and.(c1.eq.'y'.or.c1.eq.'Y')) opt(8) = 1
        if (j.eq.16) read (c80,*,err=661) rmax
        if (j.eq.17) read (c80,*,err=661) rcen
        if (j.eq.18) read (c80,*,err=661) m(1)
        if (j.eq.19) read (c80,*,err=661) jcen(1)
        if (j.eq.20) read (c80,*,err=661) jcen(2)
        if (j.eq.21) read (c80,*,err=661) jcen(3)
        if (j.eq.24) read (c80,*,err=661) cefac
        if (j.eq.25) read (c80,*,err=661) ndump
        if (j.eq.26) read (c80,*,err=661) nfun
      end do
      h0 = abs(h0)
      tol = abs(tol)
      rmax = abs(rmax)
      rcen = abs(rcen)
      cefac = abs(cefac)
      close (13)
c
c Change quantities for central object to suitable units
      m(1) = abs(m(1)) * K2
      jcen(1) = jcen(1) * rcen * rcen
      jcen(2) = jcen(2) * rcen * rcen * rcen * rcen
      jcen(3) = jcen(3) * rcen * rcen * rcen * rcen * rcen * rcen
      s(1,1) = 0.d0
      s(2,1) = 0.d0
      s(3,1) = 0.d0
c
c Make sure that RCEN isn't too small, since it is used to scale the output
c (Minimum value corresponds to a central body with density 100g/cm^3).
      temp = 1.1235d-3 * m(1) ** .333333333333333d0
      if (rcen.lt.temp) then
        rcen = temp
        write (13,'(/,2a)') mem(121)(1:lmem(121)),mem(131)(1:lmem(131))
      end if
c
c-----------------------------------------------------------------------
c
c  READ  IN  DATA  FOR  BIG  AND  SMALL  BODIES
c
      nbod = 1
```

```
      do j = 1, 2
        if (j.eq.2) nbig = nbod
c
c Check if the file containing data for Big bodies exists, and open it
        filename = infile(j)
        if (oldflag) filename = dumpfile(j)
        inquire (file=filename, exist=test)
        if (.not.test) call mio_err (23,mem(81),lmem(81),mem(88),
     %    lmem(88),' ',1,filename,80)
 110    open (11, file=filename, status='old', err=110)
c
c Read data style
 120    read (11,'(a150)') string
        if (string(1:1).eq.')') goto 120
        call mio_spl (150,string,nsub,lim)
        c3 = string(lim(1,nsub):(lim(1,nsub)+2))
        if (c3.eq.'Car'.or.c3.eq.'car'.or.c3.eq.'CAR') then
          informat = 1
        else if (c3.eq.'Ast'.or.c3.eq.'ast'.or.c3.eq.'AST') then
          informat = 2
        else if (c3.eq.'Com'.or.c3.eq.'com'.or.c3.eq.'COM') then
          informat = 3
        else
          call mio_err (23,mem(81),lmem(81),mem(91),lmem(91),' ',1,
     %      mem(82+j),lmem(82+j))
        end if
c
c Read epoch of Big bodies
        if (j.eq.1) then
 125      read (11,'(a150)') string
          if (string(1:1).eq.')') goto 125
          call mio_spl (150,string,nsub,lim)
          read (string(lim(1,nsub):lim(2,nsub)),*,err=667) time
        end if
c
c Read information for each object
 130    read (11,'(a)',end=140) string
        if (string(1:1).eq.')') goto 130
        call mio_spl (150,string,nsub,lim)
        if (lim(1,1).eq.-1) goto 140
c
c Determine the name of the object
        nbod = nbod + 1
        if (nbod.gt.NMAX) call mio_err (23,mem(81),lmem(81),mem(90),
     %    lmem(90),' ',1,mem(82),lmem(82))
c
        if ((lim(2,1)-lim(1,1)).gt.7) then
          write (23,'(/,3a)') mem(121)(1:lmem(121)),
     %      mem(122)(1:lmem(122)),string( lim(1,1):lim(2,1) )
        end if
        id(nbod) = string( lim(1,1):min(7+lim(1,1),lim(2,1)) )
c Check if another object has the same name
        do k = 1, nbod - 1
          if (id(k).eq.id(nbod)) call mio_err (23,mem(81),lmem(81),
     %      mem(103),lmem(103),id(nbod),8,' ',1)
        end do
c
c Default values of mass, close-encounter limit, density etc.
        m(nbod) = 0.d0
        rceh(nbod) = 1.d0
        rho(nbod) = rhocgs
        epoch(nbod) = time
        do k = 1, 4
          ngf(k,nbod) = 0.d0
        end do
c
c Read values of mass, close-encounter limit, density etc.
        do k = 3, nsub, 2
          c80 = string(lim(1,k-1):lim(2,k-1))
          read (string(lim(1,k):lim(2,k)),*,err=666) temp
          if (c80(1:1).eq.'m'.or.c80(1:1).eq.'M') then
            m(nbod) = temp * K2
```

```
          else if (c80(1:1).eq.'r'.or.c80(1:1).eq.'R') then
            rceh(nbod) = temp
          else if (c80(1:1).eq.'d'.or.c80(1:1).eq.'D') then
            rho(nbod) = temp * rhocgs
          else if (m(nbod).lt.0.or.rceh(nbod).lt.0.or.rho(nbod).lt.0)
     %      then
            call mio_err (23,mem(81),lmem(81),mem(97),lmem(97),id(nbod),
     %        8,mem(82+j),lmem(82+j))
          else if (c80(1:2).eq.'ep'.or.c80(1:2).eq.'EP'.or.c80(1:2)
     %        .eq.'Ep') then
            epoch (nbod) = temp
          else if (c80(1:2).eq.'a1'.or.c80(1:2).eq.'A1') then
            ngf (1,nbod) = temp
          else if (c80(1:2).eq.'a2'.or.c80(1:2).eq.'A2') then
            ngf (2,nbod) = temp
          else if (c80(1:2).eq.'a3'.or.c80(1:2).eq.'A3') then
            ngf (3,nbod) = temp
          else if (c80(1:1).eq.'b'.or.c80(1:1).eq.'B') then
            ngf (4,nbod) = temp
          else
            goto 666
          end if
        end do
c
c If required, read Cartesian coordinates, velocities and spins of the bodies
        jtmp = 100
 135    read (11,'(a150)',end=666) string
        if (string(1:1).eq.')') goto 135
        backspace 11
        if (informat.eq.1) then
          read (11,*,err=666) x(1,nbod),x(2,nbod),x(3,nbod),
     %      v(1,nbod),v(2,nbod),v(3,nbod),s(1,nbod),s(2,nbod),s(3,nbod)
        else
          read (11,*,err=666) a,e,i,p,n,l,s(1,nbod),s(2,nbod),
     %      s(3,nbod)
          i = i * DR
          p = (p + n) * DR
          n = n * DR
          temp = m(nbod)  +  m(1)
c
c Alternatively, read Cometary or asteroidal elements
          if (informat.eq.3) then
            q = a
            a = q / (1.d0 - e)
            l = mod (sqrt(temp/(abs(a*a*a))) * (epoch(nbod) - l), TWOPI)
          else
            q = a * (1.d0 - e)
            l = l * DR
          end if
          if (algor.eq.11.and.nbod.ne.2) temp = temp + m(2)
          call mco_el2x (temp,q,e,i,p,n,l,x(1,nbod),x(2,nbod),x(3,nbod),
     %      v(1,nbod),v(2,nbod),v(3,nbod))
        end if
c
        s(1,nbod) = s(1,nbod) * K2
        s(2,nbod) = s(2,nbod) * K2
        s(3,nbod) = s(3,nbod) * K2
c
        goto 130
 140    close (11)
      end do
c
c Set non-gravitational-forces flag, NGFLAG
      ngflag = 0
      do j = 2, nbod
        if (ngf(1,j).ne.0.or.ngf(2,j).ne.0.or.ngf(3,j).ne.0) then
          if (ngflag.eq.0) ngflag = 1
          if (ngflag.eq.2) ngflag = 3
        else if (ngf(4,j).ne.0) then
          if (ngflag.eq.0) ngflag = 2
          if (ngflag.eq.1) ngflag = 3
        end if
```

```
      end do
c
c-----------------------------------------------------------------------------
c
c  IF  CONTINUING  AN  OLD  INTEGRATION
c
      if (oldflag) then
        if (opt(3).eq.1) then
          call mio_jd2y (time,year,month,t1)
          write (23,'(/,a,i10,i2,f8.5,/)') mem(62)(1:lmem(62)),year,
     %      month,t1
        else if (opt(3).eq.3) then
          t1 = (time - tstart) / 365.25d0
          write (23,'(/,a,f18.7,a,/)') mem(62)(1:lmem(62)),t1,
     %      mem(2)(1:lmem(2))
        else
          if (opt(3).eq.0) t1 = time
          if (opt(3).eq.2) t1 = time - tstart
          write (23,'(/,a,f18.5,a,/)') mem(62)(1:lmem(62)),t1,
     %      mem(1)(1:lmem(1))
        end if
c
c Read in energy and angular momentum variables, and convert to internal units
 330    open (35, file=dumpfile(4), status='old', err=330)
          read (35,*) opflag
          read (35,*) en(1),am(1),en(3),am(3)
          en(1) = en(1) * K2
          en(3) = en(3) * K2
          am(1) = am(1) * K2
          am(3) = am(3) * K2
          read (35,*) s(1,1),s(2,1),s(3,1)
          s(1,1) = s(1,1) * K2
          s(2,1) = s(2,1) * K2
          s(3,1) = s(3,1) * K2
        close (35)
        if (opflag.eq.0) opflag = 1
c
c-----------------------------------------------------------------------------
c
c  IF  STARTING  A  NEW  INTEGRATION
c
      else
        opflag = -2
c
c Write integration parameters to information file
        write (23,'(/,a)') mem(11)(1:lmem(11))
        write (23,'(a)') mem(12)(1:lmem(12))
        j = algor + 13
        write (23,'(/,2a)') mem(13)(1:lmem(13)),mem(j)(1:lmem(j))
        if (tstart.ge.1.d11.or.tstart.le.-1.d10) then
          write (23,'(/,a,1p,e19.13,a)') mem(26)(1:lmem(26)),tstart,
     %    mem(1)(1:lmem(1))
        else
          write (23,'(/,a,f19.7,a)') mem(26)(1:lmem(26)),tstart,
     %    mem(1)(1:lmem(1))
        end if
        if (tstop.ge.1.d11.or.tstop.le.-1.d10) then
          write (23,'(a,1p,e19.13)') mem(27)(1:lmem(27)),tstop
        else
          write (23,'(a,f19.7)') mem(27)(1:lmem(27)),tstop
        end if
        write (23,'(a,f15.3)') mem(28)(1:lmem(28)),dtout
        if (opt(4).eq.1) write (23,'(2a)') mem(40)(1:lmem(40)),
     %    mem(7)(1:lmem(7))
        if (opt(4).eq.2) write (23,'(2a)') mem(40)(1:lmem(40)),
     %    mem(8)(1:lmem(8))
        if (opt(4).eq.3) write (23,'(2a)') mem(40)(1:lmem(40)),
     %    mem(9)(1:lmem(9))
c
        write (23,'(/,a,f10.3,a)') mem(30)(1:lmem(30)),h0,
     %    mem(1)(1:lmem(1))
        write (23,'(a,1p1e10.4)') mem(31)(1:lmem(31)),tol
```

```
         write (23,'(a,1p1e10.4,a)') mem(32)(1:lmem(32)),m(1)/K2,
     %     mem(3)(1:lmem(3))
         write (23,'(a,1p1e11.4)') mem(33)(1:lmem(33)),jcen(1)/rcen**2
         write (23,'(a,1p1e11.4)') mem(34)(1:lmem(34)),jcen(2)/rcen**4
         write (23,'(a,1p1e11.4)') mem(35)(1:lmem(35)),jcen(3)/rcen**6
         write (23,'(a,1p1e10.4,a)') mem(36)(1:lmem(36)),rmax,
     %     mem (4)(1:lmem(4))
         write (23,'(a,1p1e10.4,a)') mem(37)(1:lmem(37)),rcen,
     %     mem (4)(1:lmem(4))
c
         itmp = 5
         if (opt(2).eq.1.or.opt(2).eq.2) itmp = 6
         write (23,'(/,2a)') mem(41)(1:lmem(41)),mem(itmp)(1:lmem(itmp))
         itmp = 5
         if (opt(2).eq.2) itmp = 6
         write (23,'(2a)') mem(42)(1:lmem(42)),mem(itmp)(1:lmem(itmp))
         itmp = 5
         if (opt(7).eq.1) itmp = 6
         write (23,'(2a)') mem(45)(1:lmem(45)),mem(itmp)(1:lmem(itmp))
         itmp = 5
         if (opt(8).eq.1) itmp = 6
         write (23,'(2a)') mem(46)(1:lmem(46)),mem(itmp)(1:lmem(itmp))
c
c Check that element and close-encounter files don't exist, and create them
         do j = 1, 2
           inquire (file=outfile(j), exist=test)
           if (test) call mio_err (23,mem(81),lmem(81),mem(87),lmem(87),
     %       ' ',1,outfile(j),80)
 430       open  (20+j, file=outfile(j), status='new', err=430)
           close (20+j)
         end do
c
c Check that dump files don't exist, and then create them
         do j = 1, 4
           inquire (file=dumpfile(j), exist=test)
           if (test) call mio_err (23,mem(81),lmem(81),mem(87),lmem(87),
     %       ' ',1,dumpfile(j),80)
 450       open  (30+j, file=dumpfile(j), status='new', err=450)
           close (30+j)
         end do
c
c Write number of Big bodies and Small bodies to information file
         write (23,'(/,a,i4)') mem(38)(1:lmem(38)), nbig - 1
         write (23,'(a,i4)') mem(39)(1:lmem(39)), nbod - nbig
c
c Calculate initial energy and angular momentum and write to information file
         s(1,1) = 0.d0
         s(2,1) = 0.d0
         s(3,1) = 0.d0
         call mxx_en (jcen,nbod,nbig,m,x,v,s,en(1),am(1))
         write (23,'(//,a)') mem(51)(1:lmem(51))
         write (23,'(a)')    mem(52)(1:lmem(52))
         write (23,'(/,a,1p1e12.5,a)') mem(53)(1:lmem(53)),en(1)/K2,
     %     mem(72)(1:lmem(72))
         write (23,'(a,1p1e12.5,a)')   mem(54)(1:lmem(54)),am(1)/K2,
     %     mem(73)(1:lmem(73))
c
c Initialize lost energy and angular momentum
         en(3) = 0.d0
         am(3) = 0.d0
c
c Write warning messages if necessary
         if (tstop.lt.tstart) write (23,'(/,2a)') mem(121)(1:lmem(121)),
     %     mem(123)(1:lmem(123))
         if (nbig.le.0) write (23,'(/,2a)') mem(121)(1:lmem(121)),
     %     mem(124)(1:lmem(124))
         if (nbig.eq.nbod) write (23,'(/,2a)') mem(121)(1:lmem(121)),
     %     mem(125)(1:lmem(125))
       end if
c
c-----------------------------------------------------------------------
c
```

```
c  CHECK  FOR  ATTEMPTS  TO  DO  INCOMPATIBLE  THINGS
c
c If using close-binary algorithm, set radius of central body to be no less
c than the periastron of binary star.
      if (algor.eq.11) then
        temp = m(1) + m(2)
        call mco_x2el (temp,x(1,2),x(2,2),x(3,2),v(1,2),v(2,2),v(3,2),
     %    a,tmp2,tmp3,tmp4,tmp5,tmp6)
        rcen = max (rcen, a)
      end if
c
c Check if non-grav forces are being used with an incompatible algorithm
      if (ngflag.ne.0.and.(algor.eq.3.or.algor.eq.11.or.algor.eq.12))
     %  call mio_err (23,mem(81),lmem(81),mem(92),lmem(92),' ',1,
     %  mem(85),lmem(85))
c
c Check if user-defined force routine is being used with wrong algorithm
      if (opt(8).eq.1.and.(algor.eq.11.or.algor.eq.12)) call mio_err
     %  (23,mem(81),lmem(81),mem(93),lmem(93),' ',1,mem(85),lmem(85))
c
c Check whether MVS is being used to integrate massive Small bodies,
c or whether massive Small bodies have different epochs than Big bodies.
      flag1 = .false.
      flag2 = .false.
      do j = nbig + 1, nbod
        if (m(j).ne.0) then
          if (algor.eq.1) call mio_err (23,mem(81),lmem(81),mem(94),
     %      lmem(94),' ',1,mem(85),lmem(85))
          flag1 = .true.
        end if
        if (epoch(j).ne.time) flag2 = .true.
      end do
      if (flag1.and.flag2) call mio_err (23,mem(81),lmem(81),mem(95),
     %    lmem(95),' ',1,mem(84),lmem(84))
c
c Check if central oblateness is being used with close-binary algorithm
      if (algor.eq.11.and.(jcen(1).ne.0.or.jcen(2).ne.0.or.jcen(3)
     %  .ne.0)) call mio_err (23,mem(81),lmem(81),mem(102),lmem(102),
     %  ' ',1,mem(85),lmem(85))
c
c Check whether RCEN > RMAX or RMAX/RCEN is very large
      if (rcen.gt.rmax) call mio_err (23,mem(81),lmem(81),mem(105),
     %  lmem(105),' ',1,mem(85),lmem(85))
      if (rmax/rcen.ge.1.d12) write (23,'(/,2a,/a)')
     %  mem(121)(1:lmem(121)),mem(106)(1:lmem(106)),mem(85)(1:lmem(85))
      close (23)
      return
c
c Error reading from the input file containing integration parameters
 661  write (c3,'(i3)') lineno
      call mio_err (23,mem(81),lmem(81),mem(99),lmem(99),c3,3,
     %  mem(85),lmem(85))
c
c Error reading from the input file for Big or Small bodies
 666  call mio_err (23,mem(81),lmem(81),mem(100),lmem(100),id(nbod),8,
     %  mem(82+j),lmem(82+j))
c
c Error reading epoch of Big bodies
 667  call mio_err (23,mem(81),lmem(81),mem(101),lmem(101),' ',1,
     %  mem(83),lmem(83))
c
c-----------------------------------------------------------------------
c
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_JD2Y.FOR    (ErikSoft  7 July 1999)
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_jd2y (jd0,year,month,day)
c
      implicit none
c
c Input/Output
      integer year,month
      real*8 jd0,day
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 doesn'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_LOG.FOR    (ErikSoft   25 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Writes a progress report to the log file (or the screen if you are running
c Mercury interactively).
c
c-----------------------------------------------------------------------
c
      subroutine mio_log (time,tstart,en,am,opt,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer lmem(NMESS), opt(8)
      real*8 time, tstart, en(3), am(3)
      character*80 mem(NMESS)
c
c Local
      integer year, month
      real*8 tmp0, tmp1, t1
      character*38 flog
      character*6 tstring
c
c-----------------------------------------------------------------------
c
      if (opt(3).eq.0.or.opt(3).eq.2) then
        tstring = mem(1)
        flog = '(1x,a,f14.1,a,2(a,1p1e12.5))'
      else if (opt(3).eq.1) then
        flog = '(1x,a,i10,1x,i2,1x,f4.1,2(a,1p1e12.5))'
      else
        tstring = mem(2)
        flog = '(1x,a,f14.3,a,2(a,1p1e12.5))'
      end if
c
      tmp0 = 0.d0
      tmp1 = 0.d0
      if (en(1).ne.0) tmp0 = (en(2) + en(3) - en(1)) / abs(en(1))
      if (am(1).ne.0) tmp1 = (am(2) + am(3) - am(1)) / abs(am(1))
c
      if (opt(3).eq.1) then
        call mio_jd2y (time,year,month,t1)
```

```
         write (*,flog) mem(64)(1:lmem(64)), year, month, t1,
     %     mem(65)(1:lmem(65)), tmp0,mem(66)(1:lmem(66)), tmp1
        else
         if (opt(3).eq.0) t1 = time
         if (opt(3).eq.2) t1 = time - tstart
         if (opt(3).eq.3) t1 = (time - tstart) / 365.25d0
         write (*,flog) mem(63)(1:lmem(63)), t1, tstring,
     %     mem(65)(1:lmem(65)), tmp0, mem(66)(1:lmem(66)), tmp1
        end if
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_OUT.FOR    (ErikSoft   13 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Writes output variables for each object to an output file. Each variable
c is scaled between the minimum and maximum possible values and then
c written in a compressed format using ASCII characters.
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 If this is the first output (OPFLAG = -1), or the first output since the
c number of the objects or their masses have changed (OPFLAG = 1), then
c the names, masses and spin components of all the objects are also output.
c
c N.B. Each object's distance must lie between RCEN < R < RMAX
c ===
c
c-----------------------------------------------------------------------
c
      subroutine mio_out (time,jcen,rcen,rmax,nbod,nbig,m,xh,vh,s,rho,
     %  stat,id,opt,opflag,algor,outfile)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig, stat(nbod), opt(8), opflag, algor
      real*8 time,jcen(3),rcen,rmax,m(nbod),xh(3,nbod),vh(3,nbod)
      real*8 s(3,nbod),rho(nbod)
      character*80 outfile
      character*8 id(nbod)
c
c Local
      integer k, len, nchar
      real*8 rhocgs,k_2,rfac,rcen_2,fr,fv,theta,phi,vtheta,vphi
      character*80 header,c(NMAX)
      character*8 mio_fl2c,mio_re2c
      character*5 fout
c
c-----------------------------------------------------------------------
c
      rhocgs = AU * AU * AU * K2 / MSUN
      k_2 = 1.d0 / K2
      rcen_2 = 1.d0 / (rcen * rcen)
c
c Scaling factor (maximum possible range) for distances
      rfac = log10 (rmax / rcen)
```

```
c
c Create the format list, FOUT, used when outputting the orbital elements
      if (opt(4).eq.1) nchar = 2
      if (opt(4).eq.2) nchar = 4
      if (opt(4).eq.3) nchar = 7
      len = 3  +  6 * nchar
      fout(1:5) = '(a  )'
      if (len.lt.10) write (fout(3:3),'(i1)') len
      if (len.ge.10) write (fout(3:4),'(i2)') len
c
c Open the orbital-elements output file
  10  open (21, file=outfile, status='old', access='append', err=10)
c
c-----------------------------------------------------------------------------
c
c  SPECIAL  OUTPUT  PROCEDURE
c
c If this is a new integration or a complete output is required (e.g. because
c the number of objects has changed), then output object details & parameters.
      if (opflag.eq.-1.or.opflag.eq.1) then
c
c Compose a header line with time, number of objects and relevant parameters
        header(1:8)   = mio_fl2c (time)
        header(9:16)  = mio_re2c (dble(nbig - 1),   0.d0, 11239423.99d0)
        header(12:19) = mio_re2c (dble(nbod - nbig),0.d0, 11239423.99d0)
        header(15:22) = mio_fl2c (m(1) * k_2)
        header(23:30) = mio_fl2c (jcen(1) * rcen_2)
        header(31:38) = mio_fl2c (jcen(2) * rcen_2 * rcen_2)
        header(39:46) = mio_fl2c (jcen(3) * rcen_2 * rcen_2 * rcen_2)
        header(47:54) = mio_fl2c (rcen)
        header(55:62) = mio_fl2c (rmax)
c
c For each object, compress its index number, name, mass, spin components
c and density (some of these need to be converted to normal units).
        do k = 2, nbod
          c(k)(1:8) = mio_re2c (dble(k - 1), 0.d0, 11239423.99d0)
          c(k)(4:11) = id(k)
          c(k)(12:19) = mio_fl2c (m(k) * k_2)
          c(k)(20:27) = mio_fl2c (s(1,k) * k_2)
          c(k)(28:35) = mio_fl2c (s(2,k) * k_2)
          c(k)(36:43) = mio_fl2c (s(3,k) * k_2)
          c(k)(44:51) = mio_fl2c (rho(k) / rhocgs)
        end do
c
c Write compressed output to file
        write (21,'(a1,a2,i2,a62,i1)') char(12),'6a',algor,header(1:62),
     %    opt(4)
        do k = 2, nbod
          write (21,'(a51)') c(k)(1:51)
        end do
      end if
c
c-----------------------------------------------------------------------------
c
c  NORMAL  OUTPUT  PROCEDURE
c
c Compose a header line containing the time and number of objects
      header(1:8)   = mio_fl2c (time)
      header(9:16)  = mio_re2c (dble(nbig - 1),    0.d0, 11239423.99d0)
      header(12:19) = mio_re2c (dble(nbod - nbig), 0.d0, 11239423.99d0)
c
c Calculate output variables for each body and convert to compressed format
      do k = 2, nbod
        call mco_x2ov (rcen,rmax,m(1),m(k),xh(1,k),xh(2,k),xh(3,k),
     %    vh(1,k),vh(2,k),vh(3,k),fr,theta,phi,fv,vtheta,vphi)
c
c Object's index number and output variables
        c(k)(1:8) = mio_re2c (dble(k - 1), 0.d0, 11239423.99d0)
        c(k)(4:11)                 = mio_re2c (fr,     0.d0, rfac)
        c(k)(4+  nchar:11+  nchar) = mio_re2c (theta,  0.d0, PI)
        c(k)(4+2*nchar:11+2*nchar) = mio_re2c (phi,    0.d0, TWOPI)
        c(k)(4+3*nchar:11+3*nchar) = mio_re2c (fv,     0.d0, 1.d0)
```

```
        c(k)(4+4*nchar:11+4*nchar) = mio_re2c (vtheta, 0.d0, PI)
        c(k)(4+5*nchar:11+5*nchar) = mio_re2c (vphi,   0.d0, TWOPI)
      end do
c
c Write compressed output to file
      write (21,'(a1,a2,a14)') char(12),'6b',header(1:14)
      do k = 2, nbod
        write (21,fout) c(k)(1:len)
      end do
c
      close (21)
      opflag = 0
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MIO_RE2C.FOR    (ErikSoft  27 June 1999)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Converts a REAL*8 variable X, where XMIN <= X < XMAX, into an ASCII string
c of 8 characters, using the new format compression:
c
c X is first converted to base 224, and then each base 224 digit is converted
c to an ASCII character, such that 0 -> character 32, 1 -> character 33...
c and 223 -> character 255.
c
c ASCII characters 0 - 31 (CTRL characters) are not used, because they
c cause problems when using some operating systems.
c
c-----------------------------------------------------------------------
c
      function mio_re2c (x,xmin,xmax)
c
      implicit none
c
c Input/output
      real*8 x,xmin,xmax
      character*8 mio_re2c
c
c Local
      integer j
      real*8 y,z
c
c-----------------------------------------------------------------------
c
      mio_re2c(1:8) = '        '
      y = (x - xmin) / (xmax - xmin)
c
      if (y.ge.1) then
        do j = 1, 8
          mio_re2c(j:j) = char(255)
        end do
      else if (y.gt.0) then
        z = y
        do j = 1, 8
          z = mod(z, 1.d0) * 224.d0
          mio_re2c(j:j) = char(int(z) + 32)
        end do
      end if
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_EJEC.FOR    (ErikSoft   2 November 2000)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
```

```
c
c Calculates the distance from the central body of each object with index
c I >= I0. If this distance exceeds RMAX, the object is flagged for ejection
c (STAT set to -3). If any object is to be ejected, EJFLAG = 1 on exit,
c otherwise EJFLAG = 0.
c
c Also updates the values of EN(3) and AM(3)---the change in energy and
c angular momentum due to collisions and ejections.
c
c
c N.B. All coordinates must be with respect to the central body!!
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mxx_ejec (time,tstart,rmax,en,am,jcen,i0,nbod,nbig,m,x,
     %  v,s,stat,id,opt,ejflag,outfile,mem,lmem)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer i0, nbod, nbig, stat(nbod), opt(8), ejflag, lmem(NMESS)
      real*8 time, tstart, rmax, en(3), am(3), jcen(3)
      real*8 m(nbod), x(3,nbod), v(3,nbod), s(3,nbod)
      character*80 outfile, mem(NMESS)
      character*8 id(nbod)
c
c Local
      integer j, year, month
      real*8 r2,rmax2,t1,e,l
      character*38 flost
      character*6 tstring
c
c-----------------------------------------------------------------------------
c
      if (i0.le.0) i0 = 2
      ejflag = 0
      rmax2 = rmax * rmax
c
c Calculate initial energy and angular momentum
      call mxx_en (jcen,nbod,nbig,m,x,v,s,e,l)
c
c Flag each object which is ejected, and set its mass to zero
      do j = i0, nbod
        r2 = x(1,j)*x(1,j) + x(2,j)*x(2,j) + x(3,j)*x(3,j)
        if (r2.gt.rmax2) then
          ejflag = 1
          stat(j) = -3
          m(j) = 0.d0
          s(1,j) = 0.d0
          s(2,j) = 0.d0
          s(3,j) = 0.d0
c
c Write message to information file
  20      open  (23,file=outfile,status='old',access='append',err=20)
          if (opt(3).eq.1) then
            call mio_jd2y (time,year,month,t1)
            flost = '(1x,a8,a,i10,1x,i2,1x,f8.5)'
            write (23,flost) id(j),mem(68)(1:lmem(68)),year,month,t1
          else
            if (opt(3).eq.3) then
              t1 = (time - tstart) / 365.25d0
              tstring = mem(2)
              flost = '(1x,a8,a,f18.7,a)'
            else
              if (opt(3).eq.0) t1 = time
              if (opt(3).eq.2) t1 = time - tstart
              tstring = mem(1)
              flost = '(1x,a8,a,f18.5,a)'
            end if
            write (23,flost) id(j),mem(68)(1:lmem(68)),t1,tstring
```

```
          end if
          close (23)
        end if
      end do
c
c If ejections occurred, update ELOST and LLOST
      if (ejflag.ne.0) then
        call mxx_en (jcen,nbod,nbig,m,x,v,s,en(2),am(2))
        en(3) = en(3) + (e - en(2))
        am(3) = am(3) + (l - am(2))
      end if
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c      MXX_ELIM.FOR    (ErikSoft   13 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Removes any objects with STAT < 0 (i.e. those that have been flagged for
c removal) and reindexes all the appropriate arrays for the remaining objects.
c
c-----------------------------------------------------------------------------
c
      subroutine mxx_elim (nbod,nbig,m,x,v,s,rho,rceh,rcrit,ngf,stat,
     %  id,mem,lmem,outfile,nelim)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod, nbig, nelim, stat(nbod), lmem(NMESS)
      real*8 m(nbod), x(3,nbod), v(3,nbod), s(3,nbod)
      real*8 rho(nbod), rceh(nbod), rcrit(nbod), ngf(4,nbod)
      character*8 id(nbod)
      character*80 outfile, mem(NMESS)
c
c Local
      integer j, k, l, nbigelim, elim(NMAX+1)
c
c-----------------------------------------------------------------------------
c
c Find out how many objects are to be removed
      nelim = 0
      nbigelim = 0
      do j = 2, nbod
        if (stat(j).lt.0) then
          nelim = nelim + 1
          elim(nelim) = j
          if (j.le.nbig) nbigelim = nbigelim + 1
        end if
      end do
      elim(nelim+1) = nbod + 1
c
c Eliminate unwanted objects
      do k = 1, nelim
        do j = elim(k) - k + 1, elim(k+1) - k - 1
          l = j + k
          x(1,j) = x(1,l)
          x(2,j) = x(2,l)
          x(3,j) = x(3,l)
          v(1,j) = v(1,l)
          v(2,j) = v(2,l)
          v(3,j) = v(3,l)
          m(j)   = m(l)
          s(1,j) = s(1,l)
```

```
            s(2,j) = s(2,l)
            s(3,j) = s(3,l)
            rho(j) = rho(l)
            rceh(j) = rceh(l)
            stat(j) = stat(l)
            id(j) = id(l)
            ngf(1,j) = ngf(1,l)
            ngf(2,j) = ngf(2,l)
            ngf(3,j) = ngf(3,l)
            ngf(4,j) = ngf(4,l)
          end do
        end do
c
c Update total number of bodies and number of Big bodies
        nbod = nbod - nelim
        nbig = nbig - nbigelim
c
c If no massive bodies remain, stop the integration
        if (nbig.lt.1) then
  10      open (23,file=outfile,status='old',access='append',err=10)
          write (23,'(2a)') mem(81)(1:lmem(81)),mem(124)(1:lmem(124))
          close (23)
          stop
        end if
c
c-----------------------------------------------------------------------
c
        return
        end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c     MXX_EN.FOR    (ErikSoft   21 February 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates the total energy and angular-momentum for a system of objects
c with masses M, coordinates X, velocities V and spin angular momenta S.
c
c N.B. All coordinates and velocities must be with respect to the central
c ===  body.
c
c-----------------------------------------------------------------------
c
        subroutine mxx_en  (jcen,nbod,nbig,m,xh,vh,s,e,l2)
c
        implicit none
        include 'mercury.inc'
c
c Input/Output
        integer nbod,nbig
        real*8 jcen(3),m(nbod),xh(3,nbod),vh(3,nbod),s(3,nbod),e,l2
c
c Local
        integer j,k,iflag,itmp(8)
        real*8 x(3,NMAX),v(3,NMAX),temp,dx,dy,dz,r2,tmp,ke,pe,l(3)
        real*8 r_1,r_2,r_4,r_6,u2,u4,u6,tmp2(4,NMAX)
c
c-----------------------------------------------------------------------
c
        ke = 0.d0
        pe = 0.d0
        l(1) = 0.d0
        l(2) = 0.d0
        l(3) = 0.d0
c
c Convert to barycentric coordinates and velocities
        call mco_h2b(temp,jcen,nbod,nbig,temp,m,xh,vh,x,v,tmp2,iflag,itmp)
c
c Do the spin angular momenta first (probably the smallest terms)
```

```
      do j = 1, nbod
        l(1) = l(1) + s(1,j)
        l(2) = l(2) + s(2,j)
        l(3) = l(3) + s(3,j)
      end do
c
c Orbital angular momentum and kinetic energy terms
      do j = 1, nbod
        l(1) = l(1)  +  m(j)*(x(2,j) * v(3,j)  -  x(3,j) * v(2,j))
        l(2) = l(2)  +  m(j)*(x(3,j) * v(1,j)  -  x(1,j) * v(3,j))
        l(3) = l(3)  +  m(j)*(x(1,j) * v(2,j)  -  x(2,j) * v(1,j))
        ke = ke + m(j)*(v(1,j)*v(1,j)+v(2,j)*v(2,j)+v(3,j)*v(3,j))
      end do
c
c Potential energy terms due to pairs of bodies
      do j = 2, nbod
        tmp = 0.d0
        do k = j + 1, nbod
          dx = x(1,k) - x(1,j)
          dy = x(2,k) - x(2,j)
          dz = x(3,k) - x(3,j)
          r2 = dx*dx + dy*dy + dz*dz
          if (r2.ne.0) tmp = tmp + m(k) / sqrt(r2)
        end do
        pe = pe  -  tmp * m(j)
      end do
c
c Potential energy terms involving the central body
      do j = 2, nbod
        dx = x(1,j) - x(1,1)
        dy = x(2,j) - x(2,1)
        dz = x(3,j) - x(3,1)
        r2 = dx*dx + dy*dy + dz*dz
        if (r2.ne.0) pe = pe  -  m(1) * m(j) / sqrt(r2)
      end do
c
c Corrections for oblateness
      if (jcen(1).ne.0.or.jcen(2).ne.0.or.jcen(3).ne.0) then
        do j = 2, nbod
          r2 = xh(1,j)*xh(1,j) + xh(2,j)*xh(2,j) + xh(3,j)*xh(3,j)
          r_1 = 1.d0 / sqrt(r2)
          r_2 = r_1 * r_1
          r_4 = r_2 * r_2
          r_6 = r_4 * r_2
          u2 = xh(3,j) * xh(3,j) * r_2
          u4 = u2 * u2
          u6 = u4 * u2
          pe = pe + m(1) * m(j) * r_1
     %       * (jcen(1) * r_2 * (1.5d0*u2 - 0.5d0)
     %       +  jcen(2) * r_4 * (4.375d0*u4 - 3.75d0*u2 + .375d0)
     %       +  jcen(3) * r_6
     %       *(14.4375d0*u6 - 19.6875d0*u4 + 6.5625d0*u2 - .3125d0))
        end do
      end if
c
      e = .5d0 * ke  +  pe
      l2 = sqrt(l(1)*l(1) + l(2)*l(2) + l(3)*l(3))
c
c-----------------------------------------------------------------------
c
      return
      end
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c     MXX_JAC.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Calculates the Jacobi constant for massless particles. This assumes that
```

```
c there are only 2 massive bodies (including the central body) moving on
c circular orbits.
c
c N.B. All coordinates and velocities must be heliocentric!!
c ===
c
c-----------------------------------------------------------------------------
c
      subroutine mxx_jac (jcen,nbod,nbig,m,xh,vh,jac)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig
      real*8 jcen(3),m(nbod),xh(3,nbod),vh(3,nbod)
c
c Local
      integer j,itmp(8),iflag
      real*8 x(3,NMAX),v(3,NMAX),temp,dx,dy,dz,r,d,a2,n,jac(NMAX)
      real*8 tmp2(4,NMAX)
c
c-----------------------------------------------------------------------------
c
      call mco_h2b(temp,jcen,nbod,nbig,temp,m,xh,vh,x,v,tmp2,iflag,itmp)
      dx = x(1,2) - x(1,1)
      dy = x(2,2) - x(2,1)
      dz = x(3,2) - x(3,1)
      a2 = dx*dx + dy*dy + dz*dz
      n = sqrt((m(1)+m(2)) / (a2*sqrt(a2)))
c
      do j = nbig + 1, nbod
        dx = x(1,j) - x(1,1)
        dy = x(2,j) - x(2,1)
        dz = x(3,j) - x(3,1)
        r = sqrt(dx*dx + dy*dy + dz*dz)
        dx = x(1,j) - x(1,2)
        dy = x(2,j) - x(2,2)
        dz = x(3,j) - x(3,2)
        d = sqrt(dx*dx + dy*dy + dz*dz)
c
        jac(j) = .5d0*(v(1,j)*v(1,j) + v(2,j)*v(2,j) + v(3,j)*v(3,j))
     %         - m(1)/r - m(2)/d - n*(x(1,j)*v(2,j) - x(2,j)*v(1,j))
      end do
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      MXX_SYNC.FOR    (ErikSoft   2 March 2001)
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
c Author: John E. Chambers
c
c Synchronizes the epochs of NBIG Big bodies (having a common epoch) and
c NBOD-NBIG Small bodies (possibly having differing epochs), for an
c integration using MERCURY.
c The Small bodies are picked up in order starting with the one with epoch
c furthest from the time, TSTART, at which the main integration will begin
c producing output.
c
c N.B. The synchronization integrations use Everhart's RA15 routine.
c ---
c
c-----------------------------------------------------------------------------
c
      subroutine mxx_sync (time,tstart,h0,tol,jcen,nbod,nbig,m,x,v,s,
     %  rho,rceh,stat,id,epoch,ngf,opt,ngflag)
c
      implicit none
      include 'mercury.inc'
c
c Input/Output
      integer nbod,nbig,ngflag,opt(8),stat(nbod)
      real*8 time,tstart,h0,tol,jcen(3),m(nbod),x(3,nbod),v(3,nbod)
      real*8 s(3,nbod),rceh(nbod),rho(nbod),epoch(nbod),ngf(4,nbod)
      character*8 id(nbod)
c
```

```
c Local
      integer j,k,l,nsml,nsofar,indx(NMAX),itemp,jtemp(NMAX)
      integer raflag,nce,ice(NMAX),jce(NMAX)
      real*8 temp,epsml(NMAX),rtemp(NMAX)
      real*8 h,hdid,tsmall,rphys(NMAX),rcrit(NMAX)
      character*8 ctemp(NMAX)
      external mfo_all
c
c-----------------------------------------------------------------------------
c
c Reorder Small bodies by epoch so that ep(1) is furthest from TSTART
      nsml = nbod - nbig
      do j = nbig + 1, nbod
        epsml(j-nbig) = epoch(j)
      end do
      call mxx_sort (nsml,epsml,indx)
c
      if (abs(epsml(1)-tstart).lt.abs(epsml(nsml)-tstart)) then
        k = nsml + 1
        do j = 1, nsml / 2
          l = k - j
          temp = epsml(j)
          epsml(j) = epsml (l)
          epsml(l) = temp
          itemp = indx(j)
          indx(j) = indx (l)
          indx(l) = itemp
        end do
      end if
c
      do j = nbig + 1, nbod
        epoch(j) = epsml(j-nbig)
      end do
c
c Reorder the other arrays associated with each Small body
      do k = 1, 3
        do j = 1, nsml
          rtemp(j) = x(k,j+nbig)
        end do
        do j = 1, nsml
          x(k,j+nbig) = rtemp(indx(j))
        end do
        do j = 1, nsml
          rtemp(j) = v(k,j+nbig)
        end do
        do j = 1, nsml
          v(k,j+nbig) = rtemp(indx(j))
        end do
        do j = 1, nsml
          rtemp(j) = s(k,j+nbig)
        end do
        do j = 1, nsml
          s(k,j+nbig) = rtemp(indx(j))
        end do
      end do
c
      do j = 1, nsml
        rtemp(j) = m(j+nbig)
      end do
      do j = 1, nsml
        m(j+nbig) = rtemp(indx(j))
      end do
      do j = 1, nsml
        rtemp(j) = rceh(j+nbig)
      end do
      do j = 1, nsml
        rceh(j+nbig) = rtemp(indx(j))
      end do
      do j = 1, nsml
        rtemp(j) = rho(j+nbig)
      end do
      do j = 1, nsml
```

```
        rho(j+nbig) = rtemp(indx(j))
      end do
c
      do j = 1, nsml
        ctemp(j) = id(j+nbig)
        jtemp(j) = stat(j+nbig)
      end do
      do j = 1, nsml
        id(j+nbig) = ctemp(indx(j))
        stat(j+nbig) = jtemp(indx(j))
      end do
c
c Integrate Small bodies up to the same epoch
      h = h0
      tsmall = h0 * 1.d-12
      raflag = 0
c
      do j = nbig + 1, nbod
        nsofar = j - 1
        do while (abs(time-epoch(j)).gt.tsmall)
          temp = epoch(j) - time
          h = sign(max(min(abs(temp),abs(h)),tsmall),temp)
          call mdt_ra15 (time,h,hdid,tol,jcen,nsofar,nbig,m,x,v,s,rphys,
     %       rcrit,ngf,stat,raflag,ngflag,opt,nce,ice,jce,mfo_all)
          time = time + hdid
        end do
        raflag = 1
      end do
c
c-----------------------------------------------------------------------------
c
      return
      end
c
c*************************************************************************
c                        DRIFT_DAN.F
c*************************************************************************
c This subroutine does the Danby and decides which vbles to use
c
c             Input:
c                 nbod          ==>  number of massive bodies (int scalar)
c                 mass          ==>  mass of bodies (real array)
c                 x0,y0,z0         ==>  initial position in jacobi coord
c                                       (real scalar)
c                 vx0,vy0,vz0      ==>  initial position in jacobi coord
c                                       (real scalar)
c                 dt0            ==>  time step
c             Output:
c                 x0,y0,z0         ==>  final position in jacobi coord
c                                       (real scalars)
c                 vx0,vy0,vz0      ==>  final position in jacobi coord
c                                       (real scalars)
c                 iflg             ==>  integer flag (zero if satisfactory)
c                                            (non-zero if nonconvergence)
c
c Authors:  Hal Levison & Martin Duncan
c Date:    2/10/93
c Last revision: April 6/93 - MD adds dt and keeps dt0 unchanged

      subroutine drift_dan(mu,x0,y0,z0,vx0,vy0,vz0,dt0,iflg)

      include 'swift.inc'

c...  Inputs Only:
      real*8 mu,dt0

c...  Inputs and Outputs:
      real*8 x0,y0,z0
      real*8 vx0,vy0,vz0

c...  Output
      integer iflg
```

```
c...  Internals:
      real*8 x,y,z,vx,vy,vz,dt
      real*8 f,g,fdot,c1,c2
      real*8 c3,gdot
      real*8 u,alpha,fp,r0,v0s
      real*8 a,asq,en
      real*8 dm,ec,es,esq,xkep
      real*8 fchk,s,c

c----
c...  Executable code

c...  Set dt = dt0 to be sure timestep is not altered while solving
c...  for new coords.
        dt = dt0
        iflg = 0
        r0 = sqrt(x0*x0 + y0*y0 + z0*z0)
        v0s = vx0*vx0 + vy0*vy0 + vz0*vz0
        u = x0*vx0 + y0*vy0 + z0*vz0
        alpha = 2.0*mu/r0 - v0s

        if (alpha.gt.0.d0) then
           a = mu/alpha
           asq = a*a
           en = sqrt(mu/(a*asq))
           ec = 1.0d0 - r0/a
           es = u/(en*asq)
           esq = ec*ec + es*es
           dm = dt*en - int(dt*en/TWOPI)*TWOPI
           dt = dm/en
           if((dm*dm .gt. 0.16d0) .or. (esq.gt.0.36d0)) goto 100

           if(esq*dm*dm .lt. 0.0016) then

               call drift_kepmd(dm,es,ec,xkep,s,c)
               fchk = (xkep - ec*s +es*(1.-c) - dm)

               if(fchk*fchk .gt. DANBYB) then
                  iflg = 1
                  return
               endif

               fp = 1. - ec*c + es*s
               f = (a/r0) * (c-1.) + 1.
               g = dt + (s-xkep)/en
               fdot = - (a/(r0*fp))*en*s
               gdot = (c-1.)/fp + 1.

               x = x0*f + vx0*g
               y = y0*f + vy0*g
               z = z0*f + vz0*g
               vx = x0*fdot + vx0*gdot
               vy = y0*fdot + vy0*gdot
               vz = z0*fdot + vz0*gdot

               x0 = x
               y0 = y
               z0 = z
               vx0 = vx
               vy0 = vy
               vz0 = vz

               iflg = 0
               return

             endif

           endif

100        call drift_kepu(dt,r0,mu,alpha,u,fp,c1,c2,c3,iflg)
```

```
         if(iflg .eq.0) then
           f = 1.0 - (mu/r0)*c2
           g = dt - mu*c3
           fdot = -(mu/(fp*r0))*c1
           gdot = 1. - (mu/fp)*c2

           x = x0*f + vx0*g
           y = y0*f + vy0*g
           z = z0*f + vz0*g
           vx = x0*fdot + vx0*gdot
           vy = y0*fdot + vy0*gdot
           vz = z0*fdot + vz0*gdot

           x0 = x
           y0 = y
           z0 = z
           vx0 = vx
           vy0 = vy
           vz0 = vz
         endif

         return
         end   ! drift_dan
c
c*************************************************************************#
c                   DRIFT_KEPMD
c*************************************************************************#
c  Subroutine for solving kepler's equation in difference form for an
c  ellipse, given SMALL dm and SMALL eccentricity.  See DRIFT_DAN.F
c  for the criteria.
c  WARNING - BUILT FOR SPEED : DOES NOT CHECK HOW WELL THE ORIGINAL
c  EQUATION IS SOLVED! (CAN DO THAT IN THE CALLING ROUTINE BY
c  CHECKING HOW CLOSE (x - ec*s +es*(1.-c) - dm) IS TO ZERO.
c
c       Input:
c           dm          ==> increment in mean anomaly M (real*8 scalar)
c           es,ec       ==> ecc. times sin and cos of E_0 (real*8 scalars)
c
c       Output:
c            x          ==> solution to Kepler's difference eqn (real*8 scalar)
c            s,c        ==> sin and cosine of x (real*8 scalars)
c

         subroutine drift_kepmd(dm,es,ec,x,s,c)

         implicit none

c...     Inputs
         real*8 dm,es,ec

c...     Outputs
         real*8 x,s,c

c...     Internals
         real*8 A0, A1, A2, A3, A4
         parameter(A0 = 39916800.d0, A1 = 6652800.d0, A2 = 332640.d0)
         parameter(A3 = 7920.d0, A4 = 110.d0)
         real*8 dx
         real*8 fac1,fac2,q,y
         real*8 f,fp,fpp,fppp


c...     calc initial guess for root
         fac1 = 1.d0/(1.d0 - ec)
         q = fac1*dm
         fac2 = es*es*fac1 - ec/3.d0
         x = q*(1.d0 -0.5d0*fac1*q*(es -q*fac2))

c...   excellent approx. to sin and cos of x for small x.
         y = x*x
         s = x*(A0-y*(A1-y*(A2-y*(A3-y*(A4-y)))))/A0
         c = sqrt(1.d0 - s*s)
```

```
c...    Compute better value for the root using quartic Newton method
        f = x - ec*s + es*(1.-c) - dm
        fp = 1. - ec*c + es*s
        fpp = ec*s + es*c
        fppp = ec*c - es*s
        dx = -f/fp
        dx = -f/(fp + 0.5*dx*fpp)
        dx = -f/(fp + 0.5*dx*fpp + 0.16666666666666666*dx*dx*fppp)
        x = x + dx

c...  excellent approx. to sin and cos of x for small x.
        y = x*x
        s = x*(A0-y*(A1-y*(A2-y*(A3-y*(A4-y)))))/A0
        c = sqrt(1.d0 - s*s)

        return
        end
c-----------------------------------------------------------------------------
c
c*****************************************************************
c                        DRIFT_KEPU.F
c*****************************************************************
c subroutine for solving kepler's equation using universal variables.
c
c             Input:
c                 dt            ==>  time step (real scalor)
c                 r0            ==>  Distance between 'Sun' and paritcle
c                                    (real scalor)
c                 mu            ==>  Reduced mass of system (real scalor)
c                 alpha         ==>  energy (real scalor)
c                 u             ==>  angular momentun  (real scalor)
c             Output:
c                 fp            ==>  f' from p170
c                                    (real scalors)
c                 c1,c2,c3      ==>  c's from p171-172
c                                    (real scalors)
c                 iflg          ==>  =0 if converged; !=0 if not
c
c Author:  Hal Levison
c Date:    2/3/93
c Last revision: 2/3/93

      subroutine drift_kepu(dt,r0,mu,alpha,u,fp,c1,c2,c3,iflg)

      include 'swift.inc'

c...  Inputs:
      real*8 dt,r0,mu,alpha,u

c...  Outputs:
      real*8 fp,c1,c2,c3
      integer iflg

c...  Internals:
      real*8 s,st,fo,fn

c----
c...  Executable code

        call drift_kepu_guess(dt,r0,mu,alpha,u,s)

        st = s
c..     store initial guess for possible use later in
c..     laguerre's method, in case newton's method fails.

        call drift_kepu_new(s,dt,r0,mu,alpha,u,fp,c1,c2,c3,iflg)
        if(iflg.ne.0) then
           call drift_kepu_fchk(dt,r0,mu,alpha,u,st,fo)
           call drift_kepu_fchk(dt,r0,mu,alpha,u,s,fn)
           if(abs(fo).lt.abs(fn)) then
               s = st
```

```
            endif
            call drift_kepu_lag(s,dt,r0,mu,alpha,u,fp,c1,c2,c3,iflg)
         endif

         return
         end    ! drift_kepu
c-----------------------------------------------------------------------
c
c*************************************************************************
c                           DRIFT_KEPU_FCHK.F
c*************************************************************************
c Returns the value of the function f of which we are trying to find the root
c in universal variables.
c
c             Input:
c                 dt            ==>  time step (real scalar)
c                 r0            ==>  Distance between 'Sun' and particle
c                                    (real scalar)
c                 mu            ==>  Reduced mass of system (real scalar)
c                 alpha         ==>  Twice the binding energy (real scalar)
c                 u             ==>  Vel. dot radial vector (real scalar)
c                 s             ==>  Approx. root of f
c             Output:
c                 f             ==>  function value ( = 0 if O.K.) (integer)
c
c Author:  Martin Duncan
c Date:    March 12/93
c Last revision: March 12/93

      subroutine drift_kepu_fchk(dt,r0,mu,alpha,u,s,f)

c...  Inputs:
      real*8 dt,r0,mu,alpha,u,s

c...  Outputs:
      real*8 f

c...  Internals:
      real*8  x,c0,c1,c2,c3

c----
c...  Executable code

        x=s*s*alpha
        call drift_kepu_stumpff(x,c0,c1,c2,c3)
        c1=c1*s
        c2 = c2*s*s
        c3 = c3*s*s*s
        f = r0*c1 + u*c2 + mu*c3 - dt

        return
        end     !   drift_kepu_fchk
c-----------------------------------------------------------------------
c
c*************************************************************************
c                         DRIFT_KEPU_GUESS.F
c*************************************************************************
c Initial guess for solving kepler's equation using universal variables.
c
c             Input:
c                 dt            ==>  time step (real scalor)
c                 r0            ==>  Distance between 'Sun' and paritcle
c                                    (real scalor)
c                 mu            ==>  Reduced mass of system (real scalor)
c                 alpha         ==>  energy (real scalor)
c                 u             ==>  angular momentun  (real scalor)
c             Output:
c                 s             ==>  initial guess for the value of
c                                    universal variable
c
c Author:  Hal Levison & Martin Duncan
c Date:    3/12/93
```

```
c Last revision: April 6/93
c Modified by JEC: 8/6/98

      subroutine drift_kepu_guess(dt,r0,mu,alpha,u,s)

      include 'swift.inc'

c...  Inputs:
      real*8 dt,r0,mu,alpha,u

c...  Inputs and Outputs:
      real*8 s

c...  Internals:
      integer iflg
      real*8 y,sy,cy,sigma,es
      real*8 x,a
      real*8 en,ec,e

c----
c...  Executable code

        if (alpha.gt.0.0) then
c...       find initial guess for elliptic motion

             if( dt/r0 .le. 0.4)  then
               s = dt/r0 - (dt*dt*u)/(2.0*r0*r0*r0)
               return
             else
               a = mu/alpha
               en = sqrt(mu/(a*a*a))
               ec = 1.0 - r0/a
               es = u/(en*a*a)
               e = sqrt(ec*ec + es*es)
               y = en*dt - es
c
               call mco_sine (y,sy,cy)
c
               sigma = dsign(1.d0,(es*cy + ec*sy))
               x = y + sigma*.85*e
               s = x/sqrt(alpha)
             endif

        else
c...       find initial guess for hyperbolic motion.
           call drift_kepu_p3solve(dt,r0,mu,alpha,u,s,iflg)
           if(iflg.ne.0) then
              s = dt/r0
           endif
        endif

        return
        end     !   drift_kepu_guess
c-----------------------------------------------------------------------
c
c*************************************************************************
c                        DRIFT_KEPU_LAG.F
c*************************************************************************
c subroutine for solving kepler's equation in universal variables.
c using LAGUERRE'S METHOD
c
c             Input:
c                 s             ==>  inital value of universal variable
c                 dt            ==>  time step (real scalor)
c                 r0            ==>  Distance between 'Sun' and paritcle
c                                     (real scalor)
c                 mu            ==>  Reduced mass of system (real scalor)
c                 alpha         ==>  energy (real scalor)
c                 u             ==>  angular momentun  (real scalor)
c             Output:
c                 s             ==>  final value of universal variable
c                 fp            ==>  f' from p170
```

```
c                                       (real scalors)
c                 c1,c2,c3      ==>  c's from p171-172
c                                       (real scalors)
c                 iflgn          ==>  =0 if converged; !=0 if not
c
c Author:  Hal Levison
c Date:    2/3/93
c Last revision: 4/21/93

      subroutine drift_kepu_lag(s,dt,r0,mu,alpha,u,fp,c1,c2,c3,iflg)

      include 'swift.inc'

c...  Inputs:
      real*8 s,dt,r0,mu,alpha,u

c...  Outputs:
      real*8 fp,c1,c2,c3
      integer iflg

c...  Internals:
      integer nc,ncmax
      real*8 ln
      real*8 x,fpp,ds,c0,f
      real*8 fdt

      integer NTMP
      parameter(NTMP=NLAG2+1)

c----
c...  Executable code

c...    To get close approch needed to take lots of iterations if alpha<0
        if(alpha.lt.0.0) then
           ncmax = NLAG2
        else
           ncmax = NLAG2
        endif

        ln = 5.0
c...    start laguere's method
        do nc =0,ncmax
           x = s*s*alpha
           call drift_kepu_stumpff(x,c0,c1,c2,c3)
           c1 = c1*s
           c2 = c2*s*s
           c3 = c3*s*s*s
           f = r0*c1 + u*c2 + mu*c3 - dt
           fp = r0*c0 + u*c1 + mu*c2
           fpp = (-40.0*alpha + mu)*c1 + u*c0
           ds = - ln*f/(fp + dsign(1.d0,fp)*sqrt(abs((ln - 1.0)*
     &       (ln - 1.0)*fp*fp - (ln - 1.0)*ln*f*fpp)))
           s = s + ds

           fdt = f/dt

c..        quartic convergence
           if( fdt*fdt.lt.DANBYB*DANBYB) then
             iflg = 0
             return
           endif
c...      Laguerre's method succeeded
        enddo

        iflg = 2

        return

        end    !    drift_kepu_leg
c-----------------------------------------------------------------------
c
c***********************************************************************
```

```
c                        DRIFT_KEPU_NEW.F
c*************************************************************************
c subroutine for solving kepler's equation in universal variables.
c using NEWTON'S METHOD
c
c             Input:
c                 s             ==>  inital value of universal variable
c                 dt            ==>  time step (real scalor)
c                 r0            ==>  Distance between 'Sun' and paritcle
c                                     (real scalor)
c                 mu            ==>  Reduced mass of system (real scalor)
c                 alpha         ==>  energy (real scalor)
c                 u             ==>  angular momentun  (real scalor)
c             Output:
c                 s             ==>  final value of universal variable
c                 fp            ==>  f' from p170
c                                       (real scalors)
c                 c1,c2,c3      ==>  c's from p171-172
c                                       (real scalors)
c                 iflgn          ==>  =0 if converged; !=0 if not
c
c Author:  Hal Levison
c Date:    2/3/93
c Last revision: 4/21/93
c Modified by JEC: 31/3/98

      subroutine drift_kepu_new(s,dt,r0,mu,alpha,u,fp,c1,c2,c3,iflgn)

      include 'swift.inc'

c...  Inputs:
      real*8 s,dt,r0,mu,alpha,u

c...  Outputs:
      real*8 fp,c1,c2,c3
      integer iflgn

c...  Internals:
      integer nc
      real*8 x,c0,ds,s2
      real*8 f,fpp,fppp,fdt

c----
c...  Executable code

      do nc=0,6
         s2 = s * s
         x = s2*alpha
         call drift_kepu_stumpff(x,c0,c1,c2,c3)
         c1 = c1*s
         c2 = c2*s2
         c3 = c3*s*s2
         f = r0*c1 + u*c2 + mu*c3 - dt
         fp = r0*c0 + u*c1 + mu*c2
         fpp = (mu - r0*alpha)*c1 + u*c0
         fppp = (mu - r0*alpha)*c0 - u*alpha*c1
         ds = - f/fp
         ds = - f/(fp + .5d0*ds*fpp)
         ds = -f/(fp + .5d0*ds*fpp + ds*ds*fppp*.1666666666666667d0)
         s = s + ds
         fdt = f/dt

c..      quartic convergence
         if( fdt*fdt.lt.DANBYB*DANBYB) then
             iflgn = 0
             return
         endif
c...     newton's method succeeded

        enddo

c..     newton's method failed
```

```
        iflgn = 1
        return

        end  ! drift_kepu_new
c-----------------------------------------------------------------------
c
c*************************************************************************
c                        DRIFT_KEPU_P3SOLVE.F
c*************************************************************************
c Returns the real root of cubic often found in solving kepler
c problem in universal variables.
c
c             Input:
c                 dt            ==>  time step (real scalar)
c                 r0            ==>  Distance between 'Sun' and paritcle
c                                    (real scalar)
c                 mu            ==>  Reduced mass of system (real scalar)
c                 alpha         ==>  Twice the binding energy (real scalar)
c                 u             ==>  Vel. dot radial vector (real scalar)
c             Output:
c                 s             ==>  solution of cubic eqn for the
c                                    universal variable
c                 iflg          ==>  success flag ( = 0 if O.K.) (integer)
c
c Author:  Martin Duncan
c Date:    March 12/93
c Last revision: March 12/93

      subroutine drift_kepu_p3solve(dt,r0,mu,alpha,u,s,iflg)

c...  Inputs:
      real*8 dt,r0,mu,alpha,u

c...  Outputs:
      integer iflg
      real*8 s

c...  Internals:
      real*8 denom,a0,a1,a2,q,r,sq2,sq,p1,p2

c----
c...  Executable code

        denom = (mu - alpha*r0)/6.d0
        a2 = 0.5*u/denom
        a1 = r0/denom
        a0 =-dt/denom

        q = (a1 - a2*a2/3.d0)/3.d0
        r = (a1*a2 -3.d0*a0)/6.d0 - (a2**3)/27.d0
        sq2 = q**3 + r**2

        if( sq2 .ge. 0.d0) then
           sq = sqrt(sq2)

           if ((r+sq) .le. 0.d0) then
              p1 =  -(-(r + sq))**(1.d0/3.d0)
           else
              p1 = (r + sq)**(1.d0/3.d0)
           endif
           if ((r-sq) .le. 0.d0) then
              p2 =  -(-(r - sq))**(1.d0/3.d0)
           else
              p2 = (r - sq)**(1.d0/3.d0)
           endif

           iflg = 0
           s = p1 + p2 - a2/3.d0

        else
           iflg = 1
           s = 0
```

```
        endif

        return
        end     !   drift_kepu_p3solve
c-----------------------------------------------------------------------
c
c*************************************************************************
c                        DRIFT_KEPU_STUMPFF.F
c*************************************************************************
c subroutine for the calculation of stumpff functions
c see Danby p.172  equations 6.9.15
c
c             Input:
c                 x             ==>  argument
c             Output:
c                 c0,c1,c2,c3   ==>  c's from p171-172
c                                    (real scalors)
c Author:  Hal Levison
c Date:    2/3/93
c Last revision: 2/3/93
c Modified by JEC: 31/3/98
c
      subroutine drift_kepu_stumpff(x,c0,c1,c2,c3)

      include 'swift.inc'

c...  Inputs:
      real*8 x

c...  Outputs:
      real*8 c0,c1,c2,c3

c...  Internals:
      integer n,i
      real*8 xm,x2,x3,x4,x5,x6

c----
c...  Executable code

      n = 0
      xm = 0.1
      do while(abs(x).ge.xm)
         n = n + 1
         x = x * .25d0
      enddo
c
      x2 = x  * x
      x3 = x  * x2
      x4 = x2 * x2
      x5 = x2 * x3
      x6 = x3 * x3
c
      c2 = 1.147074559772972d-11*x6 - 2.087675698786810d-9*x5
     %   + 2.755731922398589d-7*x4  - 2.480158730158730d-5*x3
     %   + 1.388888888888889d-3*x2  - 4.166666666666667d-2*x + .5d0
c
      c3 = 7.647163731819816d-13*x6 - 1.605904383682161d-10*x5
     %   + 2.505210838544172d-8*x4  - 2.755731922398589d-6*x3
     %   + 1.984126984126984d-4*x2  - 8.333333333333333d-3*x
     %   + 1.666666666666667d-1
c
      c1 = 1. - x*c3
      c0 = 1. - x*c2
c
      if(n.ne.0) then
         do i=n,1,-1
            c3 = (c2 + c0*c3)*.25d0
            c2 = c1*c1*.5d0
            c1 = c0*c1
            c0 = 2.*c0*c0 - 1.
            x = x * 4.
          enddo
```

```
       endif

       return
       end     !   drift_kepu_stumpff
c-----------------------------------------------------------------------
c
c***********************************************************************
c                        DRIFT_ONE.F
c***********************************************************************
c This subroutine does the danby-type drift for one particle, using
c appropriate vbles and redoing a drift if the accuracy is too poor
c (as flagged by the integer iflg).
c
c             Input:
c                 nbod          ==>  number of massive bodies (int scalar)
c                 mass          ==>  mass of bodies (real array)
c                 x,y,z         ==>  initial position in jacobi coord
c                                    (real scalar)
c                 vx,vy,vz      ==>  initial position in jacobi coord
c                                    (real scalar)
c                 dt            ==>  time step
c             Output:
c                 x,y,z         ==>  final position in jacobi coord
c                                       (real scalars)
c                 vx,vy,vz      ==>  final position in jacobi coord
c                                       (real scalars)
c                 iflg          ==>  integer (zero for successful step)
c
c Authors:  Hal Levison & Martin Duncan
c Date:    2/10/93
c Last revision: 2/10/93
c

      subroutine drift_one(mu,x,y,z,vx,vy,vz,dt,iflg)

      include 'swift.inc'

c...  Inputs Only:
      real*8 mu,dt

c...  Inputs and Outputs:
      real*8 x,y,z
      real*8 vx,vy,vz

c...  Output
        integer iflg

c...  Internals:
        integer i
        real*8 dttmp

c----
c...  Executable code

           call drift_dan(mu,x,y,z,vx,vy,vz,dt,iflg)

           if(iflg .ne. 0) then

             do i = 1,10
               dttmp = dt/10.d0
               call drift_dan(mu,x,y,z,vx,vy,vz,dttmp,iflg)
               if(iflg .ne. 0) return
             enddo

           endif

        return
        end    ! drift_one
c-----------------------------------------------------------------------
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,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_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_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                    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-----------------------------------------------------------------------
```