```
c===========================================================
c     nbody: Solution of gravitational n-body problem
c     using direct summation of (un-softened) forces,
c     global and constant time step, and second order
c     finite-difference technique.
c===========================================================
      program         nbody

      implicit        none

      integer         iargc,        i4arg
      real*8          r8arg

c-----------------------------------------------------------
c     Command-line arguments.
c-----------------------------------------------------------
      real*8          tmax,         dt,        dtout
      real*8          r8_never
      parameter     ( r8_never = -1.0d-60 )

c-----------------------------------------------------------
c     Gravitational constant.  Work in units where G = 1.
c-----------------------------------------------------------
      real*8          G
      parameter     ( G = 1.0d0 )
```

```
c-----------------------------------------------------------
c     "Data-structures" for finite difference evolution:
c
c     mxnpart: Maximum # of particles.
c     ndim:    Number of spatial dimensions (nominally 3)
c     ntlev:   Number of time-levels of position data stored
c     npart:   Actual # of particles.
c
c     r(mxnpart,ndim,ntlev):  Particle positions.
c     a(mxnpart,ndim):        Particle accelerations (F / m).
c     m(mxnpart):             Particle masses.
c     r0(mxnpart,ndim):       Initial particle positions.
c     rdot0(mxnpart,ndim):    Initial particle velocities.
c-----------------------------------------------------------
      integer         mxnpart,         ndim,      ntlev
      parameter     ( mxnpart = 10 000, ndim = 3,  ntlev = 3 )
      real*8          r(mxnpart,ndim,ntlev),
     &                a(mxnpart,ndim),  m(mxnpart),
     &                r0(mxnpart,ndim), rdot0(mxnpart,ndim)
      integer         npart

c-----------------------------------------------------------
c     "Pointers" for various time-levels of data.
c-----------------------------------------------------------
      integer         np1,      n,       nm1

c-----------------------------------------------------------
c     Other locals:
c-----------------------------------------------------------
      integer         it,       nt,       freqout
      real*8          t

c-----------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      if( iargc() .lt. 2 ) go to 900
      tmax  = r8arg(1,r8_never)
      if( tmax .eq. r8_never .or. tmax .le. 0.0d0 ) go to 900
      dt    = r8arg(2,r8_never)
      if( dt .eq. r8_never .or. dt .le. 0.0d0 ) go to 900
      dtout = r8arg(3,dt)
      if( dtout .le. 0.0d0 ) go to 900
      freqout = max(dtout / dt,1.0d0) + 0.5d0
      nt = tmax / dt + 1.5d0

c------------------------------------------------------------
c     Get particle masses, initial positions and velocities
c     from standard input.
c------------------------------------------------------------
      call getid(r0,rdot0,m,mxnpart,ndim,npart)

c------------------------------------------------------------
c     Dump some informative ouput to stderr.
c------------------------------------------------------------
      write(0,1000) npart, tmax, dt, nt, freqout
1000  format(' nbody:  Number of particles: ', i5/
     &      '        Final integration time: ', f8.2/
     &      '        Time step: ', f8.4/
     &      '        Total number of time steps: ',i5/
     &      '        Output frequency: ',i3)

c------------------------------------------------------------
c     Initialize finite difference approximation.
c------------------------------------------------------------
      call initfda(r,a,m,r0,rdot0,mxnpart,ndim,ntlev,
     &             npart,np1,n,nm1,G,dt)

c------------------------------------------------------------
c     Output initial particle positions.
c------------------------------------------------------------
      t = 0.0d0
      call output(r,mxnpart,ndim,ntlev,npart,nm1,t)
      t = t + dt
      if( freqout .eq. 1 ) then
         call output(r,mxnpart,ndim,ntlev,npart,n,t)
      end if
```

```
c-----------------------------------------------------------
c     T I M E   S T E P   L O O P
c-----------------------------------------------------------
      do it = 3 , nt
c-----------------------------------------------------------
c        Compute accelerations.
c-----------------------------------------------------------
         call calca(a,r,m,mxnpart,ndim,ntlev,npart,n,G)
c-----------------------------------------------------------
c        Update positions.
c-----------------------------------------------------------
         call update(r,a,mxnpart,ndim,ntlev,npart,
     &               np1,n,nm1,dt)
         t = t  + dt
c-----------------------------------------------------------
c        Swap "pointers".
c-----------------------------------------------------------
         call cyclelevels(np1,n,nm1)
c-----------------------------------------------------------
c        Periodic output of positions.
c-----------------------------------------------------------
         if( mod((it-1),freqout) .eq. 0 ) then
            call output(r,mxnpart,ndim,ntlev,npart,n,t)
         end if
      end do

      stop

 900  continue
         write(0,*) 'usage: nbody <tmax> <dt> [<dt out>]'
         write(0,*)
         write(0,*) '       Reads masses, initial positions'
         write(0,*) '       and velocities from standard '
         write(0,*) '       input (7 numbers per line)'
      stop

      end
```

```
c-----------------------------------------------------------
c     getid: Reads particle masses, initial positions
c     and initial velocities from standard input. Returns
c     number of particles.  Data format:
c
c     m  x_0  y_0  z_0  vx_0  vy_0  vz_0
c-----------------------------------------------------------
      subroutine getid(r0,rdot0,m,mxnpart,ndim,npart)
         implicit      none

         integer       mxnpart, ndim, npart
         real*8        r0(mxnpart,ndim), rdot0(mxnpart,ndim),
     &                 m(mxnpart)

         integer       i,      rc

         npart = 0
 100     continue
            if( npart .ge. mxnpart ) then
               write(0,*) 'getid: Read initial data for ',
     &            'maximum of ', mxnpart, ' particles.'
               return
            end if
            i = npart + 1
            read(*,*,iostat=rc,end=200)  m(i),
     &           r0(i,1),    r0(i,2),    r0(i,3),
     &           rdot0(i,1), rdot0(i,2), rdot0(i,3)
            if( rc .eq. 0 ) then
               npart = npart + 1
            end if
         go to 100
 200     continue

         return

      end
```

```
c-----------------------------------------------------------
c     initfda: Initializes second order FDA using initial
c     positions and velocities of particles and Taylor
c     series expansion up to and including terms of order
c     dt**2.
c-----------------------------------------------------------
      subroutine initfda(r,a,m,r0,rdot0,mxnpart,ndim,ntlev,
     &                  npart,np1,n,nm1,G,dt)
         implicit      none

         integer       mxnpart, ndim, ntlev, npart, np1, n,
     &                 nm1
         real*8        r(mxnpart,ndim,ntlev), a(mxnpart,ndim),
     &                 r0(mxnpart,ndim), rdot0(mxnpart,ndim),
     &                 m(mxnpart)
         real*8        G,    dt

         integer       i,     k
         real*8        hdtsq
```

```
c------------------------------------------------------------
c       Initialize pointers
c------------------------------------------------------------
        nm1 = 1
        n   = 2
        np1 = 3
c------------------------------------------------------------
c       Initialize t = 0 positions.
c------------------------------------------------------------
        do k = 1 , ndim
           do i = 1 , npart
              r(i,k,nm1) = r0(i,k)
           end do
        end do
c------------------------------------------------------------
c       Compute t = 0 accelerations.
c------------------------------------------------------------
        call calca(a,r,m,mxnpart,ndim,ntlev,npart,nm1,G)

c------------------------------------------------------------
c       Compute t = dt positions using initial velocities
c       and Taylor series.
c------------------------------------------------------------
        hdtsq = 0.5d0 * dt**2
        do k = 1 , ndim
           do i = 1 , npart
              r(i,k,n) = r(i,k,nm1) + dt * rdot0(i,k) +
     &                   hdtsq * a(i,k)
           end do
        end do

        return

      end
```

```
c-----------------------------------------------------------
c     calca: Calulates particle accelerations via direct
c     summation of pair-wise gravitational forces.
c     positions and velocities of particles and Taylor
c     series expansion up to and including terms of order
c     dt**2.
c-----------------------------------------------------------
      subroutine calca(a,r,m,mxnpart,ndim,ntlev,npart,n,G)
         implicit      none

         integer       mxnpart, ndim, ntlev, npart, n
         real*8        r(mxnpart,ndim,ntlev), a(mxnpart,ndim),
     &                 m(mxnpart)
         real*8        G

         real*8        rsq,     ca1
         integer       i,       j,      k
```

```
c------------------------------------------------------------
c        For each particle ...
c------------------------------------------------------------
         do i = 1 , npart
c------------------------------------------------------------
c           Zero all components of the particle's accn.
c------------------------------------------------------------
            do k = 1 , ndim
               a(i,k) = 0.0d0
            end do
c------------------------------------------------------------
c           For all of the other particles ...
c------------------------------------------------------------
            do j = 1 , npart
               if( i .ne. j ) then
c------------------------------------------------------------
c                 Compute the square of the separation,
c------------------------------------------------------------
                  rsq = 0.0d0
                  do k = 1 , ndim
                     rsq = rsq + (r(j,k,n) - r(i,k,n))**2
                  end do
c------------------------------------------------------------
c                 ... then update each component of the ith
c                 particle's accn.
c------------------------------------------------------------
                  ca1 = G * m(j) / (rsq ** 1.5d0)
                  do k = 1 , ndim
                     a(i,k) = a(i,k) +
     &                        ca1 * (r(j,k,n) - r(i,k,n))
                  end do
               end if
            end do
         end do

         return

      end
```

```
c-----------------------------------------------------------
c     update: Updates particle positions using second
c     order FDA and previously computed accelerations.
c-----------------------------------------------------------
      subroutine update(r,a,mxnpart,ndim,ntlev,npart,
     &                  np1,n,nm1,dt)
         implicit      none

         integer       mxnpart, ndim, ntlev, npart
         real*8        r(mxnpart,ndim,ntlev), a(mxnpart,ndim)

         integer       np1, n, nm1
         real*8        dt

         real*8        dtsq
         integer       i,      k,     ntmp

c-----------------------------------------------------------
c        Straightforward implementation of FDA.
c-----------------------------------------------------------
         dtsq = dt**2
         do k = 1 , ndim
            do i = 1 , npart
               r(i,k,np1) = 2.0d0 * r(i,k,n) - r(i,k,nm1) +
     &                      dtsq * a(i,k)
            end do
         end do

         return

      end
```

```
c-----------------------------------------------------------
c     Swaps 'np1, n, nm1' "pointers" to effect time step.
c     On exit time-level 'n' refers to most current data.
c-----------------------------------------------------------
      subroutine cyclelevels(np1,n,nm1)
         implicit        none

         integer         np1,    n,    nm1,     ntmp

         ntmp = nm1
         nm1  = n
         n    = np1
         np1  = ntmp

         return

      end
c-----------------------------------------------------------
c     output: Outputs particle positions.  Currently
c     configured to send particle (x, y) coordinates to
c     Choptuik's 'ser' program.
c-----------------------------------------------------------
      subroutine output(r,mxnpart,ndim,ntlev,npart,n,t)
         implicit      none

         integer       vsxynt,      vsrc

         integer       mxnpart, ndim, ntlev, npart,  n
         real*8        r(mxnpart,ndim,ntlev)
         real*8        t

         vsrc = vsxynt('position (xy)',t,
     &                 r(1,1,n), r(1,2,n), npart)
         write(0,1000) t
1000     format(' output: t = ',f10.3)

         return

      end
```

```
#############################################################
# Building 'nbody' and sample run on SGIs
#############################################################
einstein% pwd; ls
/usr2/people/phy329/part/ex1
Makefile   binary     nbody.f

einstein% make
        f77 -g -c nbody.f
        f77 -g -L/usr/local/lib nbody.o -lfvs -lp329f -o nbody

einstein% nbody
 usage: nbody <tmax> <dt> [<dt out>]

        Reads masses, initial positions
        and velocities from standard
        input (7 numbers per line)

#############################################################
# Initial data for equal-mass binary
#############################################################
einstein% more binary
1.0    1.0 0.0 0.0   0.0 -0.5 0.0
1.0   -1.0 0.0 0.0   0.0  0.5 0.0
```

```
###############################################################
# This invocation integrates for about 1/3 of an orbit, with
# output every time-step.  Tracing output also occurs every
# time-step and is only partially reproduced here.
###############################################################
einstein% nbody 4.0 .01 < binary
 nbody:  Number of particles:     2
         Final integration time:     4.00
         Time step:   0.0100
         Total number of time steps:   401
         Output frequency:   1
 >>> vsxynt:: Opened <positionxy.segdat>
 output: t =      0.000
 output: t =      0.010
 output: t =      0.020
 output: t =      0.030
 output: t =      0.040
 output: t =      0.050
           .
           .
           .
 output: t =      4.000
```