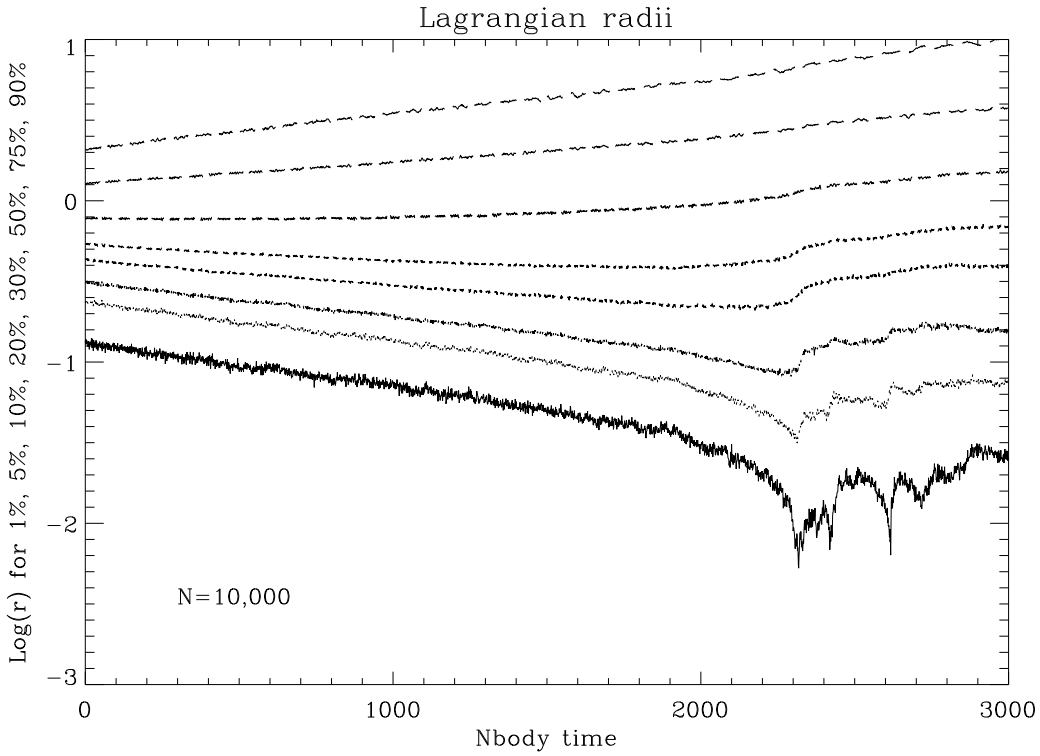
NBODY6++

Features of the computer–code

Emil Khalisi, Rainer Spurzem

Astronomisches Rechen–Institut
Mönchhofstr. 12–14, 69120 Heidelberg, Germany



This is an intermediate text version released for the cambody school Aug. 2006.
Please watch for updates on the ftp site given in the text.

Latest update: 9. August 2006

Table of contents

1 Introduction

Gravity is an ever–present force in the Universe and is involved into the dynamics of all kinds of bodies, from the tiny atom to the clusters of galaxies. At small spatial scales, its influence is covered by other strong forces (e.g. magnetic, pressure, radiation induced), while on the very large scale it becomes the most dominant power. In astrophysics, it governs the dynamical evolution of many self–gravitating systems. Here, we concentrate on such systems that are dominated by mutual gravitation between particles.

The numerical star-by-star simulation of a simple cluster containing some more than hundred thousand members still places heavy demands on the available hard- and software. A balance has to be found between two constraints: On one hand the *realism*, i.e. the input of profound physics, inclusion of all astrophysical effects as well as the maintenance of the accuracy of calculations; and on the other hand, the *efficiency*, i.e. the limitations given by the computational possibilities and suitable codes to be finished in a reasonable time. Many different kinds of approaches have been undertaken to suffice both:

- codes based on the direct force integration [2], [5], [6], see also: `http://www.sverre.com/`,
- statistical models, which themselves divide into several subgroups (Fokker–Planck approximation by [10]; Monte–Carlo method by [13]; Gas models by [27]),
- usage of high-performance parallel computers [28], [11],
- or the construction of special hardware devoted for these purposes (GRAPE [19], see also: `http://www.astrograpе.org/` and `http://www.ari.uni-heidelberg.de/grace/` `http://www.cs.rit.edu/~grapecluster/`.

The code NBODY6++ described in this manual is designed for an accurate integration of many bodies (e.g. in a star cluster, planetary system, galactic nucleus) based on the direct integration of the Newtonian equations of motion. It is optimal for collisional systems, where long times of integration and high accuracy or both are required, in order to follow with high precision the secular evolution of the objects.

NBODY6++ is a descendant of the family of NBODY codes initiated by Sverre Aarseth [4], which has been extended to be suitable for parallel computers [28]. The basic features of the code increasing the efficiency may be considered under four separate headings: fourth order prediction–correction method (Hermite scheme), individual and block time–steps, regularization of close encounters and few-body subsystems, and a neighbour scheme (Ahmad–Cohen scheme). We briefly describe these ideas in this booklet, while a detailed description can be found in [3] as well as his book [6].

While NBODY6++ is not that different from NBODY6 to justify a completely new name, the user should, however, be aware that in order to make a parallelization of regular and irregular force computations possible at all, some significant changes in the order of operations became necessary. As a consequence, trajectories of the same initial system, simulated by NBODY6 and NBODY6++ will diverge from each other, due to the inherent exponential instability and deterministic chaos in N-body systems. Still one should always expect that the *global* properties are well behaved in both cases (e.g. energy conservation). While much effort is taken to keep NBODY6 and NBODY6++ as close as possible this is never 100% the case, and the interested reader should always contact Sverre Aarseth or Rainer Spurzem if in doubt about these matters.

This manual should serve as a practical starter kit for new students working with NBODY6++. It is not meant as a complete reference or scientific paper; for that see the references and in particular the excellent compendium of Aarseth's book on Gravitational N-Body Simulations [6].

Acknowledgements

The authors of this manual would like to express their sincere gratitude to Sverre Aarseth and Seppo Mikkola, for their continuous support and work over the decades. Also, many students and postdocs in Heidelberg and elsewhere have contributed towards development, debugging and improving the software for the benefit of the community. Particular thanks in Heidelberg are due to Patrick Glaschke, Andreas Ernst, Kristin Warnick, Andrea Borch, Chingis Omarov. This booklet was written at the Astronomisches Rechen–Institut, ZAH, Univ. of Heidelberg.

2 Code versions

The development of the NBODY–code has begun in the 1960ies [1], though there exist some earlier precursors [29], [30]. It has set a quasi-standard for the precise direct integration of gravitating many-body systems. There exist several code groups (NBODY1–7, and a number of special implementations) for different usage, some of which are rather of historical interest.

The NBODY6++ code is available publicly under
`ftp://ftp.ari.uni-heidelberg.de/pub/staff/spurzem/nb6mpi/`.
In irregular intervals updates of the code are provided on this ftp-site. These updates implement changes communicated by S. Aarseth for the standard NBODY6 code (but note that not always it is possible to implement these updates one-to-one, although a maximum synchronicity between NBODY6 and NBODY6++ is aimed at), but also genuine updates and bugfixes which apply to NBODY6++ only. Usually there will be a special README.xxx file provided (where xxx is the timestamp such as june2003 or mar2005), detailing the changes.

The original N–body codes can be accessed publicly via Sverre Aarseth's ftp and web sites at `ftp://ftp.ast.cam.ac.uk/pub/sverre/` and `http://www.sverre.com/`.

A brief comparison of the code versions:

ITS: Individual time–steps
ACS: Neighbour scheme (Ahmad–Cohen scheme) with block time–steps
KS: KS–regularization of few-body subsystems
HITS: Hermite scheme integration method combined with hierarchical block time steps

	ITS	ACS	KS	HITS
NBODY1	•			
NBODY2		•		•
NBODY3	•		•	
NBODY4			•	•
NBODY5	•	•	•	
NBODY6(++)		•	•	•

3 Getting started

The code NBODY6++ is written in Fortran and consists of about 250 files. Their functionality was improved as well as new routines included all the way through the decades along with the technological achievements of the hardware. The starting (main) routine is called `nbody6.F` and begins as given in Figure 3.1. We will return to the very first operations of this routine later.

To get the code running, you need to download and unpack the package nb6mpi–*version*.tar.gz :

homedir`> gzip -d -c nb6mpi-`*version*`.tar.gz | tar xvf -`

A directory will be created containing all the source files (routines and functions). By default the directory is called `nb6mpi-`*version*`/` , and it can be renamed to any other name; for example, we will change it to `Nbody` for convenience.

Most of the files have the suffix "`.f`", some "`.F`" and a few "`.h`". All `.f`–files are directly read by a Fortran compiler. The `.F`'s will pass a pre–compiler first, which selects code lines separated by pre–compiler options, e.g. between `#ifdef PARALLEL` and `#endif` , for they activate the parallel code on different multi–processor machines. By this, some portability between different hardware is ensured at least, and a single processor version of the code can easily be compiled as well. The `.h` are header–files and declare the variables and their blocks.

The files have to be compiled on the computer you are working at. There are several options for compilation, and their usage is explained in the `Makefile` . For example:

homedir`> cd Nbody`
homedir`/Nbody> make nbody6`

will apply the standard Fortran compiler on your hardware. Other targets are also specified for different hardware such as PC clusters, CRAY T3E or Sun parallel machines. Compilations like `make pgf` or `make pgfp4` make use of commercial Fortran compilers and special optimisation for individual processors. More of them can be created by the user or provided by the authors on request. The compilation creates machine–readable object files with the suffix `.o` . The files are linked to a resulting executable with the name `nbody6` . Now, the code is ready to start.

It is recommendable to start the simulations in another folder, e.g. `Run/` and copy the executable therein. While unpacking, this folder is created by default and some trial files are delivered. Copy the newly compiled executable to this directory:

homedir`/Nbody> cd Run`
homedir`/Nbody/Run> cp ../nbody6 .`

Depending on the user's individual research, the Nbody code opens a wide field of application possibilities. The user has to define his model by a number of input control variables, e.g. number of stars, the size of the cluster, a mass function, profile, and many more. These control variables are gathered in the file `input1000` . The detailed explanation of its handling is given in Chapter 4. Alternatively, a data file named `dat.10` can be used, which contains data for an initial configuration (see Ch. **??**). If the model criteria are defined, a single processor simulation run is started with the command

homedir`/Nbody/Run> ./nbody6 < input1000 > out1000 &`

```
          PROGRAM NBODY6
*
*             N B O D Y 6++
*             *************
*
*       Regularized AC N-body code with triple & binary collisions.
*       -----------------------------------------------------------
*
*       Hermite integration scheme with block-steps (V 4.0.0 April/99).
*       ---------------------------------------------------------------
*
*       Developed by Sverre Aarseth, IOA, Cambridge.
*       ............................................
*       Message Passing Version NBODY6++ for Massively Parallel Systems
*       Developed by Rainer Spurzem, ARI, Heidelberg
*
      INCLUDE 'common6.h'
      COMMON/STSTAT/  TINIT,NIR,NIB,NRGL,NKS
      EXTERNAL MERGE
*
#ifdef PARALLEL
#define MPIINIT 1
#else
#ifdef ENSEMBLE
#define MPIINIT 1
#else
#define MPIINIT 0
#endif
#endif

#if MPIINIT
*       Initialize MPI
      CALL MPI_INIT(ierr)
      CALL MPI_COMM_GROUP(MPI_COMM_WORLD,group,ierr)
      CALL MPI_GROUP_SIZE(group,isize,ierr)
      CALL MPI_GROUP_RANK(group,rank,ierr)
*     PRINT*,' This is rank=',rank,' size=',isize,' group=',group
#endif
*
*       Initialize the timer.
      CALL CPUTIM(ttota)
*
*       Read start/restart indicator & CPU time.
      IF(rank.eq.0)READ (5,*)  KSTART, TCOMP, TCRITp,
     *    isernb,iserreg
*
#if MPIINIT
      CALL MPI_BCAST(isernb,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(iserreg,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(KSTART,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(TCOMP,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(TCRITp,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
*
      iserreg = max(isize,iserreg)
      isernb = max(isize,isernb)
      IF(rank.eq.0)PRINT*,' iserreg,isernb=',iserreg,isernb
#endif
*
      IF (KSTART.EQ.1) THEN
*
*       Read input parameters, perform initial setup and obtain output.
```

Figure 3.1: The beginning of the Nbody6–code.

In this example, the code reads the control variables given in the file `input1000` from Unix standard input *stdin*. Then, a star cluster is created according to the user's instructions, and the bodies are moved one by one with respect to their time maturity. In NBODY6 they are due one by one in individual steps (which might be blocked together in groups, though), in NBODY6++ they are moved simultaneously forward in a group (see Chapter **??**). In certain time intervals (controlled by the user), some first results and error checks are directed via the Unix standard output *stdout* to `out1000` . This file provides snapshots of the state of the system for a brief overview of some key data of the simulation to judge about the quality and performance of the run.

There are several more files created, many of them binary files. Most important are `comm.1` and `comm.2`, which contain dumps of the complete common blocks for a restart and checkpoint purposes, and `conf.3` that contains the particle data for the user's analysis. In the latter, many details of the run are saved, e.g. positions, velocities, neighbour densities, potential of *each* particle in *any* predefined time interval. The volume of data in all three mentioned files critically depends on the dimensions of vectors in `params.h` . Here, the particle data plus some user-defined dimensions are given a threshold in order to save disk space when outputting to `conf.3` — see Chapter 5.

At the time of this writing, the user has to provide own routines to postprocess the particle data from the simulation, using e.g. additional routines or programs (like IDL, gnuplot etc.), in order to extract the binary data from this file and plot graphics. Work is in progress to provide a better visual interface delivered with the program.

A run will be finished when one of 4 conditions becomes true:

- the specified CPU–time on the computer is exceeded (variable `TCOMP` in the input file), or
- the maximum Nbody–time (see Ch. 4) is reached (variable `TCRIT`), or
- the physical cluster time in Myr is reached (variable `TCRITp`), or
- the number of cluster stars has fallen below a minimum (variable `NCRIT`).

A soft termination of a running simulation can be realized by generating of a file `STOP` in the executing directory:

homedir`/Nbody/Run> touch STOP`

In that case, a checkpoint of the code is done, which is located in the routine `intgr.F` and shown in Figure 3.2. The program writes out the current variables, saves a complete common dump in

```
*
*       Include facility for termination of run (create dummy file STOP).
      IF(rank.EQ.0)THEN
      OPEN (99,FILE='STOP',STATUS='OLD',FORM='FORMATTED',IOSTAT=IO)
      IF (IO.EQ.0) THEN
          CLOSE (99)
          IF (NSUM.EQ.0.and.rank.eq.0) WRITE (6,90)
   90     FORMAT  (/,9X,'TERMINATION BY MANUAL INTERVENTION')
          CPU = 0.0
      END IF
      END IF
*
```

Figure 3.2: Soft interruption of a simulation run in `intgr.F`: If the dummy file "STOP" exists, then the run terminates.

`comm.1` and terminates. The run can be restarted and continued from the same point where it was left.

Before a restart, it is recommendable to copy or rename the files, otherwise they may be overwritten. Any file `comm.1` and `comm.2` is restartable. The different names are just for getting common dumps at different time units. For example, if an irregular termination takes place, `comm.2` contains the data at some earlier time point, while `comm.1` always contains the last time data.

To restart a run, a different very short input control data file needs to be used, because most of the control data are already stored in `comm.1` . Only the first line corresponds to the standard input file, but the first input variable, `KSTART`, has to be changed to "2" or higher. In this case, the routine `modify.F` will be entered.

KSTART	Function
1	new run, start from initial values given in `data.F`
2	continuation of a run without changes
3	restart of a run with changes of the following parameters given in the second line of a newly created input file: DTADJ, DELTAT, TADJ, TNEXT, TCRIT, QE, J, K where the options KZ can be changed via KZ(J)=K
4	restart of a run with following parameters changed in the second line: ETAI, ETAR, ETAU, DTMIN, RMIN, NNBOPT
5	restart of a run with all parameter changes in the run control index 3 and 4. The changes must succeed the first line.

"0" values in the fields are interpreted as: Do not change the value of this parameter. Example: A new input file

```
3   10000   1.E6   40   40
0   2.0     0      0    2000   2.0E-06   30   0
```

will change the values of DELTAT=2.0, TCRIT=2000, QE=$2{\cdot}10^{-6}$, and KZ(30)=0 . Note: It is only possible to change one of the KZ-vector values at a restart.

4 Input variables

The input control file of NBODY6++, e.g. `in1000`, contains a minimum of 83 parameters which steer one simulation run for its technical and physical properties (it is very similar but not identical to the one used for NBODY6). As for the technical aspect, the file supervises the run e.g. for its duration, intervals of the output, or error check; the physical parameters concern the size of a cluster, initial conditions, or a number of optional features related to the numerical problem to be studied. The handling of this input file appears rather entangled at first sight, for it has grown rather historically and "ready–for–use" than customer–oriented. Thus, the input variables are read by different routines (functions) in the code, and the nature of the parameters are woven with each other in some cases. Also, some paramters require additional input, such that the total number of lines and parameters may vary.

In the following, we explain the main input file and give an example of typical values for a simulation of an isolated globular cluster. Then, we proceed to the thresholds.

`in1000`:

```
KSTART   TCOMP    TCRITp   isernb   iserreg
N        NFIX     NCRIT    NRAND    NNBOPT   NRUN
ETAI     ETAR     RS0      DTADJ    DELTAT   TCRIT    QE       RBAR     ZMBAR
KZ(1)    KZ(2)    KZ(3)    KZ(4)    KZ(5)    KZ(6)    KZ(7)    KZ(8)    KZ(9)    KZ(10)
KZ(11)   KZ(12)   KZ(13)   KZ(14)   KZ(15)   KZ(16)   KZ(17)   KZ(18)   KZ(19)   KZ(20)
KZ(21)   KZ(22)   KZ(23)   KZ(24)   KZ(25)   KZ(26)   KZ(27)   KZ(28)   KZ(29)   KZ(30)
KZ(31)   KZ(32)   KZ(33)   KZ(34)   KZ(35)   KZ(36)   KZ(37)   KZ(38)   KZ(39)   KZ(40)
BZ(1)    BZ(2)    BZ(3)    BZ(4)    BZ(5)    BZ(6)    BZ(7)    BZ(8)    BZ(9)    BZ(10)
DTMIN    RMIN     ETAU     ECLOSE   GMIN     GMAX
ALPHA    BODY1    BODYN    NBIN0    ZMET     EPOCH0
Q        VXROT    VZROT    RSPH2
NBIN     SEMI0    ECC0     RATIO    RANGE    NSKIP    IDORM
```

KSTART Run control index
=1: new run (construct new model or read from `dat.10`)
=2: restart/continuation of a run, needs `comm.1`
=3: restart + changes of DTADJ, DELTAT, TADJ, TNEXT, TCRIT, QE, KZ
=4: restart + changes of ETAI, ETAR, ETAU, DTMIN, RMIN, NNBOPT
=5: restart containing the combination of the control index 3 and 4

TCOMP Maximum wall-clock time in seconds (parallel runs: wall clock)

TCRITp Termination time in Myrs

isernb For parallel runs: only irregular block sizes larger than this value are executed in parallel mode (dummy variable for single CPU)

iserreg For parallel runs: only regular block sizes larger than this value are executed in parallel mode (dummy variable for single CPU)

N Total number of particles (single + c.m.s. of binaries; < NMAX−2)

NFIX Multiplicator for output interval of data on `conf.3` and of data for binary stars (compare KZ(3) and KZ(6))

NCRIT Minimum particle number (termination criterion)

NRAND Random number seed; any positive integer

NNBOPT Desired optimal neighbour number

NRUN Run identification index

ETAI	Time–step factor for irregular force polynomial
ETAR	Time–step factor for regular force polynomial
RS0	Initial guess for all radii of neighbour spheres
DTADJ	Time interval for parameter adjustment and energy check (in TCR if KZ(35)=0)
DELTAT	Time interval for writing output data and diagnostics, multiplied by NFIX – in units of $t_{\rm cr}$ if KZ(35) = 0; – in scaled units if KZ(35) > 0. for DTADJ=DELTAT (recommended) output data is written every adjust time
TCRIT	Termination time in units of TCR (if KZ(35)=0)
QE	Energy tolerance: – immediate termination if DE/E > 5*QE & KZ(2) < 1; – restart if DE/E > 5*QE & KZ(2) > 1; – termination after second restart attempt, otherwise.
RBAR	Scaling unit in pc for radial N–body unit
ZMBAR	Scaling unit for average particle mass in solar masses (in scale-free simulations RBAR and ZMBAR can be set to zero)

KZ(1)	COMMON save on unit 1, file `comm.1` =1: at end of run =2: every 100*NMAX steps
KZ(2)	COMMON save on unit 2, file `comm.2` =1: at output =2: restart if energy error DE/E > 5*QE
KZ(3)	=1: Basic data on `conf.3` at output time (frequency NFIX)
KZ(4)	Binary diagnostics on `bdat.4` (# = threshold levels <10)
KZ(5)	Initial conditions of the particle distribution =0: uniform & isotropic =1: Plummer random placing
KZ(6)	Output of significant and regularized binaries on unit 6 =1: output of regularized and significant binaries (|E|>0.1 ECLOSE) =2: output of regularized binaries only =4: output of regularized binaries only, frequency NFIX
KZ(7)	Determine Lagrangian radii, routine `lagr.F` =2: write to unit 6 (diagnostics) =3: write to unit 7, file (`lagr.7`) =4: write to both units 6 and 7
KZ(8)	Primordial binaries, routine `binpop.f` =0: (Note possible reading of primordial binaries from `dat.10`, if KZ(8)=0, KZ(24)>0, NBIN0>0) ≥ 2: create binary data banks on units 8 and 9 > 3: data on hierarchies on unit 13, file `hid.13`
KZ(9)	Individual bodies printed at output time (MIN(5**KZ9,NTOT))
KZ(10)	KS output to unit 6 (diagnostics) >0: begin >1: end $\geq$3: each step

KZ(11) Hierarchical systems, routine `hiarch.f`
=1,3: write to unit 12, file `hia.12`
=2: create primordial triples, routine `hipop.f`
KZ(12) HR diagnostics of evolving stars (interval DTPLOT)
KZ(13) Interstellar clouds (<0 or >2: Gaussian velocities)
KZ(14) External force
=1: standard tidal field
>1: reserved for other external field, not implemented yet
KZ(15) Triple, quad, chain (with KZ(30) > 0) or merger search
>1: full output
KZ(16) Updating of regularization parameters (RMIN, DTMIN & ECLOSE)
KZ(17) Modification of ETAI, ETAR (≥ 1) and ETAU (>1) by tolerance QE
KZ(18) Neighbour addition in routine `checkl.f`
=1: high velocity, LISTV
=2: all types
KZ(19) Mass loss
=1: supernova scheme
=3: Eggleton, Tout & Hurley
KZ(20) Initial mass functions:
=0: self-defined power-law (`data.F`)
=1: Miller-Scalo-(1979) IMF (`imf.f`)
=2: KTG (1993) with random pairing (`imf2.f`)
=3: Eggleton-IMF (`imf2.f`)
=4: KTG(1993) with binary-ratio $(m_1/m_2)\prime = (m_1/m_2)^{0.4}$ + const.
=5: Eggleton-IMF with binary-ratio, see #4
=6: KTG(1993) extended to Brown Dwarf regime (`imfbd.f`)
KZ(21) Extra output line (MODEL no., TCOMP, DMIN, AMIN, RMAX & RSMIN)
KZ(22) Initial data of m, r, v on unit 10
=0: uniform or Plummer sphere, see KZ(5)
=1: write initial conditions to diagnostics (`scale.F`)
=2: input through NBODY–format
=3: input through Tree-format and no scaling (`data.F`)
=4: input through Starlab-format
=6: input through NBODY-format + Scaling
=7: input through Tree-Format + Scaling
=8: input through Starlab-format + Scaling
KZ(23) Removal of escapers, routine `escape.F`
=1: from isolated cluster
=2,4: write to diagnostics and to unit 11, file `esc.11`
=3: clusters with tidal cut–off
KZ(24) Initial conditions for subsystems (routine `scale.F`)
KZ(25) Partial reflection of KS binary orbit (GAMMA < GMIN; suppressed)
KZ(26) Slow-down of two-body motion
$\geq$1: for KS binary
=2: for chain binary
KZ(27) Two–body tidal interaction (n=1.5: type 3 and 5; n=3: others)
KZ(28) Magnetic braking and gravitational radiation (with KZ(19) and KZ(27))
KZ(29) Boundary reflection for hot system (suppressed)

KZ(30) Chain regularization
=0: no chain regularization
$\geq$=2: main output
>2: write to diagnostics
KZ(31) Centre of mass correction after energy check
KZ(32) Increase of output intervals (based on single particle energy)
KZ(33) Block-step statistics on unit 6 (diagnostics)
>0: if primordial binaries: STEPU
$\geq$1: STEP
=2: STEPR
KZ(34) Roche lobe overflow (not implemented yet)
KZ(35) TIME reset to zero every 100 time units, total time in TTOT
KZ(36) Step reduction for hierarchical systems, routines `nbint.f, kepler.f`
KZ(37) Fast time-step criterion, routine `nbint.f`
KZ(38) No force polynomial corrections (I $\leq$ N)
KZ(39) =2: Shape analysis, routine `ellan.f` (by Ch. Theis)
KZ(40) Adjust neighbour number to optimal neighbour number, routine `regint.f`

BK(1) =0: no proto-star evolution
=1: "proto-star" evolution of eccentricity and period in `binpop_4new.f`
BK(2) = -1: use NBGR and REDUCE in `binpop_pk.f`
=0: flat distribution in semi–major axis
=1: f=$0.034388 \cdot \log P$
=2: f=$3.5 \cdot \log P/[100 + (\log P)^2]$; KZ(40)=1,2 are 1st and 2nd iterations
=3: f=$2.3 \cdot (\log P - 1)/[45 + (\log P - 1)^2]$
=4: f=$2.5 \cdot (\log P - 1)/[45 + (\log P - 1)^2]$; derived in K2; KZ(40)>0 is used to adjust neighbour number (`adjust.F`)
=5: f = Gaussian in $\log P$, Duquennoy & Mayor (1991)
BK(3) *unused*
BK(4) =1: open and write to unit 15, see `ksint.f`
BK(5) *unused*
BK(6) *unused*
BK(7) *unused*
BK(8) *unused*
BK(9) *unused*
BK(10) *unused*

DTMIN Time–step criterion for regularization search
RMIN Distance criterion for regularization search
ETAU Regularized time-step parameter (6.28/ETAU steps/orbit)
ECLOSE Binding energy per unit mass for hard binary (positive)
GMIN Relative two-body perturbation for unperturbed motion
GMAX Secondary termination parameter for soft KS binaries

ALPHA Power-law index for initial mass function, routine `data.F`
BODY1 Maximum particle mass before scaling

BODYN	Minimum particle mass before scaling
NBIN0	Number of primordial binaries: – by routine `imf2.F` using a binary IMF (KZ(20)$\geq$2) – by routine `binpop.F` splitting single stars (KZ(8)>0) – by reading subsystems from unit 10 `dat.10` (KZ(24)>0)
ZMET	Metal abundance (in range 0.03 - 0.0001)
EPOCH0	Evolutionary epoch (in 10^6 yrs)

Q	Virial ratio (routine `scale.F`; Q=0.5 for equilibrium)
VXROT	XY–velocity scaling factor (> 0 for solid-body rotation)
VZROT	Z–velocity scaling factor (not used if VXROT = 0)
RSPH2	Radius of reflecting sphere (see KZ(29); units of RSCALE)

NBIN	Number of initial binaries (routine `binpop.f`)
SEMI0	Initial semi-major axis (= 0 for range of energies)
ECC0	Initial eccentricity < 0: thermal distribution, $f(e) = 2e$ $0 \leq$ ECC0 ≤ 1: fixed eccentricity =20, 30, 40: see `binpop.F`
RATIO	Mass ratio M1/(M1 + M2) =1.0: M1 = M2 = $\langle$M$\rangle$
RANGE	Defines range of distribution in semi major axis
NSKIP	Binary frequency of mass spectrum (starting from body #1)
IDORM	Indicator for dormant binaries (>0: merged components)

A typical input file can look like as follows. It defines a new simulation running for 1,000,000 CPU–minutes with $N = 1,000$ particles distributed along a Plummer profile (KZ(5)=1). The run may alternatively terminate when TCRIT=1000.0, or if a final particle number of NCRIT=10 has been reached. The initial mass function follows a power–law with an index of $\alpha = 2.35$ (KZ(20)=0 and ALPHA=2.35), ranging from $m_{\rm max} = 5.0$ to $m_{\rm min} = 1.0$ (BODY1 and BODYN), *etc.*

```
1 1000000.0 1.E6 40 40
1000 1 10 183 50 1
0.01 0.02 0.3 10.0 10.0 1000.0 2.0E-05 1.0 0.7
1 1 1 0 1 1 4 0 0 2
1 0 0 0 2 1 0 2 0 0
1 0 2 0 0 2 0 0 0 2
0 0 2 0 1 0 1 1 0 1
0 0 0 0 0 0 0 0 0 0
1.0E-04 0.01 0.1 1.0 1.0E-06 0.01
2.35 20.0 0.1 0 0.0 0.0
0.5 0.0 0.0 0.0
0 0.005 -1.0 1.0 5.0 5 0
```

Input variables for primordial Binaries

Many star clusters contain initial hard binaries with binding energies much larger than the thermal energy (the threshold ECLOSE is a suitable division between hard and soft binaries). There are two ways to initialise primordial binaries:

The first one always starts from some initial mass function (IMF) provided by the routines `imf.f` or `imf2.f`. The option KZ(8)>0 invokes the routine `binpop.F`, which reads the last line of the input file containing NBIN and the parameters of their distribution (see above). In case of KZ(8)>0, binaries are created either by random pairing of single stars obtained from the IMF or by splitting them, depending on the value of KZ(20) — see there.

The second way assumes that particle data, including the binaries, are provided via the input data on file `dat.10` (as e.g. in the Kyoto–II collaborative experiment). In such a case KZ(8)=0 and NBIN=0 should be set, but KZ(24)>0 and NBIN0>0 must be equal to the expected number of regularized binaries from the file. The code will first create NBIN0 centers of masses, and then use those for scaling, before regularizing the pairs.

A typical input file with primordial binaries looks as follows. Here, we use binary random pairing from `imf2.f` and `binpop.F` (KZ(20)=2 and KZ(8)=1, respectively) for 200 initial hard binaries. In the package of the code, the file `out1000.binary.comment` is included. It was created from this input file running for 20 time units. Stellar evolution was also switched on in this file (KZ(12)=1, KZ(19)=3, see below).

```
1 14400.0 1.e6 40 40
1000 1 10 1042 100 1
0.01 0.02 0.3 10.0 10.0 20.0 2.0E-05 1.0 0.7
1 1 1 0 1 4 4 1 0 2
1 1 0 0 2 1 0 0 3 2
1 0 2 0 0 2 0 0 0 2
0 0 2 0 1 0 1 1 0 1
0 1 2 0 0 0 0 0 0 0
1.0E-04 0.01 0.1 1.0 1.0E-06 0.01
2.3 20.0 0.1 200 2.e-2 0.0
0.5 0.0 0.0 0.0
200 0.005 -1.0 1.0 5.0 5 0
```

Stellar Evolution

Stellar evolution is invoked by KZ(19) = 1 or 3, offering two different schemes. The simpler one is KZ(19)=1, while the more complex one, K(19)=3, is based on the Cambridge stellar evolution school (Hurley, Pols, Tout 2000). Binaries are evolved as single stars without perturbing each other, any more complex binary evolution is not (yet) supported in NBODY6++. The main effects are changing stellar masses, radii, and luminosities, which give rise to cluster mass loss. The mass is assumed to escape from the cluster immediately and possible collisions depend on stellar radii.

With the additional option KZ(12)>0, information on binaries and single stars is written on two files (unit 82, file `bev.82` and unit 83, file `sev.83`) in regular time intervals determined by TPLOT. The data for each star in unit 83 comprise of NAME(I), KW, RI, M1, ZL1, which are name of star, stellar type (see `define.f`), distance to density centre scaled with core radius, mass of the star, logarithmic luminosity, and logarithmic radius, respectively.

5 Thresholds for the variables

Before the compilation of the code (Chapter 3), the parameter file (`params.h`) should be consulted to check whether some vector dimensions are in the desired range. Most important are

- the maximum particle number `NMAX`,
- the maximum number of regularised KS pairs `KMAX`, and
- the maximum number of neighbours per particle `LMAX`.

The particles are saved in various lists which serve to distinguish between their funcionality. The table below and Figure **??** (**Figure not finished!!!**) explain their nomenclature. “KS–pairs” are particles that approach each other in a hyperbolic encounter; they are given a special treatment by the code (see Chapter 11). If NPAIRS is the amount of KS–pairs, then IFIRST = 2*NPAIRS + 1 is the first single particle (not member of a KS pair), and N the last one. NTOT = N + NPAIRS is the total number of particles plus c.m.’s. Therefore NMAX, the dimension of all vectors containing particle data should be at least of size N + KMAX, where N is the number of particles and KMAX the maximum number of expected KS pairs. If one starts with single particles, KMAX = 10 or 20 should usually be enough, but in clusters with a large number of primordial binaries, KMAX must be large.

N:	Total number of particles
NBIN0:	number of primordial binaries (physical bound stars)
NBIN:	???
NPAIRS:	Number of binaries (KS–pairs, see Chapter **??**), transient unbound pairs as well as persistent binaries
NTOT:	= N + NPAIRS; Number of single particles plus centres of masses of regularized (KS) pairs
KMAX:	threshold for the amount of allowed KS pairs
NMAX:	= N + KMAX; threshold for the total number of particles and the centre of masses

Hier gibt’s noch ein Bildchen!

6 How to read the diagnostics

The diagnostics is the ASCII readable text printed on unit 6 *stdout* (“out1000” in Chapter 3) that gives a brief overview of the global status and progress of the cluster simulation. Different routines write into that file, depending on the options chosen as the input variables. The following lines occur:

```
   N  NFIX  NCRIT  NRAND  NNBOPT  NRUN
 1000     5     10   1006      50     1

  ETAI      ETAR      RS0       DTADJ     DELTAT     TCRITp    TCRIT     QE        RBAR       ZMBAR
  1.0E-02   2.0E-02   3.0E-01   1.0E+01   1.0E+01   1.0E+06   2.0E+01   2.0E-05   1.0E+00   7.0E-01

 OPTIONS
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
1  1  1  0  1  0  4  0  0  2  1  0  0  0  1  1  1  0  0  0  1  0  1  0  0  2  0  0  0  2  0  0  2  0  1  0  1  1  0  1

 OPTIONS BK:
1  2  3  4  5  6  7  8  9 10
0  0  0  0  0  0  0  0  0  0

 DTMIN     RMIN      ETAU      ECLOSE    GMIN      GMAX
 1.0E-04   1.0E-02   1.0E-01   1.0E+00   1.0E-06   1.0E-02
****** NOTE: new random number seed initialisation!
****** AND new ran2 from new ed. of Press et al.
```

written by the routine: `input.F`

Usage: Repetition of the input variables

```
STANDARD IMF    ALPHA = 2.35  BODY1 = 20.0  BODYN = 0.10 ZMASS = 3.36752E+02 NBIN0=    0 ZMET = 0.00 EPOCH0 = 0.00
.....................................................................................................
BINARY STAR IMF:   NB =  400  RANGE =  3.27E+01  2.14E-01 ZMB = 3.74E+02  <MB> = 9.36E-01
SINGLE STAR IMF:   NS = 1200  RANGE =  7.76E+00  1.01E-01 ZMS = 5.25E+02  <MS> = 4.38E-01
```

IMF power law index, max. mass, min. mass, total mass, # of primordial bin., metallicity, evolution. epoch [Myrs].
.....
number of objects, mass range, average mass before scaling.

`data.F`,
(if KZ(20)=0 & BODY1≠BODYN)

or

`imf2.F`, if KZ(20)≥2

Information about initial mass function (IMF).

```
          SCALING:   SX =  1.00421D+00  E = -2.49E-01  M(1) = 5.94E-02  M(N) = 2.97E-04  <M> = 1.00E-03

          TIME SCALES:   TRH = 2.8E+01  TCR = 2.8E+00  2<R>/<V> = 2.8E+00

     PHYSICAL SCALING:    R* = 1.0000E+00  M* = 7.0000E+02  V* = 1.7348E+00  T* = 5.6466E-01  <M> = 7.0000E-01
                          SU = 4.4335E+07  AU = 2.0627E+05  YRS = 3.5408E+06
```

`scale.F, units.f`

scaling factor for energy, total energy, max. mass, min. mass, average mass *after scaling*;
Spitzer's half-mass relaxation time, crossing time obtained from total energy and mass, crossing time obtained from virial radius (see 12);
information about physical scaling: values of one N–body unit in length (pc), mass (solar masses), velocity (km/s), time (million years), average mass of particles (solar massses), astronomical units (one N–body unit) and years (one N–body unit).

```
fpoly1 time=  0.1200000035762785
fpoly2 time=  0.2100000062584875
```

`start.F`

CPU (wall clock in parallel execution) time for initialising the force and its time derivative (`fpoly1, fpoly_mpi.f`) and the second and third time derivative of the force (`fpoly2, fpoly2_mpi.f`). The mpi-versions are called for initialisation in case of parallel runs.

`lagr.F`

```
   TIME  M/MT:    1.00D-02  2.00D-02  5.00D-02  1.00D-01  2.00D-01  3.00D-01  4.00D-01  5.00D-01  7.00D-01  9.00D-01  1.00D+00 <RC
     0.0 RLAGR:   1.52D-01  1.81D-01  1.91D-01  2.83D-01  4.33D-01  5.22D-01  6.17D-01  7.52D-01  1.16D+00  1.96D+00  5.86D+00  2.97D-01
     0.0 AVMASS:  6.26D-04  4.26D-03  6.13D-04  9.45D-04  7.82D-04  1.11D-03  1.09D-03  8.35D-04  9.19D-04  1.25D-03  9.01D-04  1.23D-03
     0.0 NPARTC:        17         9         2        53       130        90        91       122       216       163       107        90
     0.0 SIGR2:   2.19D-01  1.70D-01  7.37D-01  2.44D-01  1.89D-01  2.33D-01  2.16D-01  2.43D-01  1.72D-01  8.36D-02  5.09D-02  2.09D-01
     0.0 SIGT2:   2.01D-01  8.25D-02  4.46D-02  2.12D-01  3.39D-01  2.42D-01  1.71D-01  1.83D-01  1.51D-01  1.03D-01  6.16D-02  1.63D-01
     0.0 VROT:   -8.34D-02  4.41D-01 -6.28D-01  3.14D-02 -1.54D-01 -9.44D-02 -6.45D-02  1.17D-02  5.35D-02  1.98D-02 -3.46D-02  4.46D-01
```

Time, specification of the Lagrangian radii, core radius
Time, Lagrangian radii, core radius (if primordial binaries: separately for singles and binaries, not shown above)
Time, average mass between Lagrangian radii, avmass in the core
Time, number of particles within the shell, in the core
Time, radial velocity dispersion within the shell, in the core
Time, tangential vel. dispersion within the shell, in the core
Time, rotational vel. within the shell, in the core (not shown above)

```
0 ADJUST:  TIME =    1.00000D+01  T[Myr] =     5.65  Q = 0.52  DE =  -1.403819E-05  E =  -2.500038E-01 EBIN=   0.000000E+00 EMERGE=   0.000000E+00
```

adjust.F

rank, “ADJUST:”, total time in NB units, physical time, virial ratio, relative energy error, total energy, total energy of regularized pairs, energy of mergers

---

```
RMIN = 1.1E-03  DTMIN = 3.5E-05 RHOM = 3.5E+02 RSCALE = 9.5E-01 RSMIN = 2.2E-01  ECLOSE = 1.05  TC =    3
PE  N       ttot      treg      tirr      tpredtot  tint      tinit      tks      ttcomm    tadj     tmov      tprednb  tsub     tsub2    xtsub1   xtsub2
 0  1000     41.46000     29.54      7.23      0.63     40.39      0.99      0.07      0.00      0.59      0.00      1.50      0.00      0.00  0.00000D+00  0.00000D+00
```

close encounter distance and minimum time step (for regularization search, updated from input parameters if KZ(16)=1), maximum density, virial radius, minimum neighbour sphere, hard binary threshold energy, total run time in units of initial crossing times

number of processors, number of particles, total processing time, total regular processing, total irregular processing, processing of prediction, time spent in `intgrt.F`, for initialisation, for KS integration, for communication, for adjust and energy check, for overhead of moving data in parallel runs, for neighbour predictions, for MPI communication after irregular (tsub) and regular (tsub2) blocks, number of bytes transferred respectively. From xtsub1/tsub and xtsub2/tsub2 the sustained bandwidth of MPI communication can be read off. Note, that the determination of these quantities involves a certain overhead by many calls of `cputim.F` per block, so for critically large production runs one may want to comment these out (most of them in `intgrt.F`).

adjust.F

```
 0 T =   10.0  N = 1000  <NB> = 20  KS =    0  NM = 0 MM = 0 NS =  1000 NSTEPS =    1610624       273    321696       1016  DE =  -0.140382E-04  E =       -0.250004

NRUN =  1  M# =  1  CPU = 6.91000E-01  TRC =  0.0  DMIN = 6.6E-05 6.6E-05 1.0E+02 1.0E+02  AMIN = 1.0E+02  RMAX = 0.0E+00  RSMIN = 0.22  NEFF =   128

   <R>  RTIDE  RDENS   RC      NC   MC   RHOD   RHOM  CMAX   <Cn>  Ir/R    UN    NP    RCM    VCM         AZ     EB/E   EM/E   TCR     T6
#1 0.95   9.5   0.21  0.08      5  0.073  159.   350.    5.  37.0  0.13     0     0    0.000  0.0000   0.006197  0.000  0.000  2.83     5

      NNPRED    NBCORR  NBFULL  NBVOID  NRCONV    NICONV  NBSMIN  NBDIS  NBDIS2  NCMDER  NBDER  NFAST  NBFAST    NBLOCK     NBPRED
#2     20204    294307       0      98    2664      9227    1576      0       0      33      0      0       0     58132    3045868

    NKSTRY  NKSREG  NKSHYP     NKSPER  NPRECT  NKSREF  NKSMOD  NTTRY  NTRIP  NQUAD  NCHAIN  NMERG  NSTEPT  NSTEPQ  NSTEPC    NBLCKR    NBFLUX
#3   14463      45      33          0       0       0       0      0      0      0       0      0       0       0       0     10333   1903963
```

output.F

time, actual particle number, average neighbour number, number of KS pairs, number of merged KS pairs, number of hierarchical subsystems, number of single stars, step numbers (irregular, irr. c.m., regular, KS), relative energy error since last output, total energy

several more lines uncommented here....

```
STEP I     0    3   63   91  154  220  160  133  109   44   19    4
STEP R     0    4   77  133  249  310  179   45    3
Max Speedup Irr:     4 3.76D+00    8 6.82D+00   16 1.14D+01   32 1.66D+01   64 2.15D+01  128 2.49D+01  256 2.66D+01  512 2.74D+01 1024 2.77D+01
Max Speedup Reg:     4 3.71D+00    8 6.69D+00   16 1.12D+01   32 1.67D+01   64 2.22D+01  128 2.62D+01  256 2.91D+01  512 3.04D+01 1024 3.11D+01
```

levels.f

histogram of distribution of irregular (STEP I), regular (STEP R)
If there are primordial binaries: step distribution of their internal step (not appearing here, STEP U, in physical time), statistics of parallel work for irr. and reg. steps, figures given are theoretical speedups for infinitely fast communication (limit of large block sizes)

```
        END RUN    TIME[Myr] =   11.29  TOFF/TIME/TTOT=      0.00000000     20.00000000     20.00000000  CPUTOT =    1.6  ERRTOT =-5.15000D-05  DETOT =-1.28197D-05

       0 INTEGRATION INTERVAL =   20.00    NIRR=    3237662 NIRRB=       1245 NREG=     779010 NKS=       4625

         PER TIME UNIT: NIRR= 1.61883D+05 NIRRB= 6.22500D+01 NREG= 3.89505D+04 NKS= 2.31250D+02
 Total CPU=   97.11000289410342
```

adjust.F

This is the regular end of a run giving: the integration time, total cumulative absolute and relative errors, cumulative number of regular, irregular, KS steps, the step numbers per time unit and the total CPU (wall clock for parallel) time in minutes.

To check a regular stop of the run, look at the end of the diagnostics first. If there are failures, the line "`CALCULATION HALTED`" appears and means that the energy conservation could not be guaranteed. A restart with smaller steps (ETAI, ETAR) and larger neighbour number NNBOPT may cure the problem, but not always; persistent problems should be reported to Rainer Spurzem.

The unix command on the output file, e.g.

```
homedir> grep ADJUST out1000
```

produces an overview of the accuracy (energy error at every DTADJ interval). It may show where problems originated; a restart from the last ADJUST before the error with smaller output intervals is one way to look after it. Watch out, because sometimes errors are not reproducible, because changes in ADJUST intervals change frequencies of prediction and small differences can build up. A quick possibility to see the real evolution of the system is to `grep` for the lines with Lagrangian radii and other quantities (see above), which can directly be plotted, e.g. with gnuplot, because the first column is always the time.

7 Runs on parallel machines

For parallel runs, the file `mpif.h` is very important, and system specialists should be consulted in addition to us what to use. Again, for some standard systems templates are provided (e.g. `mpif.t3e.h` or `mpif.mpich.h`). The routine providing CPU–time measurements, `cputim.F` , and the use of the function `flush.f` may need special attention depending on the hardware.

The `Makefile` contains support for several common parallel computer architectures, such as PC Beowulf clusters using the MPICH communication library, CRAY T3E systems, Sun systems, or parallel IBM clusters using the load leveler system. A brief comment details this on top of the `Makefile` in some comment lines.

For an example, on a PC Beowulf cluster one would usually compile by `make mpich` and run the code by `mpirun -hostfile machines -np` *n* `./nbody6 < input > output &` where n is the number of processors, `machines` a file containing the list of accessible nodes, input and output your NBODY files. It has been assumed that the executable file `nbody6` has been copied before, after issuing the `make mpich` command, into the working directory of the run. Details are strongly dependent on your local system configuration. For example, a different flavour of mpirun command (`mpirun.rsh`) has to be used, also you have to make sure that your environment variables point to the correct place, such that `mpirun` and `mpif77` (used in `make mpich`) point to the correct system location, e.g. that where the correct network hardware (Infiniband) is used. Check with your sysadmin if unsure. If your local cluster runs a batch system (e.g. PBS. portable batch system, or IBM load leveller) you have to write and submit a batch job file, which typically contains some resource specifications in a special form and among other unix shell commands also the above `mpirun` command.

8 The Hermite integration method

Each particle is completely specified by its mass m, position $\mathbf{r}_0$, and velocity $\mathbf{v}_0$, where the subscript 0 denotes an initial value at a time t_0. The equation of motion for a particle i is given by its momentary acceleration $\mathbf{a}_{0,i}$ due to all other particles and its time derivative $\dot{\mathbf{a}}_{0,i}$ as

$$\mathbf{a}_{0,i} = -\sum_{i \neq j} G m_j \frac{\mathbf{R}}{R^3}, \tag{1}$$

$$\dot{\mathbf{a}}_{0,i} = -\sum_{i \neq j} G m_j \left[\frac{\mathbf{V}}{R^3} + \frac{3\mathbf{R}(\mathbf{V} \cdot \mathbf{R})}{R^5} \right], \tag{2}$$

where G is the gravitational constant; $\mathbf{R} = \mathbf{r}_{0,i} - \mathbf{r}_{0,j}$ is the relative coordinate; $R = |\mathbf{r}_{0,i} - \mathbf{r}_{0,j}|$ the modulus; and $\mathbf{V} = \mathbf{v}_{0,i} - \mathbf{v}_{0,j}$ the relative space velocity to the particle j.

The Hermite scheme employed in NBODY6++ follows the trajectory of the particle by firstly "predicting" a new position and new velocity for the next time step t. A Taylor series for $\mathbf{r}_i(t)$ and $\mathbf{v}_i(t)$ is formed:

$$\mathbf{r}_{p,i}(t) = \mathbf{r}_0 + \mathbf{v}_0(t - t_0) + \mathbf{a}_{0,i}\frac{(t - t_0)^2}{2} + \dot{\mathbf{a}}_{0,i}\frac{(t - t_0)^3}{6}, \tag{3}$$

$$\mathbf{v}_{p,i}(t) = \mathbf{v}_0 + \mathbf{a}_{0,i}(t - t_0) + \dot{\mathbf{a}}_{0,i}\frac{(t - t_0)^2}{2}. \tag{4}$$

The predicted values of $\mathbf{r}_p$ and $\mathbf{v}_p$, which result from this simple Taylor series evaluation, using the force and its time derivative at t_0, do not fulfil the requirements for an accurate high–order integrator; they just give a first approximation to $\mathbf{r}_1$ and $\mathbf{v}_1$ at the upcoming time t_1. Even if the time step, $t_1 - t_0$, is chosen impracticably small, a considerable error will quickly occur, let alone the inadequate computational effort. Therefore, an improvement is made by the Hermite interpolation which approximates the higher accelerating terms by another Taylor series:

$$\mathbf{a}_i(t) = \mathbf{a}_{0,i} + \dot{\mathbf{a}}_{0,i} \cdot (t - t_0) + \frac{1}{2}\mathbf{a}^{(2)}_{0,i} \cdot (t - t_0)^2 + \frac{1}{6}\mathbf{a}^{(3)}_{0,i} \cdot (t - t_0)^3, \tag{5}$$

$$\dot{\mathbf{a}}_i(t) = \dot{\mathbf{a}}_{0,i} + \mathbf{a}^{(2)}_{0,i} \cdot (t - t_0) + \frac{1}{2}\mathbf{a}^{(3)}_{0,i} \cdot (t - t_0)^2. \tag{6}$$

Here, the values of $\mathbf{a}_{0,i}$ and $\dot{\mathbf{a}}_{0,i}$ are already known, but a further derivation of equation (2) for the two missing orders on the right hand side turns out to be quite cumbersome. Instead, one determines the additional acceleration terms from the predicted ("provisional") $\mathbf{r}_p$ and $\mathbf{v}_p$; we calculate their acceleration and time derivative according to the equations (1) and (2) anew and call these new terms $\mathbf{a}_{p,i}$ and $\dot{\mathbf{a}}_{p,i}$, respectively. Because these values ought to be generated by the former high–order terms also (which we avoided), we put them into the left–hand sides of (5) and (6). Solving equation (6) for $\mathbf{a}^{(2)}_{0,i}$, then substituting it into (5) and simplifying yields the third derivative:

$$\mathbf{a}^{(3)}_{0,i} = 12\frac{\mathbf{a}_{0,i} - \mathbf{a}_{p,i}}{(t - t_0)^3} + 6\frac{\dot{\mathbf{a}}_{0,i} + \dot{\mathbf{a}}_{p,i}}{(t - t_0)^2}. \tag{7}$$

Similarly, substituting (7) into (5) gives the second derivative:

$$\mathbf{a}^{(2)}_{0,i} = -6\frac{\mathbf{a}_{0,i} - \mathbf{a}_{p,i}}{(t - t_0)^2} - 2\frac{2\dot{\mathbf{a}}_{0,i} + \dot{\mathbf{a}}_{p,i}}{t - t_0}. \tag{8}$$

Note, that the desired high–order accelerations are found just from the combination of the low–order terms for $\mathbf{r}_0$ and $\mathbf{r}_p$. We never derived higher than the first derivative, but achieved the higher orders easily through (1) and (2). This is called the Hermite scheme.

Previously, a four–step Adams–Bashforth–Moulton integrator was used (especially in NBODY5, [2]), however, the new Hermite scheme allows twice as large timesteps for the same accuracy. Also its storage requirements are less [16], [17], [4], [5].

Finally, we extend the Taylor series for $\mathbf{r}_i(t)$ and $\mathbf{v}_i(t)$, eqs. (3) and (4), by two more orders, and find the "corrected" position $\mathbf{r}_{1,i}$ and velocity $\mathbf{v}_{1,i}$ of the particle i at the computation time t_1 as

$$\mathbf{r}_{1,i}(t) = \mathbf{r}_{p,i}(t) + \mathbf{a}^{(2)}_{0,i}\frac{(t-t_0)^4}{24} + \mathbf{a}^{(3)}_{0,i}\frac{(t-t_0)^5}{120}, \tag{9}$$

$$\mathbf{v}_{1,i}(t) = \mathbf{v}_{p,i}(t) + \mathbf{a}^{(2)}_{0,i}\frac{(t-t_0)^3}{6} + \mathbf{a}^{(3)}_{0,i}\frac{(t-t_0)^4}{24}. \tag{10}$$

The integration cycle for other upcoming steps may now be repeated from the beginning, eqs. (1) and (2). The local error in $\mathbf{r}$ and $\mathbf{v}$ within the two time steps $\Delta t = t_1 - t_0$ is expected to be of order $\mathcal{O}(\Delta t^5)$, the global error for a fixed physical integration time scales with $\mathcal{O}(\Delta t^4)$ [15].

9 Individual and block time steps

Stellar systems are characterized by a huge dynamical range in radial and temporal scales. The time scale varies e.g. in a star cluster from orbital periods of binaries of some days up to the relaxation of a few hundred million years, or even billions of years. Even if we put for a moment the very close binaries aside, which are treated differently (by regularization methods), there typically is a large dynamic range in the average local stellar density from its centre to the very outskirts, where it dissolves into the galactic tidal field. In a classical picture, the two closest bodies would determine the time–step of force calculation for the whole rest of the system. However, for bodies in regions where the changes of the force are relatively small, a permanent re–computing of the terms appears time consuming. So, in order to economize the calculation, these objects shall be allowed to move a longer distance before a recomputation becomes necessary. In between there is always the possibility to acquire particle positions and velocities via a Taylor series prediction, as described in Chapter 8. This is the idea of a vital method for assigning different time–steps, $\Delta t = t_1 - t_0$, between the force computations, the so–called "individual time–step scheme" [1], which was later advanced to the hierarchical block steps.

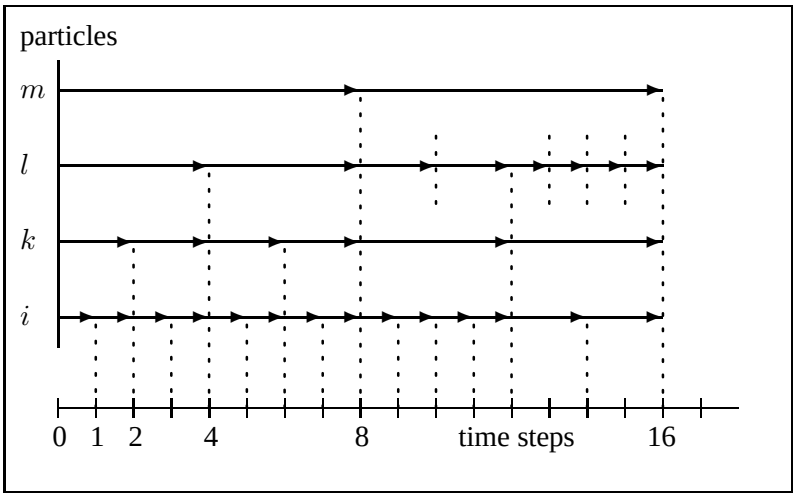


Figure 9.1: Block time steps exemplary for four particles.

Each particle is assigned its own Δt_i which is first illustrated for the case of "block time–steps" in Figure 9.1. The particle named i has the smallest time step at the beginning, so its phase space coordinates are determined at each time step. The time step of k is twice as large as i's, and its coordinates are just extrapolated ("predicted") at the odd time steps, while a full force calculation is due at the dotted times. The step width may be altered or not after the end of the integration cycle for the special particle, as demonstrated for k and l beyond the label "8". The time steps have to stay commensurable with both, each other as well as the total time, such that a hierarchy is guaranteed. This is the block step scheme.

As a first estimate, the rate of change of the acceleration seems to be a reasonable quantity for the choice of the time step: $\Delta t_i \propto \sqrt{\mathbf{a}_i/\dot{\mathbf{a}}_i}$. But it turns out that for special situations in a many-body system, it provides some undesired numerical errors. After some experimentation, the following formula was adopted [2]:

$$\Delta t_i = \sqrt{\eta \frac{|\mathbf{a}_{1,i}||\mathbf{a}^{(2)}_{1,i}| + |\dot{\mathbf{a}}_{1,i}|^2}{|\dot{\mathbf{a}}_{1,i}||\mathbf{a}^{(3)}_{1,i}| + |\mathbf{a}^{(2)}_{1,i}|^2}}, \tag{11}$$

where η is a dimensionless accuracy parameter which controls the error. In most applications it is taken to be $\eta \approx 0.01$ to 0.02, see also next chapter.

For the block–time steps, the synchronization is made by taking the next–lowest integer of Δt_i; the time steps are quantized to powers of 2 [15]. Then, there will be a group (block) of several particles which are due to movement at each time step. If one keeps the exact Δt_i's evaluated from (11) for each particle, the commensurability is destroyed, and we arrive at the so–called "individual time steps"; in this case, there exists one sole particle being due. The latter concept is realized in the earlier codes NBODY1, NBODY3, NBODY5, where a neighbour scheme is renounced. NBODY4, NBODY6, and NBODY6++ use a block step scheme.

Subsystems like star binaries, triples or a similar subgroups (they are termed KS pairs, chains, hierarchies) enter the time–step scheme with their respective centre's of masses only. Their internal motion is treated in a different way by a regularized integration (Chapter 11).

10 The Ahmad–Cohen scheme

The computation of the full force for each particle in the system makes simulations very time–consuming for large memberships. Therefore, it is desirable to construct a method in order to speed up the calculations while retaining the collisional approach. One way to achieve this is to employ a "neighbour scheme", suggested by [9].

The basic idea is to split the force polynomial (5) on a given particle i into two parts, an irregular and a regular component:

$$\mathbf{a}_i = \mathbf{a}_{i,\mathrm{irr}} + \mathbf{a}_{i,\mathrm{reg}}. \tag{12}$$

The irregular acceleration $\mathbf{a}_{i,\mathrm{irr}}$ results from particles in a certain neighbourhood of i (in the code, FI and FIDOT are the irregular force and its time derivative at the last irregular step; internally some routines use FIRR and FD as a local variable). They give rise to a stronger fluctuating gravitational force, so it is determined more frequently than the regular one of the more distant particles that do not change their relative distance to i so quickly (in the code, FR and FRDOT are the regular force and its time derivative at the last regular step; some routines use as a local variable FREG and FDR). We can replace the full summation in eq. (1) by a sum over the N_{nb} nearest particles for $\mathbf{a}_{i,\mathrm{irr}}$ and add a distant contribution from all the others. This contribution is updated using another Taylor series up to the order FRDOT, the time derivative of FR at the last regular force computation[1].

Wether a particle is a neighbour or not is determined by its distance; all members inside a specified sphere ("neighbour sphere" with radius r_{s}) are held in a list, which is modified at the end of each "regular time–step" when a total force summation is carried out. In addition, approaching particles within a surrounding shell satisfying $\mathbf{R} \cdot \mathbf{V} < 0$ are included. This "buffer zone" serves to identify fast approaching particles before they penetrate too far inside the neighbour sphere. The neighbour criterion should be improved according to relative forces rather than distances, in particular, if there are very strong mass differences between particles (black holes!) — such kind of work is under progress.

Figures 10.1 and 10.2 show how the Ahmad–Cohen scheme works for one particle [17]. At the beginning of the force calculation, a list of neighbour objects around the particle i is created first (filled dots). From this neighbour list the irregular component $\mathbf{a}_{i,\mathrm{irr}}$ is calculated, and then the summation is continued to the distant particles obtaining $\mathbf{a}_{i,\mathrm{reg}}$. At the same time we also calculate the first time derivative. From the equations (5) and (6) the position and velocity of the particle i are predicted. At time $t_{1,\mathrm{irr}}$ we apply the "corrector" only for $\mathbf{a}_{i,\mathrm{irr}}$ from the neighbours; the regular component we do not correct, but obtain by extrapolating $\mathbf{a}_{i,\mathrm{reg}}$. At the next step, $t_{2,\mathrm{irr}}$, the same predictor–corrector method proceeds for the neighbour particles, while the correction of the distant acceleration term is still neglected. When t_1 is reached, the total force is calculated on the basis of the full application of the Hermite predictor–corrector method. Also, a new neighbour list is constructed using the positions at time t_1. Thus, we calculate at certain times only the forces from neighbours (irregular time–step, t_{irr}), while at other times we calculate both the forces from neighbours and distant particles (regular time–step, t_{reg}).

For a neighbour list of size $N_{\mathrm{nb}} \ll N$, this procedure can lead to a significant gain in efficiency, provided the respective time scales for $\mathbf{a}_{i,\mathrm{irr}}$ and $\mathbf{a}_{i,\mathrm{reg}}$ are well separated.

[1] Note, that the code also keeps the variables F and FDOT, which contain one half (!) of the *total* force, and one sixth (!) of the *total* time derivative of the force; this just a handy assignment for the frequent predictions of equation 3.

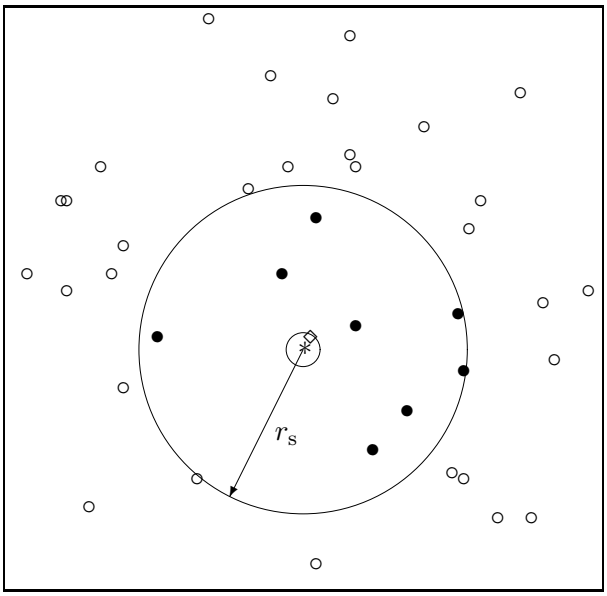


Figure 10.1: Illustration of the neighbour scheme for particle i marked as the asterisk (after [2]).

The actual size of neighbour spheres in NBODY6++ is controlled iteratively by a requirement in order to keep a certain optimal number of neighbours. This variable, NNBOPT, can be adjusted according to performance requirements. Its typical values are between 50 and 200 for a very wide range of total particle numbers N. Outside of the half-mass radius, the requirement of having NNBOPT neighbours is relaxed due to low local densities. Insisting on NNBOPT neigbours could result in undesired large amplitude fluctuations of the neighbour radii.

While [18] claim that the optimal neighbour number should grow as $N^{3/4}$ (which would be unsuitable for the performance on parallel computers), this is still an unsettled question. [2] advocates the coupling of the neighbour radius to the local density contrast, but NBODY6++ does *not* use that, since it makes average neighbour numbers much less predictable, which is bad for the performance and profiling issues on supercomputers, again.

Resuming, the method of the two particle groups is squeezed into the hierarchical time–step scheme making the overall view quite complex. Each particle is moved due to its time–step order *and* the time–steps, because the force calculation is divided: In eq. (11) a further subscript is needed which distinguishes the regular and irregular time step. The accuracy can be tuned by

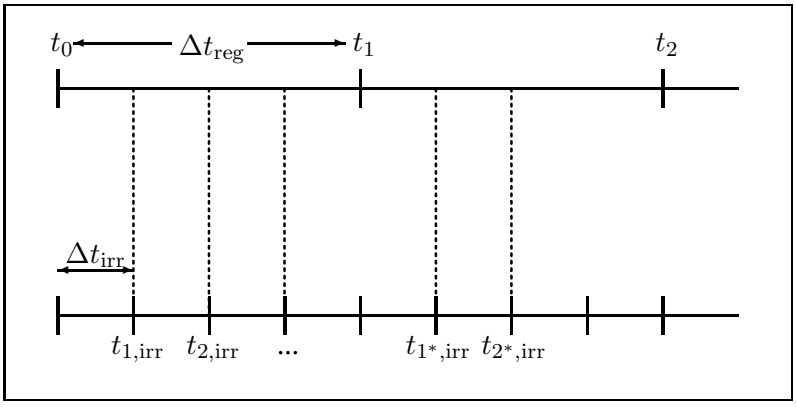


Figure 10.2: Regular and irregular time steps (after [17]).

$\eta_{\rm irr} \approx 0.01$ and $\eta_{\rm reg} \approx 0.02$, again.

Both, the neighbour scheme and the hierarchical time–step scheme have in common that they are centered on one particle i, and they distinguish between nearby and remote stars, and they save computational time. One may ask: What is the intriguing difference between them? — The neighbour scheme is a *spatial* hierarchy, which avoids a frequent force calculation of the remote particles, because their totality provides a smooth potential which does not vary so much with respect to the particle i; that potential is rather superposed by some fluctuating peaks of close–by stars which will be "worked in" by the more often force determination. The time step scheme, in contrast, exhibits the *temporal* behaviour of the intervals for re–calculation of the full force in order to maintain the exactness of the trajectory; time steps chosen too small slow down the advancing calculation losing the computer's efficiency.

11 KS–Regularization

The fourth main feature of the codes since NBODY3 is a special treatment of close binaries. A close encounter is characterised by an impact parameter that is smaller than the parameter for a 90 degree deflection

$$p_{90} = 2G(m_1 + m_2)/v_\infty^2 \tag{13}$$

where G, m_1, m_2, v_∞ are the gravitational constant, the masses of the two particles and their relative velocity at infinity. In the cluster centre, it is very likely that two (or even more) stars come very close together in a hyperbolic encounter. As the relative distance of the two bodies becomes small ($R \to 0$), their timesteps are reduced to prohibitively small values, and truncation errors grow due to the singularity in the gravitational potential, eqs. (1) and (2). In the NBODY code, the parameter RMIN is used to define a close encounter, and it is kept to the value of equation 13 (if KZ(16) > 0 is chosen in the control parameters). The corresponding time step DTMIN can be estimated from

$$dt_{\min} = \kappa \Big[\frac{\eta}{0.03}\Big] \Big(\frac{r_{\min}^3}{\langle m \rangle}\Big)^{1/2} \tag{14}$$

where κ is a free numerical factor, η the general time step factor, and $\langle m \rangle$ the average stellar mass [2]. If two particles are getting closer to each other than RMIN, and their time steps getting smaller than DTMIN, then they are candidates for "regularization".

Regularization is an elegant trick in order to deal with such particles which are as close as the diamond in the Figure 10.1. The idea is to take both stars out of the main integration cycle, replace them by their centre of mass (c.m.) and advance the usual integration with this composite particle instead of resolving the two components. The two members of the regularized pair (henceforth KS pair) will be relocated to the beginning of all vectors containing particle data, while at the end one additional c.m. particle is created (see below). One of the purposes of the code variable NAME(I) is to identify particles after such a reshuffling of data.

To be actually regularized, the two particles have to fulfil two more sufficient criteria: that they are approaching each other, and that their mutual force is dominant. In the equations in routine `search.f`, these sufficient criteria are defined as

$$\mathbf{R} \cdot \mathbf{V} > 0.1\sqrt{(G(m_1 + m_2)R}$$
$$\gamma := \tfrac{|\mathbf{a}_{\rm pert}| \cdot R^2}{G(m_1+m_2)} < 0.25$$

Here, $\mathbf{a}_{\rm pert}$ is the vectorial differential force exerted by other perturbing particles onto the two candidates, R, $\mathbf{R}$, $\mathbf{V}$ are scalar and vectorial distance and relative velocity vector between the two candidate, respectively. The factor 0.1 in the upper equation allows nearly circular orbits to be regularized; $\gamma < 0.25$ demands that the relative strength of the perturbing forces to the pairwise force is one quarter of the maximum. These conditions describe quantitatively that a two-body subsystem is dynamically separated from the rest of the system, but not unperturbed.

The internal motion of a KS pair will be determined by switching to a different (regularized) coordinate system. This transformation can be traced back to the square in quaternion space, where — by sacrificing some commutativity rules — it is guaranteed that the real-space motion does not leave the three-dimensional Cartesian space. It involves a set of four regular spatial coordinates and a fictitious time $s(t)$, obtained in its simplest variant by the transformation $dt = Rds$. Any unperturbed two–body orbit in real space is mapped onto a harmonic oscillator in KS–space with double the frequency. Since the harmonic potential is regular, numerical integration with high accuracy can proceed with much better efficiency, and there is no danger of truncation errors for arbitrarily small separations. The internal time–step of such a KS–regularized pair is independent

of the eccentricity and, depending on the parameter ETAU, of the order of some 50–100 steps per orbit. The method of regularization goes back to [14] and makes an accurate calculation of a perturbed two–body motion possible. A modern theoretical approach to this subject can be found in [25]; the Hamiltonian formalism of the underlying transformations is nicely explained in [20].

While regularization can be used for any analytical two–body solution across a mathematical collision, it is practically applied to perturbed pairs only. Once the perturbation γ falls below a critical value (input parameter GMIN $\approx 10^{-6}$), a KS–pair is considered unperturbed, and the analytical solution for the Keplerian orbit is used instead of doing numerical integration. A little bit misleading is that such unperturbed KS–pairs are denoted in the code as "mergers", e.g. in the number or merges (NM) and the energy of the mergers (EMERGE). Merged pairs can be resolved at any time if the perturbation changes. The two–body KS regularization occurs in the code either for short-lived hyperbolic encounters or for persistent binaries.

In the code, the KS–pair appears as a new particle at the postion of the centre of mass. The variable NTOT, that contains the total number of particles N plus the c.m.'s, is increased by 1. When the pair is disrupted, NTOT is decreased again. The maximum number of possible KS–pairs is saved in the variable KMAX, which sets a threshold for the extension of the vector NTOT (see Chapter 5).

Close encounters between single particles and binary stars are also a central feature of cluster dynamics. Such temporary triple systems often reveal irregular motions, ranging from just a perturbed encounter to a very complex interaction, in which disruption of binaries, exchange of components and ejection of one star may occur. Although not analytically solvable, the general three–body problem has received much attention. So, the KS–regularization was expanded to the isolated 3– and 4–body problem, and later on to the perturbed 3–, 4–, and finally to the N–body problem. The routines are called

- `triple.f` (unperturbed 3-body subsystems, [8]),
- `quad.f` (unperturbed 4-body subsystems), and
- `chain.f` with different stages of implementation (slow-down, Stumpff functions, see for consecutive references Mikkola & Aarseth 1990, 1993, 1996, 1998, and [20]).

While occurrences of "triple" and "quad" will be rare in a simulation, the chain regularization is invoked if a KS–pair has a close encounter with another single star or another pair. Especially, if systems start with a large number of primordial binaries, such encounters may lead to stable (or quasi-stable) hierarchical triples, quadruples, and higher multiples. They have to be treated by using special stability criteria. Some of them are actually already implemented, but there is ongoing research and development in the field.

A typical way to treat all such special higher subsystems is to define their c.m. to be a pseudo-particle, i.e. a particle with a known sub-structure (very much like nodes in a TREE code). The members of the pseudo-particles will be deactivated by setting their mass to zero (ghost particles). At present there can only exist one chain at a time in the code, while merged KS binaries, and hierarchical subsystems can be more frequent. Details of these procedures are beyond the scope of this introductory manual.

Every subsystem — KS pair, chain or hierarchical subsystem — is perturbed. Perturbers are typically those objects that get closer to the object than $R_{\rm sep} = R/\gamma_{\rm min}^{1/3}$, where R is the typical size of the subsystem; for perturbers, the components of the subsystem are resolved in their own force computation as well (routines `cmfreg.f`, `cmfirr.f`).

12 Nbody–units

The NBODY–code uses Dimensionless units, so–called "Nbody units". They are obtained when setting the gravitational constant G and the initial total cluster mass M equal to 1, and the initial total energy E to $-1/4$ (see [12], [7]).

Since the total energy E of the system is $E = K + W$ with $K = \frac{1}{2}M\langle v^2\rangle$ being the total kinetic energy and $W = -(3\pi/32)GM^2/R$ the potential energy of the Plummer sphere, we find from the virial theorem that

$$E = \frac{1}{2}W = -\frac{3\pi}{64}\frac{GM^2}{R}. \tag{15}$$

R is a quantity which determines the length scale of a Plummer sphere. Using the specific definitions for G, M, and E above, this scaling radius becomes $R = 3\pi/16$ in dimensionless units. The half mass radius $r_{\rm h}$ can easily be evaluated by the formula (e.g. [26]):

$$M(r) = M\frac{r^3/R^3}{(1 + r^2/R^2)^{3/2}} \tag{16}$$

when setting $M(r_{\rm h}) = \frac{1}{2}M$. It yields $r_{\rm h} = (2^{2/3} - 1)^{-1/2}R = 1.30\,R$. The half–mass radius is located at $R = 0.766$, or about 3/4 "Nbody–radii".

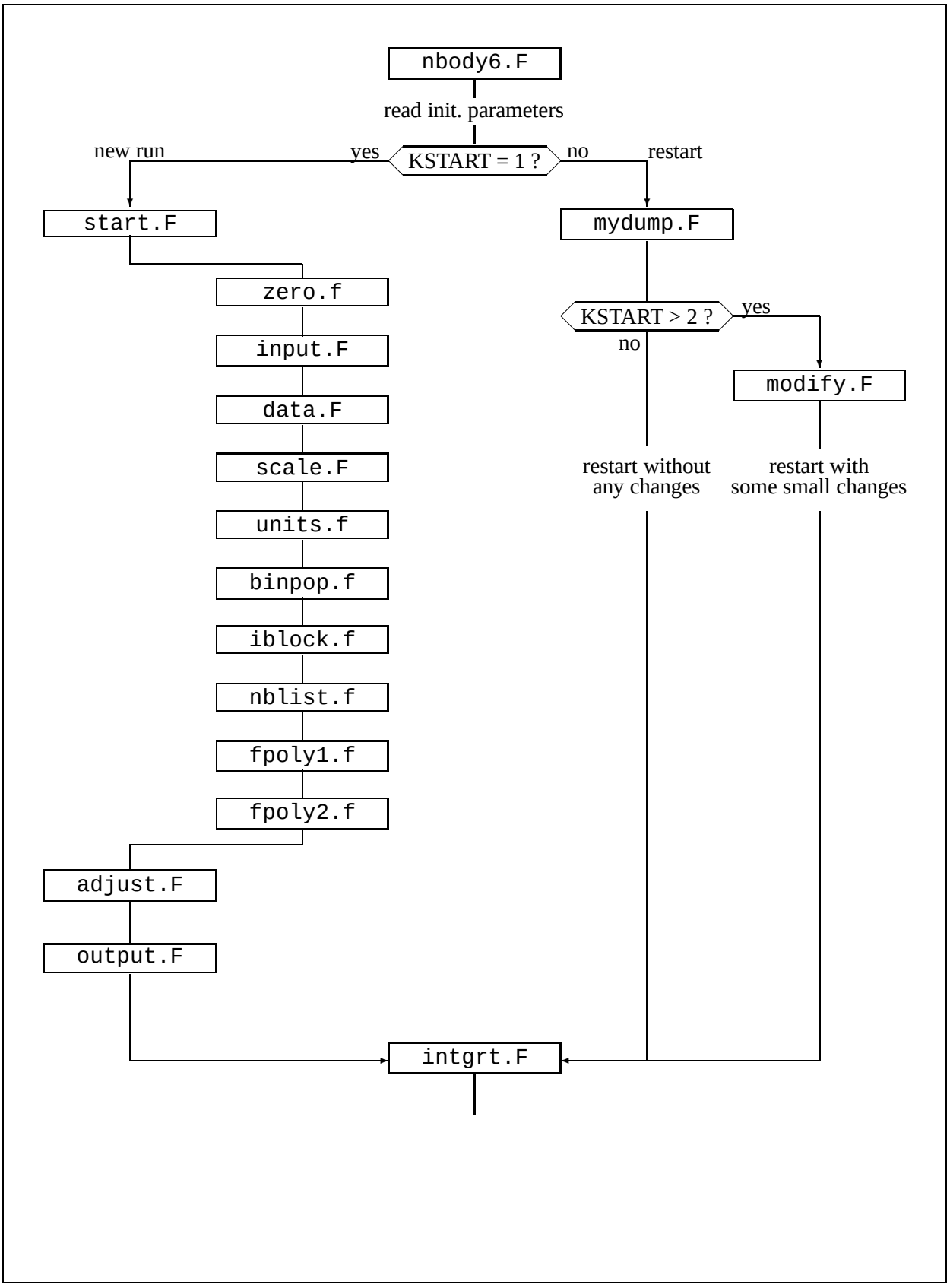
The virial radius of a system is defined by $R_{\rm vir} = GM^2/2|W|$, while the r.m.s. velocity is $\langle v^2\rangle^{1/2} = 2K/M$. In virial equilibrium $|W| = 2K$, so it follows for the crossing time

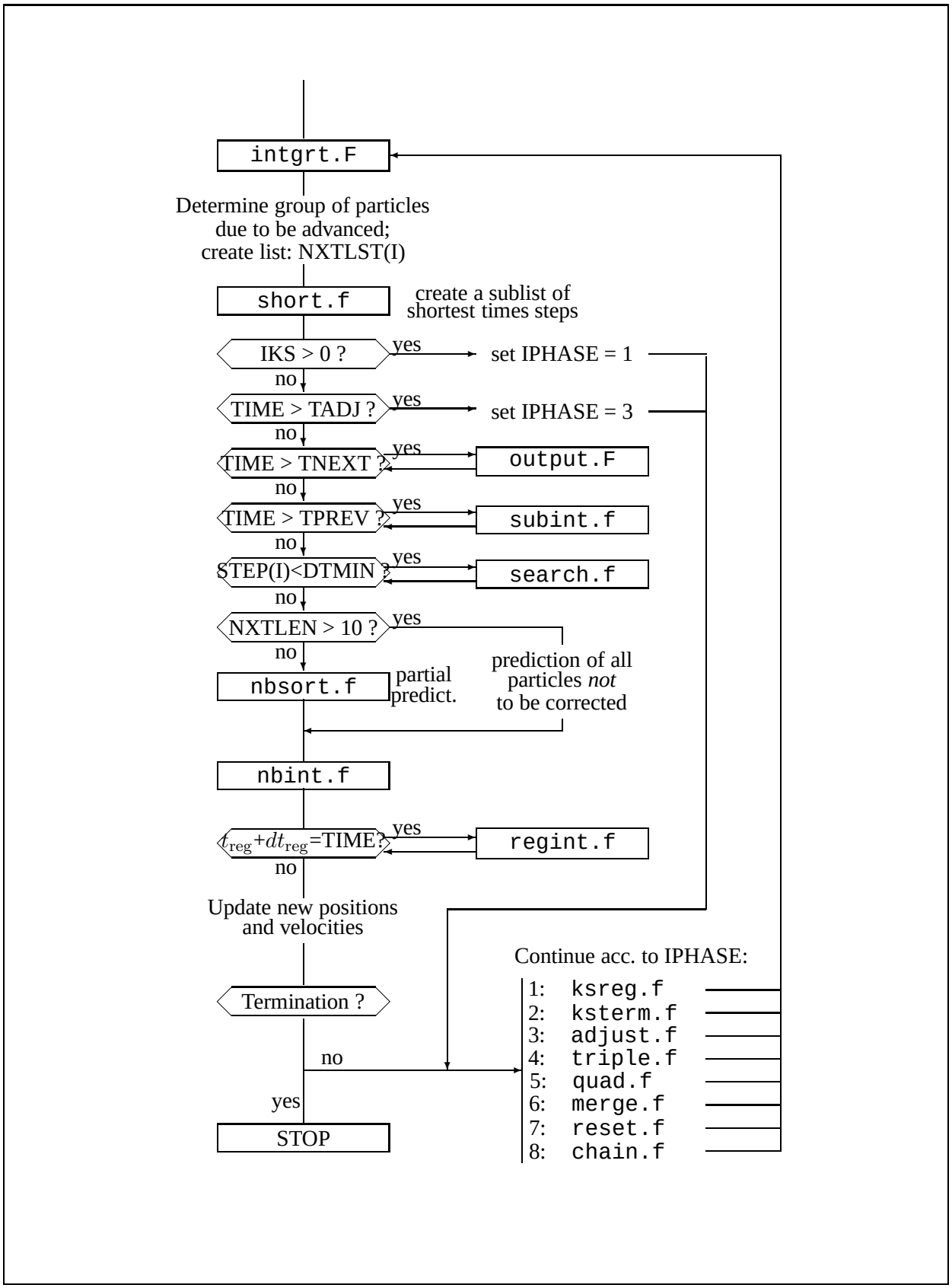
$$t_{\rm cr} := \frac{2R_{\rm vir}}{\langle v^2\rangle^{1/2}} = \frac{GM^{5/2}}{(2|E|)^{3/2}}. \tag{17}$$

The setting of $G = M = 1$ and $E = -0.25$ also determines the unit of time; so it follows that $t_{\rm cr} = 2\sqrt{2}$ in N-body units. By inversion we have

$$\tau_{\rm NB} = \frac{GM^{5/2}}{(4|E|)^{3/2}}, \tag{18}$$

for the unit of time $\tau_{\rm NB}$. The virial radius of Plummer's model is $R_{\rm vir} = 1$ in N-body units.


nbody6.F
read init. parameters
new run
yes
KSTART = 1 ?
no
restart
start.F
mydump.F
zero.f
KSTART > 2 ?
yes
no
input.F
modify.F
data.F
scale.F
restart without
any changes
restart with
some small changes
units.f
binpop.f
iblock.f
nblist.f
fpoly1.f
fpoly2.f
adjust.F
output.F
intgrt.F


intgrt.F
Determine group of particles due to be advanced; create list: NXTLST(I)
short.f
create a sublist of shortest times steps
IKS > 0 ?
yes
set IPHASE = 1
no
TIME > TADJ ?
yes
set IPHASE = 3
no
TIME > TNEXT ?
yes
output.F
no
TIME > TPREV ?
yes
subint.f
no
STEP(I)<DTMIN ?
yes
search.f
no
NXTLEN > 10 ?
yes
no
nbsort.f
partial predict.
prediction of all particles *not* to be corrected
nbint.f
$t_{reg}+dt_{reg}$=TIME?
yes
regint.f
no
Update new positions and velocities
Continue acc. to IPHASE:
Termination ?
1: ksreg.f
2: ksterm.f
3: adjust.f
4: triple.f
5: quad.f
6: merge.f
7: reset.f
8: chain.f
no
yes
STOP

Literaturverzeichnis

[1] **Aarseth S.J. (1963)**: *Mon. Not. Roy. Astron. Soc. 126*, p223

[2] **Aarseth S.J. (1985)**: "Direct methods for N–body simulations", in: *Multiple Time Scales*, Brackbill J. & Cohen B. (eds.), Ch. 12, p377

[3] **Aarseth S.J. (1993)**: "Direct methods for N–body simulations", in: *Galactic Dynamics and N– body simulations*, Contopoulos G. et. al. (eds.), Symposium in Thessaloniki, Greece

[4] **Aarseth S.J. (1999a)**: *Publ. Astron. Soc. Pac. 111*, p1333

[5] **Aarseth S.J. (1999b)**: *Celect. Mech. Dyn. Astron. 73*, p127

[6] **Aarseth S.J. (2003)**: "Gravitational N–Body Simulations, Tools and Algorithms", *Cambridge University Press*, 430 pages, ISBN 0521432723

[7] **Aarseth S.J., Hénon M., Wielen R. (1974)**: *Astron. Astrophys. 37*, p183

[8] **Aarseth S.J., Zare . (1974)**: *Celest. Mech. 10*, p185

[9] **Ahmad A. & Cohen L. (1973)**: *J. Comput. Phys. 12*, p389

[10] **Cohn H. (1980)**: *ApJ 242*, p765

[11] **Dorband E.N., Hemsendorf M., Merritt D. (2003)**: *J. Comput. Phys. 185*, p484–511

[12] **Heggie D.C. & Mathieu R.D. (1986)**: "Standardised units and time scales", in: *The Use of Supercomputers in Stellar Dynamics*, Hut P. & McMillan S. (eds.), p233

[13] **Hénon M. (1971)**: *Astrophys. Space Sci. 14*, p151

[14] **Kustaanheimo P. & Stiefel E.L. (1965)**: *J. für Reine Angewandte Mathematik 218*, p204

[15] **Makino J. (1991a)**: *Publ. Astron. Soc. Japan 43*, p859

[16] **Makino J. (1991b)**: *ApJ 369*, p200

[17] **Makino J. & Aarseth S.J. (1992)**: *Publ. Astron. Soc. Japan 44*, p141

[18] **Makino J. & Hut, P. (1988)**: *ApJ Suppl. 68*, p833

[19] **Makino J., Taiji M., Ebisuzaki T., Sugimoto D. (1997)**: *ApJ 480*, p432–446

[20] **Mikkola S. (1997)**: "Numerical Treatment of Small Stellar Systems with Binaries", in: *Visual Double Stars: Formation, Dynamics and Evolutionary Tracks*, Docobo J.A., Elipe A., & McAlister H. (eds.), p269

[21] **Mikkola S. & Aarseth, S.J. (1990)**: *Celest. Mech. Dyn. Ast. 47*, p375

[22] **Mikkola S. & Aarseth, S.J. (1993)**: *Celest. Mech. Dyn. Ast. 57*, p439

[23] **Mikkola S. & Aarseth, S.J. (1996)**: *Celest. Mech. Dyn. Ast. 64*, p197

[24] **Mikkola S. & Aarseth, S.J. (1998)**: *New Astronomy 3*, p309

[25] **Neutsch W. & Scherer K. (1992)**: “Celestial Mechanics”, *Bibliographisches Institut*, 484 pages, ISBN 3411154810

[26] **Spitzer L. jr. (1987)**: *Dynamical Evolution of Globular Clusters*, Princeton University Press, New Jersey, USA

[27] **Spurzem R. (1994)**: “Gravothermal Oscillations”, in: *Ergodic Concepts in Stellar Dynamics*, Gurzadyan V.G. & Pfenniger D. (eds.), p170

[28] **Spurzem R. (1999)**: *J. Comput. Appl. Math. 109*, p407

[29] **von Hoerner S. (1960)**: *Zeitschrift f. Astrophysik 50*, p184–214

[30] **von Hoerner S. (1963)**: *Zeitschrift f. Astrophysik 57*, p47–82