User Guide for falcON

version of May 21, 2004

Summary. falcON is the "Force Algorithm with Complexity $\mathcal{O}(N)$)" which is described by Dehnen (2000, 2002). With this packages, you can use falcON in subroutine form as Poisson solver for particle based simulations. The package also has a full N-body code, based on falcON, called gyrfalcON ("GalaxY simulatoR using falcON"), which employs the N-body tool box NEMO. This code features individual adaptive time steps employing a block-step scheme, but can also be used in single-time-step mode (in which case momentum is exactly conserved). Additionally, there are several other programs and facilities that may prove useful for setting-up, running, and analyzing N-body simulations.

1 Guarantee

This package comes with absolutely no guarantee whatsoever! The unpacking, installation, and usage of the code is entirely at the risk of the user alone.

2 Credit

Any scientific publication or presentation which has benefited from using falcON in subroutine form or from using any of the programs gyrfalcON, getgravity, or addgravity should quote the papers

Dehnen, W., 2000, ApJ, 536, L39

Dehnen, W., 2002, JCP, 179, 27.

(please find pdf file of the latter paper in the subdirectory falcON/doc.) Papers that did not use these but other parts of this packages should acknowledge that whereby explicitly mentioning me (Walter Dehnen) as author of the code.

3 What is new?

This section has been added to the user guide in 2004; earlier changes are not all reflected here.

September 2003

Individual, but fixed, softening lengths have been added to the public version. See §5 below.

April 2004

We now use a dynamic library (libfalcON.so) instead of (libfalcON.a), so that you must put the directory it resides in into the LD_LIBRARY_PATH enviroment variable, otherwise, the code will not work; see also §4.

14$^{\rm th}$ May 2004

The NEMO programs mkdehnen, mkking, and mkplum have been added to the public version of this package, see §11.2 for details.

19$^{\rm th}$ May 2004

All NEMO programs in this packages come with man pages, which replace the detailed documentation in this file.

4 Unpacking & Installation

4.1 Unpacking

After downloading the file `falcON.tgz`, unpack it typing

```
tar zxf falcON.tgz,
```

which should create the directory `falcON` with sub-directories `src`, `inc`, `doc`, and `man` and several other files.

4.2 On Compiler Issues

You need to make the library `libfalcON.so` and, possibly, the executables you want to use, see §§ below. The code is written entirely in C++ and it is strongly recommended to use a compiler that understands standard C++, I recommend GNU's gcc (version 3.2 or later) or INTEL's icc (version 8.0 or later). By default, we use gcc, if you want to use another compiler, edit the file `make.defs` and change the entry for `COMPILER`. `Makefile` is intended for use with GNU make. **Note** that using different compilers for `NEMO` and `falcON` may not work (using gcc for the former and icc for the latter appears not to work).

4.3 Make with `NEMO`

If you want to use the various `NEMO` programs in this packages, you must first start `NEMO` (usually by sourcing the file `nemo_start` in the `NEMO` directory). The various options for making are summarized below.

`make all`	makes the library and all executables
`make man`	copies man pages to `NEMO`
`make`	same as `make all man`
`make tonemo`	same as `make all` plus copying executables and library into `NEMO`
`make install`	same as `make tonemo man`

The making of the library and executables takes a little while, but should not produce any warnings or error messages. Otherwise, something might be wrong – I would appreciate if you, in such a case, could email me the error messages together with details of the compiler and system used.

Finally after making type `rehash` to let the shell know about the new executables.

4.4 Make without `NEMO`

If you really do not want to use the `NEMO` executables and `NEMO` features of `falcON`, i.e. if you want to use `falcON` merely as a subroutine to compute the forces in your code (see §9), then simply say `make` *without* having `NEMO` activated (i.e. `echo $NEMO` produces "`NEMO: Undefined variable`"). This will make the `falcON` library and the non-`NEMO` executables `TestGrav` and `TestPair`.

These executables live in a subdirectory

```
falcON/$MACHTYPE_$OSTYPE,
```

hereafter referred to *thedir*, where `MACHTYPE` and `OSTYPE` are environment variables unique to the machine type and operating system. In this way, you may have versions of the executables and the library (which is in subdirectory *thedir*`/lib`) for several hosts on the same file system.

In order to finish the installation, you must add the corresponding directories to the `PATH` and `LD_LIBRARY_PATH` enviroment variables. This is best done once and forever by adding the following lines to your `.cshrc` or `.tcshrc` file.

```
setenv FALCON                 "/usr/local/falcON"#  edit to reflect your actual directory
setenv FALCONBIN              $FALCON"/"$MACHTYPE"_"$OSTYPE
setenv FALCONLIB              $FALCONEXE"/lib"
setenv PATH                   $FALCONBIN"/:"$PATH
if ( $?LD_LIBRARY_PATH ) then
     setenv LD_LIBRARY_PATH   $FALCONLIB":"$LD_LIBRARY_PATH
else
     setenv LD_LIBRARY_PATH   $FALCONLIB
endif
```

5 Individual Softening Lengths

Individual softening lengths are enabled, but not obligatory (in fact default is always to have a globally constant ϵ), if line 21 of the `Makefile`

```
DSOFT                   := -DfalcON_INDI
```

is not commented out (by a # in the first column).

The softening length ϵ_{ij} used in the interaction of nodes with individual softening lengths ϵ_i and ϵ_j is simply the arithmetic mean of the two. The softening length ϵ_i of a cell is the arithmetic mean of the softening lengths of all its bodies.

6 Testing falcON

Please run `TestGrav` in order to get some rough check on the validity of your library. Issuing the command

```
TestGrav 2 1 1000000 901 0.01 1
```

shall generate a Hernquist sphere with $N = 10^6$ particles, build the tree (twice: once from scratch and once again) and compute the forces using a softening length of $\epsilon = 0.01$ scale radii with the P_1 kernel (see §7). The output of this command may look like[1]

```
time needed for set up of X_i:                 1.19
time needed for falcON::grow():                1.8
time needed for falcON::grow():                1
time needed for falcON::approximate_gravity():  9.31

state:                 tree re-grown
root center:           0 0 0
root radius:           1024
bodies loaded:         1000000
total mass:            1
N_crit:                6
cells used:            353419
of which were active   353419
maximum depth:         21
current theta:         0.6
current MAC:           theta(M)
softening:             global
softening length:      0.01
softening kernel:      P1
Taylor coeffs used:    84282 in 4 chunks of 22092
interaction statitics:
    type          approx   direct      total
# body-body :          -        0          0 =        0%
# cell-body :    2125639   479315    2604954 =   18.325%
# cell-cell :   11301375   254998   11556373 =   81.297%
# cell-self :          -    53678      53678 =    0.378%
# total     :   13427014   787991   14215005 =  100.000%

ASE(F)/<F^2>     = 0.001597663853
max (dF)^2       = 0.8176034689
Sum m_i acc_i    = -6.636294292e-10 1.564050334e-09 7.802109514e-10
```

Note that the second tree-build is much faster then the initial one. Note also the the total-momentum change (last line) vanishes within floating point accuracy – that's a generic feature of `falcON`.

7 Choice of the Softening Kernel and Length

The code allows for various forms of the softening kernel, i.e. the function by which Newton's $1/r$ is replaced in order to avoid diverging near-neighbour forces. The following kernel functions are available ($x := r/\epsilon$)

[1] Code compiled with `gcc` version 3.2.2, run on an AMD Athlon(TM) XP 1800+ with 1529Mhz and 256Kb cache size.

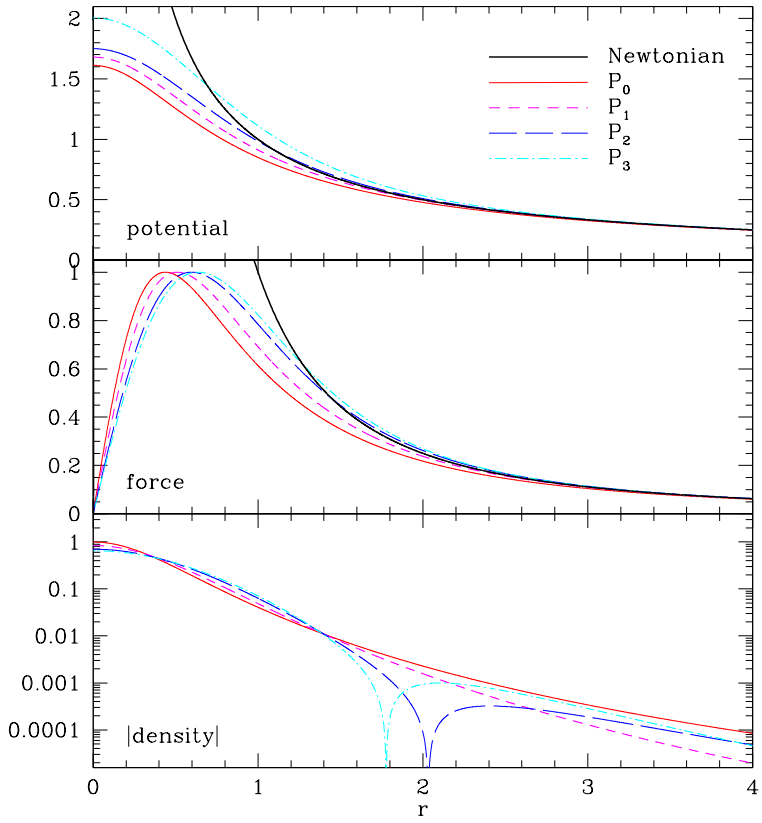

Figure 1: Potential, force, and density for the softening kernels of the table, including the standard Plummer softening (P_0). The softening lengths ϵ are scaled such that the maximum force equals unity. The kernels $P_{>0}$ approach Newtonian forces more quickly at larger r than does P_0. The kernels P_2 and P_3 have slightly super-Newtonian forces (and negative densities) in their outer parts, which compensate for the sub-Newtonian forces at small r.

name	density (is proportional to)	a_0	a_2	f
P_0	$(1+x^2)^{-5/2}$	∞	∞	1
P_1	$(1+x^2)^{-7/2}$	π	∞	1.43892
P_2	$7(1+x^2)^{-9/2} - 2(1+x^2)^{-7/2}$	0	∞	2.07244
P_3	$9(1+x^2)^{-11/2} - 4(1+x^2)^{-9/2}$	0	$-\pi/40$	2.56197

Note, that P_0 is the standard Plummer softening, however, **recommended** is the use of P_1 or P_2. There are several important issues one needs to know about these various kernels.

First, the softening length ϵ is just a parameter and using the same numerical value for it but different kernels corresponds in effect to different amounts of softening. Actually, this softening is strongest for the Plummer sphere: at fixed ϵ, the maximal force is smallest. In order to obtain comparable amounts of softening, larger ϵ are needed with all the other kernels. An idea of the factor by which ϵ has to be enlarged can be obtained by setting ϵ such that the maximum possible force between any two bodies are equal for various kernels. The last column in the previous table gives these factors. Note, that using a larger ϵ with other than the P_0 kernel does **not** mean that your resolution goes down, it in fact increases, see Dehnen (2001), but the Poisson noise is more suppressed with larger ϵ. It is recommended not to use Plummer softening, unless (i) you want $\epsilon \equiv 0$, (ii) in 2D simulations, as here ϵ is the average scale-height of the disk, and, perhaps, (iii) in simulations made to compare with others that use Plummer softening (for historical reasons).

Second, as shown in Dehnen (2001), Plummer softening results in a strong force bias, due to its slow convergence to the Newtonian force at $r \gg \epsilon$. This is quantified by the measure a_0, which for P_0 is infinite. In Dehnen (2001), I considered therefore other kernels (not mentioned above), which have finite support, ie.

the density is exactly zero for $r \geq \epsilon$. This discontinuity makes them less useful for the tree code (which is based on a Taylor expansion of the kernel). In order to overcome this difficulty, the kernels P_1 to P_3, which are continuous in all derivatives, have been designed as extensions to the Plummer softening, but with finite a_0 (P_1), zero a_0 but infinite a_2 (P_2), or even zero a_0 and finite a_2 (P_3).

8 Choice of the Tolerance Parameter

The code falcON approximates an interaction between two nodes, if their critical spheres don't overlap. The critical spheres are centered on the nodes' centers of mass and have radii

$$r_{\rm crit} = r_{\rm max}/\theta \tag{1}$$

where $r_{\rm max}$ is the radius of a sphere that is guaranteed to contain all bodies of the node (bodies have $r_{\rm max} = 0$), while θ is the tolerance parameter. The default is to use a mass-dependent $\theta = \theta(M)$ with $\theta_0 \equiv \theta(M_{\rm tot})$ being the parameter, see Dehnen (2002). For near-spherical systems or groups of such systems, θ_0 of 0.6 gives relative force errors of the order of 0.001, which is generally believed to be acceptable. However, the force error might often be dominated by discreteness noise, in which case a larger value does no harm. For disk systems, however, a smaller tolerance parameter, e.g. $\theta_0 = 0.5$, might be a better choice.

The recommendation is to either stick to θ_0 no larger than about 0.6, or perform some experiments with varying θ_0 (values larger than 0.8, however, make no sense, as there is hardly any speed-up).

9 Use of falcON as Poisson Solver in Your Code

You may use `falcON` like a subroutine in your existing code to serve as a Poisson solver for a particle distribution.

9.1 With C++

In order to make use of the code, you need to insert the `C` macro

```
#include <falcON.h>
```

somewhere at the beginning of your C++ source code. Make sure that the compiler finds the file `falcON.h` by including `-I falcON/inc` among your compiler options. The usage of the code in your application is explained in gory detail in the file `falcON.h` (don't forget that `class falcON` lives in namespace `nbdy`). In order to make an executable, add the linker options `-L`*thedir*`/lib -lfalcON -lm` so that the library will be loaded at runtime.

For examples of code using `falcON.h`, see the files `TestGrav.cc` and `TestPair.cc` in subdirectory `src/mains/`, which may be compiled by typing `make TestGrav` and `make TestPair` and produce a short summary of their usage when run without arguments.

9.2 With C

In order to make use of the code, you need to insert the `C` macro

```
#include <falcON_C.h>
```

somewhere at the beginning of your C source code. Make sure that the compiler finds the file `falcON_C.h` by including `-I falcON/inc` among your compiler options. The usage of the code in your application is explained in gory detail in the file `falcON_C.h`. In order to make an executable, add the linker options `-L`*thedir*`/lib -lfalcON -lstdc++ -lm` so that the library will be loaded at runtime.

For examples of code using `falcON_C.h`, see the files `TestGravC.cc` and `TestPairC.cc` in subdirectory `src/mains/`, which may be compiled by typing `make TestGravC` and `make TestPairC` and produce a short summary of their usage when run without arguments.

9.3 With FORTRAN

In order to make use of the code, you need to insert

```
INCLUDE 'falcON.f'
```

somewhere at the beginning of your FORTRAN program. Make sure that the compiler finds the file `falcON.f` by including `-I falcON/inc` among your compiler options. The usage of the code in your application is explained in gory detail in the file `falcON.f`. In order to make an executable, add the linker options `-L`*thedir*`/lib -lfalcON -lstdc++ -lm` so that the library will be loaded at runtime.

For examples of code using `falcON.f`, see the files `TestGravF.F` and `TestPairF.F` in subdirectory `src/mains/`, which may be compiled by typing `make TestGravF` and `make TestPairF`. Just run these programs, they are self-explanatory and provide some statistics output. You may also use the input files given and run them as `TestGravF < treeF.in` and `TestPairF < pairF.in`.

10 The *N*-Body Code gyrfalcON

The package also contains a full N-body code, called "`gyrfalcON`" (GalaxY simulatoR using `falcON`)[2]. If you want to use this code, you need first to install and invoke the N-body tool box `NEMO`, version 3.0.13 or higher[3], see http://www.astro.umd.edu/nemo. It is recommended to configure `NEMO` with `configure --enable-single --enable-lfs`. `gyrfalcON` comes with the usual `NEMO` help utility: calling

`gyrfalcON help=h` produces the following overview over the options.

```
in              : input file                                          [???]
out             : file for primary output; required, unless resume=t  []
tstop           : final integration time [default: never]             []
step            : time between primary outputs; 0 -> every step       [1]
logfile         : file for log output                                 [-]
stopfile        : stop simulation as soon as file exists              []
logstep         : # blocksteps between log outputs                    [1]
out2            : file for secondary output stream                    []
step2           : time between secondary outputs; 0 -> every step     [0]
theta           : tolerance parameter at M=M_tot                      [0.64]
hgrow           : grow fresh tree every 2^hgrow smallest steps        [0]
Ncrit           : max # bodies in un-split cells                      [16]
eps             : >=0: softening length
                  < 0: use individual fixed softening lengths         [0.05]
kernel          : softening kernel of family P_n (P_0=Plummer)        [1]
hmin            : tau_min = (1/2)^hmin                                [6]
Nlev            : # time-step levels                                  [1]
fac             : tau = fac / acc           \   If more than one of  []
fph             : tau = fph / pot            |  these is non-zero,   []
fpa             : tau = fpa * sqrt(pot)/acc  |  we use the minimum   []
fea             : tau = fea * sqrt(eps/acc) /   tau.                  []
resume          : resume old simulation?  that implies:
                  - read last snapshot from input file
                  - append primary output to input (unless out given) [f]
give            : list of output specifications. Recognizing:
                    m: mass                          (default)
                    x: position                      (default)
                    v: velocity                      (default)
                    a: acceleration
                    p: N-body potential
                    P: external Pot (added to pot before output)
                    e: individual eps_i (if they exist)
                    l: time-step level (if they exist)
                    f: body flag                                      [mxv]
give2           : list of specifications for secondary output         [mxv]
Grav            : Newton's constant of gravity                        [1]
root_center     : if given (3 numbers), forces tree-root centering    []
potname         : name of external potential                          []
potpars         : parameters of external potential                    []
```

[2]Called "YancNemo" in former versions of this package (before December 2002).

[3]Older versions of this package contained a non-`NEMO` code, called "YANC". This code was never properly tested and has hence been deprecated.

```
potfile          : file required by external potential                [] 
startout         : primary output for t=tstart?                       [t]
lastout          : primary output for t=tstop?                        [t]
VERSION          : 30-apr-2004 Walter Dehnen                          [2.1Igcc-3.2]
COMPILED         : May 20 2004, 22:45:18, with gcc-3.2 []
STATUS           : public version []
```

The last column indicates the default value, with '`[???]`' indicating that the value for the keyword must be given, while '`[]`' means that the corresponding feature is not used by default. In order to get a detailled explanation of the various options, see the manpage of `gyrfalcON`.

Traditionally on linux systems, there is a limit of 2Gb on the size of files. This will cause trouble with `NEMO` snapshot files, since the snapshots of all output times are written to one file. To overcome this, you must (i) configure `NEMO` appropriately (use `configure --enable-lfs` when installing and (ii) ensure that your file systems supports large files – consult your system administrator.

10.1 An Example

In order to integrate a Plummer sphere with $N = 10^5$ particles, you may issue the command

```
mkplummer - 100000 seed=1 scale=1 | gyrfalcON - plum.snp tstop=10 eps=0.1
```

which first creates initial conditions from a Plummer model, which are then piped into `gyrfalcON`. `gyrfalcON` creates an output file '`plum.snp`' containing output every full time unit until time $t = 10$. The log output looks like[4]

```
# ------------------------------------------------------------------------------------------------------------------------
# "gyrfalcON - plum.snp tstop=10 eps=0.1 VERSION=2.1Igcc-3.2"
#
# run at  Mon May 17 17:58:12
#     by  "wd11"
#     on  "andromeda.star.le.ac.uk"
#     pid  27577
#
#    time       E=T+V        T          V_in        W         -2T/W     |L|      |v_cm|   tree  grav  step  accumulated
# ------------------------------------------------------------------------------------------------------------------------
 0.0000     -0.1469416    0.14778    -0.29472    -0.29341    1.0073  0.0010751  6.3e-09   0.12  1.01  1.14   0:00:01.14
 0.015625   -0.1469415    0.14778    -0.29472    -0.29341    1.0073  0.0010751  6.2e-09   0.07  1.00  1.10   0:00:02.27
 0.031250   -0.1469419    0.14778    -0.29472    -0.29341    1.0073  0.0010751  6.3e-09   0.08  1.00  1.10   0:00:03.39
 0.046875   -0.1469420    0.14778    -0.29472    -0.29341    1.0073  0.0010751  6.2e-09   0.07  0.91  0.99   0:00:04.41
.
.
.
 9.9531     -0.1469412    0.14517    -0.29211    -0.29084    0.99828 0.0010775  6.6e-09   0.06  1.00  1.09   0:11:38.86
 9.9688     -0.1469415    0.14518    -0.29212    -0.29085    0.99829 0.0010775  6.6e-09   0.07  0.99  1.07   0:11:39.95
 9.9844     -0.1469413    0.14518    -0.29212    -0.29085    0.99831 0.0010776  6.6e-09   0.06  1.00  1.08   0:11:41.05
 10.000     -0.1469408    0.14518    -0.29213    -0.29085    0.99833 0.0010776  6.5e-09   0.07  1.00  1.09   0:11:42.16
```

The column `|v_cm|` gives the center-of-mass motion, which stays constant (within floating point precision) due to the momentum-conserving nature of `falcON`. The last four columns contain the CPU time in seconds spent on the tree building, force computation, and full time step, as well as the accumulated time.

11 Additional `NEMO` Programs

Note that all `NEMO` programs have a help utility: when calling them with the option `help=h` a listing of their options is printed. If a name for an I/O file is given as '`-`', the program will instead read from `stdin` or write to `stdout`, which allows piping into another program. When an output file name reads '`.`', it is interpreted as sink, i.e. no output is ever made.

Below, we give a short summary of the programs. For more details, see the relevant man page(s).

11.1 Computing Gravity

The program `addgravity` adds gravitational potential and acceleration to. `getgravity` computes the gravity generated by one set of particles (source) at the positions of another (usually smaller) set (sinks). This is useful, for instance, for computing the rotation curves of N-body galaxies.

[4] Code compiled with `gcc` version 3.2.2, run on an AMD Athlon(TM) XP 1800+ with 1529Mhz and 256Kb cache size.

11.2 Creating Initial Conditions

11.2.1 mkdehnen

This program creates initial conditions from an isotropic spherical Dehnen (1993) model, which has density

$$\rho(r) = \frac{3-\gamma}{4\pi} \frac{M\, r_s}{r^\gamma (r+r_s)^{4-\gamma}}. \tag{2}$$

11.2.2 mkking

This program creates initial conditions from a spherical King model of single-mass stars.

11.2.3 mkplum

This program creates initial conditions from a spherical Plummer model with isotropic velocities.

11.3 Manipulating Snapshots

These programs read a stream of NEMO snapshots, manipulate each of them, and write out another stream of NEMO snapshots. Both in and output may be either file or pipe. All of these programs have the following keywords in common. in and out specify the in and output streams, times (defaulting to times=all) specifies the times of the snapshots to be read, manipulated, and written out.

11.3.1 density_centre

This program (public since May 2004, previously proprietary as 'center') iteratively finds the position of the (global) density maximum and optionally centers the snapshot on this position. More specifically, it finds the position $\boldsymbol{x}_c$ where

$$\rho_h(\boldsymbol{x}_c) \equiv h^{-3} \sum_i m_i\, W\left[\frac{|\boldsymbol{x}_c - \boldsymbol{x}_i|}{h}\right] \tag{3}$$

has a global maximum. Here, the smoothing kernel is a Ferrers $n = 3$ sphere: $W(r) \propto (1-r^2)^3$ for $r < 1$ and $W = 0$ otherwise.

Note that this program is useful for isolated galaxies only.

11.3.2 symmetrize

(public since May 2004) This simple program may be used to symmetrize a snapshot with respect to the origin ($\boldsymbol{x} = \boldsymbol{v} = 0$, and, possibly, the equator ($z = 0$ plane).

Note that this program is useful for isolated galaxies only.

11.4 Analyzing Snapshots On or Off Line

These programs read in a stream of snapshots, analyze each of them, and write diagnostics output. Optionally, they also write the snapshots out. This option allows to have several analysis tools piped one after the other (the user has to take care that each receives the proper type of body data).

11.4.1 density_centre

(see §11.3.1) If the snapshot is not centred, density_centre acts as an analysis tool only.

11.4.2 lagrange_radii

(public since May 2004, previously proprietary as 'lagrange_rad') Given a set of fractions $\in (0, 1)$, the radii (w.r.t. the origin) containing corresponding fraction of the total mass are computed and written in ASCII format to a file. In order to center the snapshot, use density_centre before. This program is much faster than a global sort on the radii.

Note that this program is useful for isolated galaxies only.

12 Bugs and Features

12.1 Test-Particles

`falcON` does not support the notion of a test particle, i.e. a body with zero mass. Such bodies will never get any acceleration (that is because the code first computes the force, which is symmetric and hence better suited for mutual computations, and then divides by the mass to obtain the acceleration). To overcome this, you may use tiny masses, but note that the forces created by such light bodies will be computed, even if they are tiny, and contribute to the computational load.

Actually, this is exactly what we do in `getgravity`.

12.2 Bodies at Identical Positions

The code cannot cope with more than `Ncrit` bodies at an identical position (within floating point accuracy). Such a situation would result in an infinitely deep tree; the code aborts with an error message.

12.3 Unknown Bugs

A bug that lead `falcON` or `gyrfalcON` to occasionally crash with 'Segmentation fault' I have recently tracked down and debugged (as of 3rd April 2003). However, there may perhaps still to be similar bugs around, which are not reproducible and hence hard to track down and weed out. Measures have been taken to solve such problems eventually. If you ever encounter a problem that you think might be a bug and which is not mentioned in this documentation, please report it to me (`walter.dehnen@astro.le.ac.uk`). Thanks.

References

Dehnen, W., 1993, MNRAS, 265, 250
Dehnen, W., 2000, ApJ, 536, L39
Dehnen, W., 2001, MNRAS, 324, 273
Dehnen, W., 2002, JCP, 179, 27