User Guide for falcON

version of November 7, 2008
Walter Dehnen

Summary. falcON is the "Force Algorithm with Complexity $\mathcal{O}(N)$" which is described by Dehnen (2000, 2002). With this package, 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 the .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 Recent changes

This section list some changes made in the last two years or so.

September 2007 The code, described in McMillan & Dehnen (2007) for creating initial conditions for a disc galaxy consisting of spheroid (halo & bulge) and disc components has been included. It comes with its own detailed user guide and example script (`doc/mkGalaxy`).

May 2007 Codes for generating spherical initial conditions allow for anisotropic velocities according to Ossipkov (1979) and Merritt (1985).

February 2007 Three new NEMO executables have been added to the package: **g2s**, **s2g**, and **s2s**. The first two provide quite nifty conversion between gadget data files and NEMO snapshots, see man pages for more details, while **s2s** is very similar to NEMO's **snapcopy**, except that it can copy all data supported by falcON I/O.

August 2006 The public version of falcON has been put under CVS control under the NEMO package. This should significantly simplify installation, updating and downloading.

Some changes in the directory structure and filenames (`FAlCON`→`forces`).

July 2006 In order to allow their usage at the GH2006 summer school, several parts of the code have been migrated from the proprietary section into the public part. Of particular interest are extensions to the manipulators and support for communication between manipulators, as well as `bodyfunc.h`, which supports simple functions of a body (and time + parameters) to be generated on the fly given a user-provided string in some simple syntax (see man pages `(1bodyfunc)` and `(5bodyfunc)`; this is very similar to, but more powerful than, NEMO's `bodytrans`). Parts of the code have been documented using *doxygen*, see `falcON/dox/htlm/index.htlm` for details (partly due to my inexperience with *doxygen* this documentation is very preliminary and incomplete, but should still be useful for anybody using the falcON library or using manipulators).

April 2006 Some pieces of code which are not falcON specific have been taken out and put into a `utils` package, which resides in directory `utils` parallel to directory `falcON`, but there are some dynamic links. The `utils` package uses namespace `WDutils` and provides some generally useful code. It must be compiled **before** falcON.

4 Installation

Since August 2006, (the public version of) falcON is under CVS control within the NEMO package and you are strongly advised to install and initialize falcON using NEMO.

If you are interested in using falcON standalone, i.e. only use it as a library for your force computation or SPH neighbour search, then you may still obtain falcON from NEMO, but must compile it without the `NEMO` enviroment variable set. In this case, only two executables will be made: **`TestGrav`** and **`TestPair`**, the first allows testing of the gravity approximation, the second the search for SPH interaction partners.

Compiler Issues

You need to make the library `libfalcON.so` and, possibly, the executables you want to use. The code is written entirely in C++ and it is strongly recommended to use a compiler that understands standard C++[1], I recommend GNU's `gcc` versions 3.4 or Intel's `icc` version 8.0 or higher. By default, we use gcc, if you want to use another compiler, edit the file `makedefs` and change the entry for `COMPILER`.

You may also edit the optimization flags in file `makedefs`. This applies particularly to Intel's `icc` compiler, which allows processor specific switches.

5 Individual Softening Lengths

Individual softening lengths are enabled, but not obligatory (in fact default is always to have a globally constant ϵ).

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[2]

[1] If you use a compiler version different (usually newer) than those against which this package was tested, you may get warnings or even error messages. These do not point to genuine errors in the code but rather reflect the fact that C++ compilers do not fully implement the standard but converge there with every new version. I would appreciate if you, in such a case, could email me the error messages together with details of the compiler and system used.

[2] Code compiled with `gcc` version 3.3.4, run on an AMD Opteron (tm) with 2190Mhz and 1024Kb cache size.

```
time needed for set up of X_i:                 0.64
time needed for FAlCON::grow():                1.21
time needed for FAlCON::grow():                0.59
time needed for FAlCON::approximate_gravity(): 4.77

state:                 tree re-grown
root center:           0 0 0
root radius:           1024
bodies loaded:         1000000
total mass:            1
N_crit:                6
cells used:            353665
of which were active   353665
maximum depth:         21
current theta:         0.6
current MAC:           theta(M)
softening length:      0.01
softening kernel:      P1
Taylor coeffs used:    83880 in 4 chunks of 22108
interaction statitics:
    type          approx   direct      total
# body-body :          -        0          0 =        0%
# cell-body :    2138354   484564    2622918 =   18.427%
# cell-cell :   11302423   254979   11557402 =   81.194%
# cell-self :          -    53928      53928 =    0.379%
# total     :   13440777   793471   14234248 =  100.000%

ASE(F)/<F^2>      = 0.001392818172
max (dF)^2        = 0.8913656473
Sum m_i acc_i     = 3.240235305e-10 1.787507139e-09 2.572656954e-09
```

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 ($x := r/\epsilon$)

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.

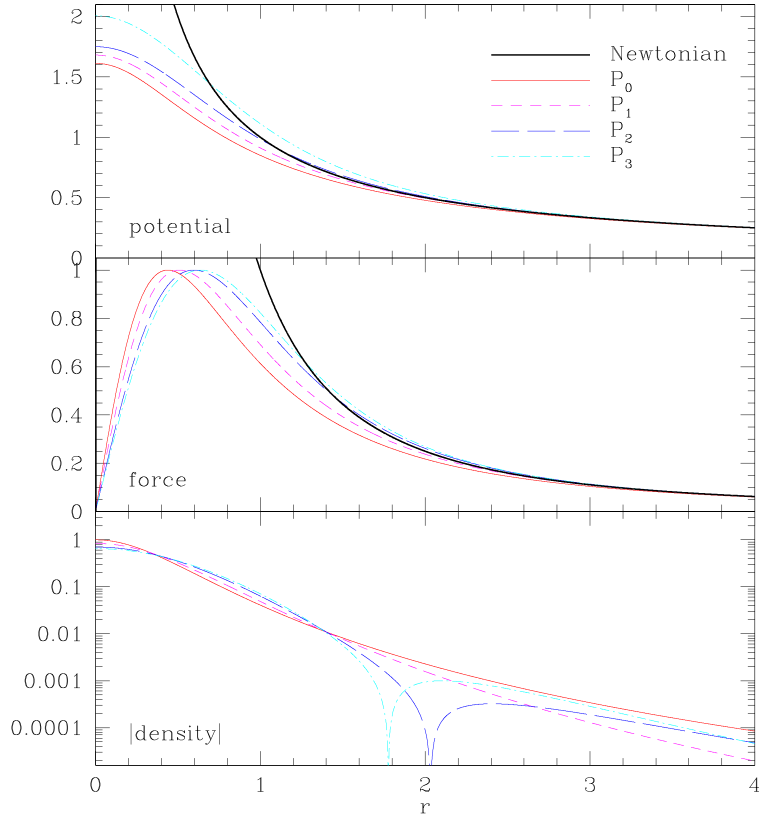

Figure 1: Potential, force, and density for the softening kernels of the table, including the standard Plummer softening (P_0). The softening lengths ϵ are scaled such that the maximum force equals unity. The kernels $P_{>0}$ approach Newtonian forces more quickly at larger r than does P_0. The kernels 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.

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 to serve as a Poisson solver in your existing code.

9.1 With C++

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

```
#include <forces.h>
```

somewhere at the beginning of your C++ source code. Make sure that the compiler finds the file `forces.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 `forces.h` (don't forget that `class forces` lives in namespace `falcON`). In order to make an executable, add the linker options `-L $FALCONLIB /lib -lfalcON -lm` so that the library will be loaded at runtime.

For examples of code using `forces.h`, see the file `TestGrav.cc` in subdirectory `src/public/exe`, which may be compiled by typing `make TestGrav` 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 <forces_C.h>
```

somewhere at the beginning of your C source code. Make sure that the compiler finds the file `forces_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 `forces_C.h`. In order to make an executable, add the linker options `-L $FALCONLIB /lib -lfalcON -lstdc++ -lm` so that the library will be loaded at runtime.

For examples of code using `forces_C.h`, see the file `TestGravC.cc` in subdirectory `src/public/exe`, which may be compiled by typing `make TestGravC` 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 'forces.f'
```

somewhere at the beginning of your FORTRAN program. Make sure that the compiler finds the file `forces.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 `forces.f`. In order to make an executable, add the linker options `-L $FALCONLIB /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/public/exe`, 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)[3]. 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[4], 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** without arguments or with the argument `help=h` produces the following overview over the options.

```
gyrfalcON -- a superb N-body code
```

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

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

```
option summary:
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                  [0.60]
hgrow            : grow fresh tree every 2^hgrow smallest steps    [0]
Ncrit            : max # bodies in un-split cells                  [6]
eps              : >=0: softening length
                   < 0: use individual fixed softening lengths     [0.05]
kernel           : softening kernel of family P_n (P_0=Plummer)    [1]
kmax             : tau_max = (1/2)^kmax  MUST be given             [???]
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.                 []
time             : time of input snapshot (default: first)         []
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.                  [mxv]
give2            : list of specifications for secondary output     [mxv]
Grav             : Newton's constant of gravity (0-> no self-gravity)  [1]
root_center      : if given (3 numbers), forces tree-root centering   []
accname          : name of external acceleration field             []
accpars          : parameters of external acceleration field       []
accfile          : file required by external acceleration field    []
manipname        : name of run-time manipulator                    []
manippars        : parameters for manipulator                      []
manipfile        : data file required by manipulator               []
manippath        : path to search for manipulator                  []
manipinit        : manipulate initial snapshot?                    [f]
startout         : primary output for t=tstart?                    [t]
lastout          : primary output for t=tstop?                     [t]
VERSION          : 28-feb-2007 Walter Dehnen                       [3.0.9I]
COMPILED         : Feb 28 2007, 09:11:29, with gcc-3.3.5           []
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 (1) create a Hernquist model with $N = 10^6$ particles that are initially symmetric w.r.t. origin, and (2) integrate it for 10 time units (using units that imply $G = 1$, $r_s = 1$, and $M = 1$) using global softening length of $\epsilon = 0.01$ and five adaptive time steps with $\tau_{\min} = 1/128$, $\tau_{\max} = 1/8$, and $\tau = 0.01 \min\{|\boldsymbol{a}|^{-1}, |\Phi|^{-1}\}$, you may issue the commands

```
mkdehnen   out=- nbody=500000 gamma=1 seed=1 q-ran=t  | \
symmetrize in=- out=- use=1 copy=1                    | \
gyrfalcON  in=- out=D.snp tstop=10 eps=0.01             \
           kmax=3 Nlev=5 fac=0.01 fph=0.01 logfile=D.log
```

gyrfalcON creates an output file '`D.snp`', containing output every full time unit until time $t = 10$, and a logfile '`D.log`' which looks like this [5].

```
# ------------------------------------------------------------------------------------------------------------------------------------------------------------
# "gyrfalcON in=- out=D.snp tstop=10 eps=0.01 kmax=3 Nlev=5 fac=0.01 fph=0.01 logfile=D.log"
#
# run at  Wed Feb 28 09:20:39
#     by  "wd11"
#     on  "virgo"
#     pid  5076
#
#    time        E=T+V         T          V_in        W          -2T/W      |L|       |v_cm|     1/8    1/16    1/32    1/64    2^-7 l2R  D  tree  grav  step  accumulated
# ------------------------------------------------------------------------------------------------------------------------------------------------------------
 0.0000     -0.08321548   0.083424   -0.16664    -0.16663    1.0013  0.00074273 0.0      211908  188264  299582  291324    8922  10 20  0.85  8.65  9.69   0:00:09.69
 0.12500    -0.08321620   0.083430   -0.16665    -0.16663    1.0014  0.00074272 1.1e-09  217168  183320  299938  288448   11126  10 20  9.45 47.25 58.50   0:01:08.20
 0.25000    -0.08321624   0.083426   -0.16664    -0.16663    1.0014  0.00074273 2.7e-09  212052  188749  300483  285388   13328  10 20  9.50 48.66 59.84   0:02:08.03
 0.37500    -0.08321605   0.083416   -0.16663    -0.16662    1.0013  0.00074275 2.7e-09  217060  184049  301008  282689   15194  10 20  9.37 48.89 59.97   0:03:08.01
.
.
.
 9.6250     -0.08321518   0.083110   -0.16633    -0.16631    0.99946 0.00074519 3.0e-08  213894  190202  297691  275447   22766  10 20  9.34 50.63 61.70   1:18:59.50
 9.7500     -0.08321520   0.083112   -0.16633    -0.16631    0.99946 0.00074510 2.0e-08  214934  189158  297532  275638   22738  10 20  9.36 50.62 61.70   1:20:01.20
 9.8750     -0.08321508   0.083122   -0.16634    -0.16632    0.99953 0.00074507 2.3e-08  213896  190095  297599  275605   22805  10 20  9.37 50.59 61.68   1:21:02.89
 10.000     -0.08321484   0.083143   -0.16636    -0.16634    0.99967 0.00074509 2.9e-08  214952  189055  297768  275387   22838  10 20  9.36 50.53 61.61   1:22:04.51
```

The first eight columns give the simulation time, total energy (which changed only by 0.002%), kinetic energy, internal gravitational energy $V_{\rm in} \equiv (\sum_i m_i \Phi_i)/2$, and $W \equiv (\sum_i m_i \boldsymbol{x}_i \cdot \boldsymbol{a}_i)/2$, virial ratio, absolute total angular momentum, and absolute center-of-mass motion. The latter hardly changes due to the momentum-conserving nature of falcON (when integrating with a single time step, the center of mass will not move within floating point precision). The next five columns give the number of bodies that move with the given time step. Usually, these numbers adjust from the initial assignment within a few blocksteps. The columns `l2R` and `D` give $\log_2$ of the root radius (always an integer) and the tree depth. 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 (in h:min:sec format).

Note that in absence of external gravitational forces, both $V_{\rm in}$ and W measure the gravitational potential energy, but in two different ways. Only if gravity is exactly Newtonian (no softening), do they agree. Thus, the difference between the two is indicative of the bias in the estimated gravity. This true for all types of N-body codes, not just **gyrfalcON**.

The CPU time consumption in the above example corresponds to about 4sec per shortest time step for $N = 10^6$.

[5] Code compiled with `gcc` version 3.4, run on a laptop with Pentium-M 735 with 1.7Ghz, 64kb cache size, and 1Gb RAM.

11 Additional NEMO-type Programs in falcON

Note that all NEMO-type programs have a help utility: when calling them without argument or 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. If an output file name is appended with a '`!`' or '`?`' any existing file of the same name (without this appendix) is overwritten or appended to, respectively.

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 existing snapshot(s). **`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.

11.2 Creating Initial Conditions

Since May 2007, the programs **`mkdehnen`**, **`mkplum`** and **`mkhalo`** create radially anisotropic models of Ossipkov (1979) -Merritt (1985) type. To this end, the parameter `r_a=` is taken as the anisotropy radius. For these models, the Binney anisotropy parameter has radial run

$$\beta \equiv 1 - \frac{\sigma_\theta^2}{\sigma_r^2} = \frac{r^2}{r_a^2 + r^2}. \tag{2}$$

11.2.1 `mkdehnen`

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

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

11.2.2 `mkhalo`

This program creates initial conditions with density

$$\rho(r) = \frac{C\,\mathrm{sech}(r/r_t)}{x^{\gamma_0}(x^\eta + 1)^{(\gamma_\infty - \gamma_0)/\eta}} \qquad \text{with} \qquad x = \frac{\sqrt{r^2 + r_c^2}}{r_s}, \tag{4}$$

where the constant C is determined such that the total mass equals M. The velocities are drawn from a Cuddeford (1991) distribution function

$$f(E,L) = L^{-2b}\, g\left(-E - \frac{L^2}{r_a^2}\right), \tag{5}$$

which is a generalisation of the Ossipkov (1979) - Merritt (1985) models and gives rise to a Binney anisotropy parameter

$$\beta \equiv 1 - \frac{\sigma_\theta^2}{\sigma_r^2} = \frac{r^2 + b\,r_a^2}{r^2 + r_a^2}. \tag{6}$$

`mkhalo` also allows for a spherical external potential, which is added to the gravity generated by the density (4) when computing the equilibrium model. For more details and how to use **`mkhalo`** in generating McMillan & Dehnen (2007) galaxy models, see the separate documentation.

11.2.3 `mkking`

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

11.2.4 `mkplum`

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

11.3 Manipulating and Analysing 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.

It is recommended to manipulate snapshots on- and off-line using manipulators (see section 12).

11.3.1 manipulate

This program (public since May 2005) initiates an off-line analysis via manipulators (see section 12).

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 Data format conversion

11.4.1 s2a and a2s

These two little programs allow simple conversion from and to ASCII tables.

11.4.2 g2s and s2g

These two programs convert NEMO snapshots to and from gadget data files. Different endianess is discovered and corrected for on the fly.

11.4.3 s2s

This simple program converts snapshots to snapshots. The user can specify which times and which data are copied. The current version does not support the possibility to only copy a sub-set of all the bodies.

12 Run-Time Manipulators

Run-time manipulators have been added in September 2004. They are functions (in fact C++ classes) that operate on the snapshot data and are called in **gyrfalcON** (for on-line analysis or manipulation) after every complete blockstep or in **manipulate** (for off-line analysis or manipulation) for every snapshot stored in file. Run-time manipulators are loaded at run-time, according to the keywords `manipname`, `manippars`, `manipfile`, and `manippath`, and hence no recompilation of the code is required. Up to 100 Manipulators can be concatinated, i.e. one called after the other, by using `manipname=`*name1*+*name2* etc., `manippars="`*pars1*`;`*pars2* etc., and `manipfile="`*file1*`;`*file2*`"`[6]. Alternatively, if only a `manipfile` (but no `manipname`) is given, that file is interpreted to contain the name, parameters and filenames of manipulators. Such a file may look like this:

```
#
# manipulators for simulation
#
  name1   file1   # manipulator name1 takes no parameter, but needs file

  name2   0,1,5   # manipulator name2 requires 3 parameters.
#
```

Lines whose first non-space character is '#' are ignored and anything following a free (i.e. preceeded by space) '#' is also ignored (can be used to put comments into the file).

The potential use of run-time manipulators is almost unlimited (one may use them for on-line analysis as well), and the number of manipulators coming with the package is growing

Manipulators can communicate data via a pointer bank supported by class snapshot. Supopse you want to compute the lagrange radii of all bound ($E < 0$) stars (particles with $i < N_\star$) w.r.t. the position of the most bound star particle. While it is possible to write a single manipulator for this job, it is much more convenient to use three existing manipulators which communicate their data; the input file may look like this:

[6] In previous (pre July 2006) versions, the characeter '#' was allowed as a separator of file names or parameters. This is no longer supported, but instead of the semicolon, you can also use a space. In either case, the whole list of parameters or file names must be given in quotes.

```
#
# 1 pick subset: all bound with i<10000
  set_subset  10000 (i<#0)&&(E<0)
# 2 set centre to position of most bound particle in subset
  bound_centre
# 3 compute lagrange radii of subset w.r.t.\ centre
  lagrange    0.01,0.03,0.05,0.1,0.3,0.5,0.7,0.9,0.95 simul.rad
#
```

A detailled *doxygen* documentation for each of the manipulators listed below can be found at `falcON/dox/html/namespacefalcON_1_1Manipulate.html`

12.1 `set_subset`

This manipulator creates a filter from a boolean `bodyfunc` expression, see man pages (`5bodyfunc`), given in `manipfile`, with parameters from `manippars` (an empty expression makes the open filter: all bodies are accepted). At each manipulation, bodies not passing the filter are flagged to be ignored, while those passing the filter will be flagged not to be ignored. This is equivalent to the sequence of manipulators `set_filter` and `use_filter`.

A simple usage is to restrict the range of bodies. For instance the bodyfunc expression "`#0<=i&&i<#1`" filters bodies with index between parameters `#0` and `#1` (note that the the indices of bodies may not be preserved; in this case use the key instead, that is '`k`' instead of '`i`' in the expression).

12.2 `set_filter`

This manipulator creates a filter from a boolean `bodyfunc` expression, see man pages (`5bodyfunc`), given in `manipfile`, with parameters from `manippars` (an empty expression makes the open filter: all bodies are accepted). The (pointer to the) filter is then registered under the key '`filter`' with the pointer bank.

12.3 `use_filter`

This manipulator uses a filter, registered under the key '`filter`' to flag bodies not passing the filter to be ignored.

12.4 `dens_centre`

This manipulator iteratively finds the position of the (global) density maximum of the defined subset (only bodies not flagged to be ignored, default: all). More specifically, it finds the position $\boldsymbol{x}_c$ where

$$\rho_h(\boldsymbol{x}_c) \equiv h^{-3} \sum_i m_i \, W \left[\frac{|\boldsymbol{x}_c - \boldsymbol{x}_i|}{h} \right] \tag{7}$$

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. The position $\boldsymbol{x}_c$ and the corresponding velocity $\boldsymbol{v}_c$ are then registered under the keys '`xcen`' and '`vcen`', respectively.

12.5 `set_centre`

This manipulator sets '`xcen`' to the position (p_0, p_1, p_2) and '`vcen`' to (p_3, p_4, p_5). If no parameters are given, '`xcen`' and '`vcen`' are deleted, implying using the origin instead for most other manipulators.

12.6 `bound_centre`

This manipulator sets '`xcen`' and '`vcen`'. The algorithm is as follows. First, the most bound body in the subset (default :all) is found. Next, its K (default: 256) nearest neighbours (including itself) from the subset are sought. Finally, the centre is evaluated as weighted average of the positions and velocities of the $K/8$ (at least 4) most bound of these K bodies with weight

$$(E_{\max} - E_i)^\alpha \tag{8}$$

where α is a second parameter (default: 3) and $E_{\max}$ is the energy of the least bound of the K bodies.

12.7 lagrange

This manipulator computes the Lagrange radii at masses given by total mass times the parameters and writes them to the file. Only bodies in the defined subset (those not flagged to be ignored, default: all) are used. The centre is taken to be that found under the key 'xcen' (default: origin).

12.8 density

This manipulator estimates for each body in the defined subset (default: all) the mass density via a kernel estimator from its K nearest neighbours:

$$\rho_i = h^{-3} \sum_j m_j \, W \left[\frac{|\boldsymbol{x}_i - \boldsymbol{x}_j|}{h} \right] \tag{9}$$

with h the distance to the Kth neighbour and $W(r) \propto (1 - r^2)^n$ for $r < 1$ and $W = 0$ otherwise. K and n are parameters and default to 32 and 1, respectively. The computation of the density is computationally expensive (more so than a time step of **gyrfalcON** without individual time steps). Therefore, the time between manipulations can be given as third parameter (the simulation time of the actual density estimation is registered under the key 'trho')

12.9 sphereprof

This manipulator estimates the radial profiles of density, velocity, velocity dispersion, axis ratios, and orientations from spherical binning of all bodies in the defined subset (those not flagged to be ignored, default: all) w.r.t. position and velocity 'xcen' and 'vcen' (defaults: origin). This is similar to, but more powerful than, the NEMO program radprof. **Note** that axis ratios obtained for non-spherical distributions are bound to be incorrect.

12.10 densprof

This manipulator estimated the centre as well as mean and dispersion velocity, axis rations, and orientations of density shells, by binning all bodies in the defined subset into density bins. This requires a preceding manipulator density, but gives reliable axis ratios (as opposed to sphereprof). The only parameter is the number of bodies per density bin. If a time is registered under the key 'dens_time' and does not match the actual simulation time, nothing is done.

12.11 projprof

This manipulator estimates the projected radial profiles of surface density; mean, rotational, and dispersion of the line-of-sight velocity; the axis ratio; position and rotation angle for all bodies in the defined subset (those not flagged to be ignored, default: all) w.r.t. position and velocity 'xcen' and 'vcen' (defaults: origin).

12.12 symmetrize_pairs

Assuming the initial conditions were forced to be reflection symmetric w.r.t. the origin (using symmetrize), there is no guarantee that this symmetry will be preserved during a simulation, since truncation errors eventually break the symmetry. To avoid this, you may want to enforce symmetry by keeping pairs of particles symmetric w.r.t. the origin. Exactly this is done by symmetrize_pairs for body pairs with the first body being in the subset (those not flagged to be ignored, default: all).

12.13 randomize_azimuth

Suppose you want to suppress any non-axisymmetric perturbations, such as a bar mode or spiral waves. One way to do this is to randomize the disk bodies' azimuths every blockstep. Exactly this is done by randomize_azimuth for bodies in the subset (those not flagged to be ignored, default: all) using a random seed given by p_0 (if $p_0 = 0$, a random seed is generated from the current time).

12.14 add_plummer

You may add new bodies to the simulation. add_plummer is a very simple manipulator that at each time step adds one new body, drawn from a Plummer sphere, to the simulation. If you fancy to design similar manipulators, you are recommended to model them on add_plummer.

13 Bugs and Features

13.1 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`.

13.2 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. When this has occured in the past, it was often because the body positions were faulty, either all zero, `nan`, or `inf`.

13.3 Unknown Bugs

A bug that lead falcON or **`gyrfalcON`** to occasionally crash with 'Segmentation fault' I have 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

Cuddeford P., 1991, MNRAS, 253, 414

Dehnen, W., 1993, MNRAS, 265, 250

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

Dehnen, W., 2001, MNRAS, 324, 273

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

McMillan, P. J., Dehnen, W., 2007, MNRAS, 378, 541

Merritt, D., 1985, AJ, 90, 1027

Ossipkov, L.P., 1979, Pis'ma Astron. Zh., 5, 77