User Guide for falcON

version of July 11, 2006

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 What is new?

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

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.

September 2005 A lot of changes have been introduced with this release, some of them affecting the users, for instance the change of the namespace's name from `nbdy` to `falcON`. Several files have been re-named to better describe their contents.

The body data layout has been changed so that adding and removing bodies becomes possible (the files `body.h` & `body.cc` and `io.h` & `io.cc` (previously `nmio.h` & `nmio.cc`) have been completely re-written). In this process, two new routines `get_data_blocked()` and `put_data_blocked()` have been added to `NEMO`.

When a `falcON` executable, like `gyrfalcON`, is run without any command-line arguments, the `help=h` argument is assumed. Output file names may be appended with an '`!`' or '`?`' to indicate that existing files shall be overwritten or appended to, respectively.

To startup `falcON`, the shell-script file `falcON_start` should be sourced, which also tries to start `NEMO`. While the `falcON` library is still dynamically linked, there is no need anymore to set the `LD_LIBRARY_PATH` environment variable (the executables know where to link to).

For `FORTRAN` and C users, the ordering of vector-typed arrays has been changed: where formerly three arrays of size N for the x, y, and z components were required, we now expect one array of size $3N$ with the order $x_0, y_0, z_0;\ x_1, y_1, z_1 \ldots$.

September 2004 The time integrator has been completely re-written (used to be in `inc/nbdy.h`; now in `inc/public/nbody.h`). Tests indicate that it behaves as the old one (but is easier to maintain and extend).

`gyrfalcON` allows now for *run-time manipulators*, see §12 below and also the `man` page for `gyrfalcON`. For them to work, you need the enviroment variable `FALCON` to be defined, see §4.2.

1st July 2004 The `gyrfalcON` options `potname`, `potpars` and `potfile` have been replaced by `accname`, `accpars` and `accfile`, respectively. This reflects the change in using an external acceleration field rather than the old-style `NEMO` potential.

19th May 2004 All `NEMO` programs in this packages come with `man` pages, which replace the detailed documentation in this file.

14th May 2004 The `NEMO` programs `mkdehnen`, `mkking`, and `mkplum` have been added to the public version of this package, see §11.2 for details.

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.

September 2003 Individual, but fixed, softening lengths have been added to the public version. See §5 below.

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 Initializing falcON

To work properly, some parts of the package require several settings. These are provided by sourcing the file falcON_start in the falcON directory (edit this file before first usage and change the falcON directory appropriately). Alternatively to sourcing, you may add the line

```
alias falcON 'setenv FALCON $HOME/falcON; source $FALCON/falcON_start'
```

to your .tcshrc file and simply type falcON to initialize. The executables live in a subdirectory (but note that make tonemo copies them to the $NEMOBIN, see above)

$FLACON/$MACHTYPE_$OSTYPE,

which is pointed to by the enviroment variable FALCONBIN, 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 $FALCONBIN/lib) for several hosts on the same file system.

There is also a file falcON_end, which can be sourced to delete all falcON related entries from the enviroment. In fact, falcON_start defines the alias NOclaf (that is falcON backwards) to do exactly that.

4.3 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 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 make.defs and change the entry for COMPILER.

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

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).

4.4 Make utils

You first need to make the utils library, simply by typing make in the utils directory (which lives parallel to the falcON directory. utils is a self-contained package and does not require anything else to be preinstalled.

4.5 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, but if you use falcON_start that might already have started NEMO). Make sure you have a NEMO distribution that contains the routines get_data_blocked() and put_data_blocked() in file $NEMOING/filestruct.h. The various options for making are summarized below.

make all	makes the library and all executables
make man	copies man pages to NEMO
make manips	makes the run-time manipulators (see §12)
make	same as make all manips
make tonemo	same as make 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[1].

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

[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.

4.6 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.

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[2]

```
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.

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

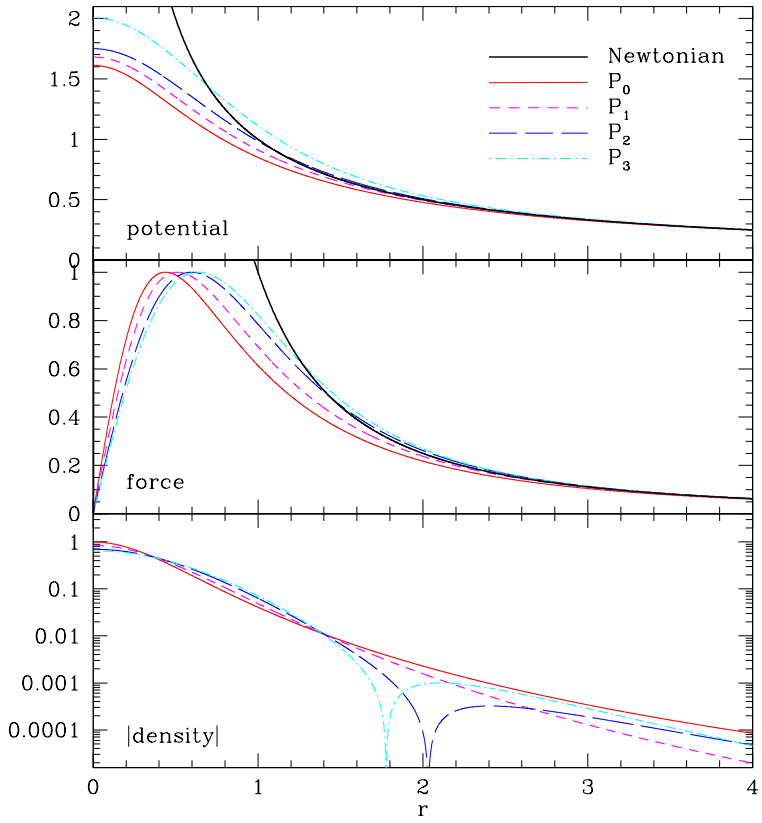

Figure 1: Potential, force, and density for the softening kernels of the table, including the standard Plummer softening (P_0). The softening lengths ϵ are scaled such that the maximum force equals unity. The kernels $P_{>0}$ approach Newtonian forces more quickly at larger r than does P_0. The kernels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

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. 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 `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 `FAlCON.h`, see the file `TestGrav.cc` in subdirectory `src/mains/`, 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 <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 $FALCONLIB/lib -lfalcON -lstdc++ -lm` so that the library will be loaded at runtime.

For examples of code using `FAlCON_C.h`, see the file `TestGravC.cc` in subdirectory `src/mains/`, 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 '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 $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/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`)[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 superberb N-body code

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                      [0.64]
hgrow            : grow fresh tree every 2^hgrow smallest steps        [0]
Ncrit            : max # bodies in un-split cells                      [16]
eps              : softening length                                    [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.                  []
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           []
```

[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.

```
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          : 22-mar-2006 Walter Dehnen                          [3.0.5PSI]
COMPILED         : Apr  6 2006, 13:46:07, with gcc-3.3.5              []
STATUS           : proprietary version; usage restricted              []
```

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 adaptive time steps with $\tau_{\rm min} = 1/128$ 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             \
           hmin=7 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=DS.snp tstop=10 eps=0.01 hmin=7 Nlev=5 fac=0.01 fph=0.01 logfile=DS.log VERSION=2.4Igcc-3.4"
#
# run at  Wed Sep 22 13:28:00
#     by  "wdl1"
#     on  "virgo"
#     pid  11564
#
#    time       E=T+V        T          V_in        W         -2T/W     |L|      |v_cm|     1/8    1/16    1/32    1/64    2^-7  tree  grav  step  accumulated
# ------------------------------------------------------------------------------------------------------------------------------------------------
 0.0000     -0.08322965   0.083410   -0.16664    -0.16663    1.0012  0.0012842  0.0       213490  187896  298948  290762    8904  0.99  8.37  9.58   0:00:09.58
 0.12500    -0.08323056   0.083422   -0.16665    -0.16664    1.0012  0.0012843  2.2e-09   218720  182980  299486  287664   11150 11.19 47.19 60.15   0:01:09.74
 0.25000    -0.08323026   0.083425   -0.16666    -0.16664    1.0013  0.0012842  1.8e-09   213636  188360  299961  284743   13300 11.16 48.29 61.11   0:02:10.86
 0.37500    -0.08323025   0.083422   -0.16665    -0.16664    1.0012  0.0012841  6.6e-10   218578  183742  300587  282059   15034 11.17 48.82 61.67   0:03:12.54
.
.
 9.6250     -0.08322973   0.083188   -0.16642    -0.16640    0.99985 0.0012829  7.8e-08   215630  189874  297455  273872   23169 11.32 50.93 63.98   1:21:47.22
 9.7500     -0.08323000   0.083185   -0.16642    -0.16640    0.99981 0.0012828  7.9e-08   216330  189142  297484  273889   23155 11.37 50.92 63.97   1:22:51.20
 9.8750     -0.08322990   0.083187   -0.16642    -0.16640    0.99983 0.0012827  8.2e-08   215680  189697  297464  274026   23133 11.36 50.84 63.91   1:23:55.12
 10.000     -0.08322997   0.083183   -0.16641    -0.16640    0.99981 0.0012825  8.0e-08   216340  188937  297452  274266   23005 11.34 50.85 63.90   1:24:59.03
```

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 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

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.

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 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).

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

11.4.1 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.

Note that the manipulator lagrange does exactly the same job and is more flexible in that the subset of bodies which are used can be defined more generally.